VXML Parse

Contents

[Solution Overview 3](#_Toc473966045)

[Get Latest VXML 4](#_Toc473966046)

[Process VXML 5](#_Toc473966047)

[Appendix A 10](#_Toc473966048)

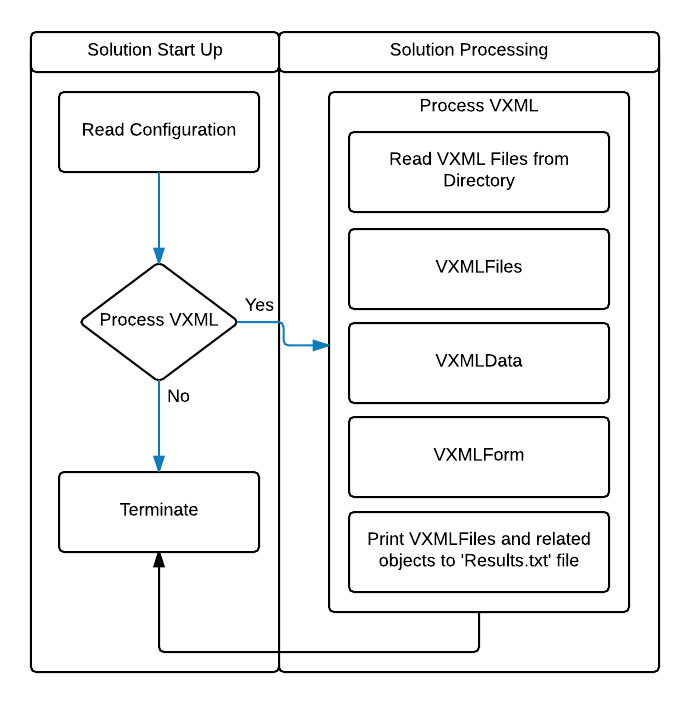
[Site Configuration Options 10](#_Toc473966049)

### Solution Overview

This is an executable java based application which can be executed from a command line.

The solution is to be developed in an operating system independent manner. So, any reference to file systems needs to be done using File.separator and so on.

Here is a high level overview of the required solution:



#### Solution Start Up

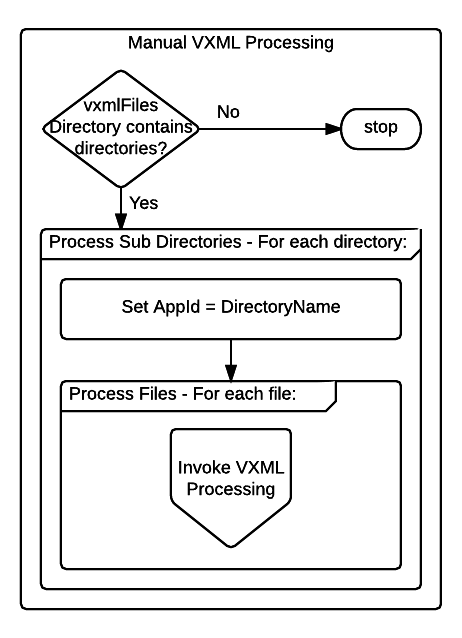
This will read a local configuration file for instructions detailed in this section. The solution will shut down when no processing is required. The solution will also shut down when it has fully completed a processing run.

#### Solution Processing

The main part of the utility application will read from zero to many files from a configured directory which have a ‘.vxml’ extension. The solution will process each file sequentially and construct and update VXMLFiles objects and related objects. Once all files are parsed the solution will print the objects to a ‘results.txt’ file and then exit.

### Get Latest VXML

The solution has a directory, ‘vxmlFiles’. This will contain from zero to many sub-directories. The name of the subdirectory will be a number which represents an ‘AppId’. Ie. 105. Beneath these sub-directories will be from zero to many files with a ‘.vxml’ extension. These are to be processed. The below image is a high level representation of the algorithm to use:



Example directory and file structure:

* D:\solution\vxmlFiles\112\CollectAuthenticationDetails.vxml
* D:\solution\vxmlFiles\112\CollectIntentAndProcess.vxml
* D:\solution\vxmlFiles\205\PaymentApplication.vxml

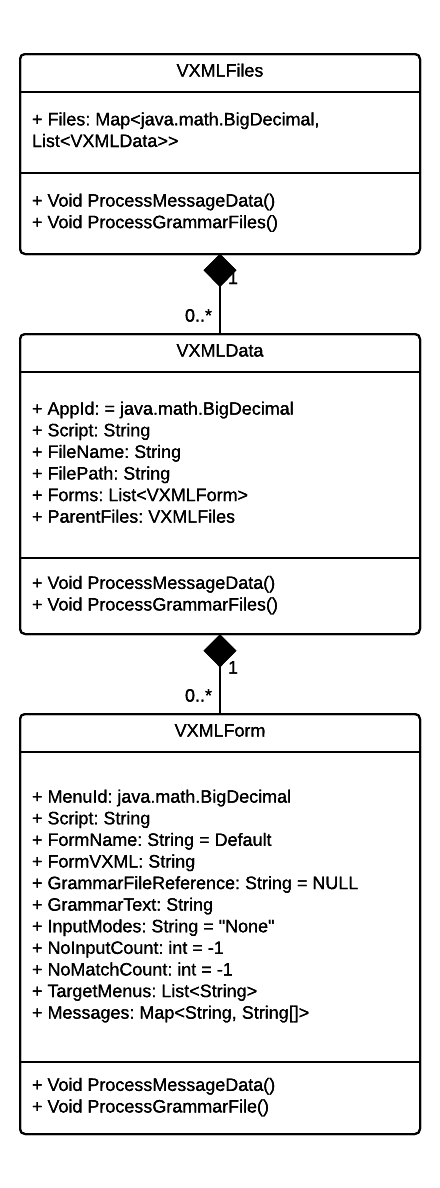
This example shows two vxml files associated with the ‘112’ application id and one for the ‘205’ application id.

For each vxml file found, we run the next step, Process VXML.

### Process VXML

For each VXML file we havewe conduct the following processing. This involves implementing an xml parser and searching for the defined element and attribute names defined below.

An example object which culd be used to store this data is shown here:



#### Object – VXMLFiles

This will simply be a Map where the key is the BigDecimal representing the application id (dbid / AppId) being processed. This is the value taken from the parent directory name.

Each VXMLData object represents the result of processing a single VXML file.

#### Object – VXMLData

This object contains the app id again, as described above and also a collection of VXMLForm objects as described below. Each VXMLForm object represents a single <form> element contained in a given VXML file.

Set FileName to the name of the vxml file being processed.

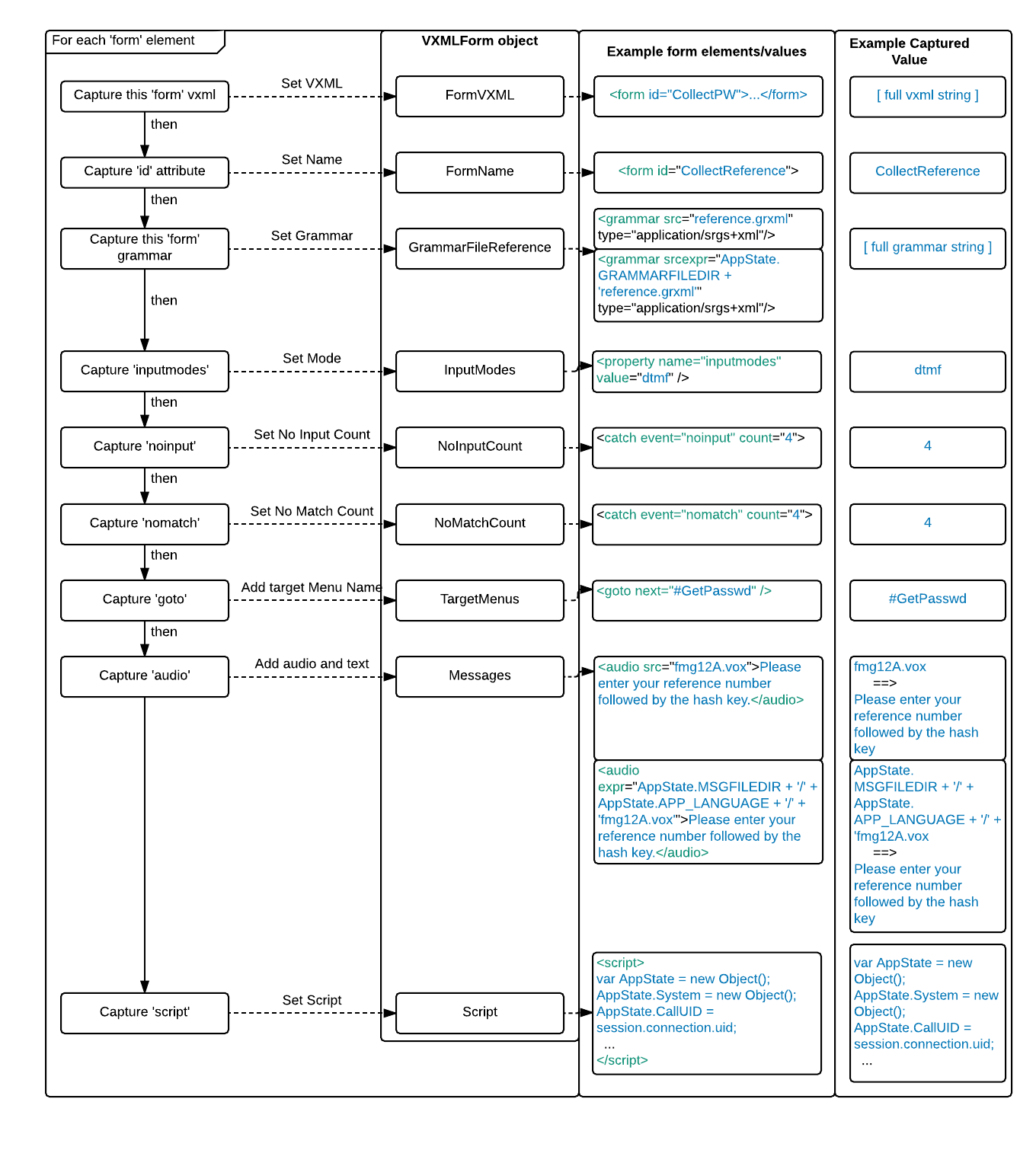
Set FilePath to empty string.

Set Script to the contents of a <script> element if present at the document level.

#### Object – VXMLForm

This object represents a single <form> element inside a given VXML file.

Each VXML file will be run through an XML parsing algorithm which will capture data to populate each VXMLForm object. The XML processing to populate this object can be visualised like this:



At the end of processing all VXML files we will have a VXMLFiles object which contains a collection of VXMLData objects, associated by the app id related to the EVENT\_LOGS record which was used to retrieve the VXML file.

At the end of processing a single VXML file we will have one VXMLData object and potentially many related VXMLForm objects.

The VMLForm object highlights that, for the grammar and audio file sections, we require post-parse processing. This is detailed further below in each associated section.

#### Script

Set Script to the contents of a <script> element if present at the <form>level.

#### Process Messages

The ‘audio’ element in a form may have either a ‘src’ or a ‘expr’ attribute. When it has a ‘src’ attribute the value will be a literal expression providing the relative location of a wave file. This can be set as the key to our Map.

When the attribute is the ‘expr’ attribute we need to do further processing as described here.

We invoke VXMLFiles.ProcessMessageData(). This will ensure that each VXMLForm.Messages object is processed and updated.

Initially, this Map simply contains a key which is potentially a file location and file name, with variables interspersed. The associated String array has one element and is the message text, if any. This can be represented structurally like this:

Map<file location and name, String[message text]>

In our ‘expr’ example above, the key and value of this entry is:

AppState.MSGFILEDIR + '/' + AppState. APP\_LANGUAGE + '/' + 'fmg12A.vox

Please enter your reference number followed by the hash key

We need to update this object to this form:

Map<file name, String[file location, message text]>

If, as in the above example, the ‘FileLocation’ attribute contains variables we will endeavour to de-reference these. In this example, we can see two variables:

* AppState.MSGFILEDIR
* AppState.APP\_LANGUAGE

When we detect variables, we refer to the value of the VXMLForm.Script parameter. If the value cannot be dereferenced here, we refer to the value of the VXMLData.Script parameter. Here is an example ‘script’ value:

var AppState = new Object();

...

AppState.APP\_LANGUAGE = initSystemTypeVariable('APP\_LANGUAGE','en-US');

…

AppState.MSGFILEDIR = '../Resources/Prompts';

...

The values we capture here are:

* AppState.MSGFILEDIR = ../Resources/Prompts
* AppState.APP\_LANGUAGE = en-US

This example demonstrates the two types of values you will need to handle. The first shows that the value is of the form: <text>(‘<text>’,’<value>’); where we capture the <value>. This should be processed using a regular expression which allows a variety of space characters, single and double quotes. The second example shows a simple quoted value. Again, account for spaces, single and double quotes.

The path to the file then is:

../Resources/Prompts/en-US/

The resulting key value pair then, in our above example, should be updated to:

Key =>fmg12A.vox

Value => [ ../Resources/Prompts/en-US/, Please enter your reference number followed by the hash key ]

#### Process Grammar References

The ‘grammar’ element in a form may have either a ‘src’ or a ‘srcexpr’ attribute. When it has a ‘src’ attribute the value will be a literal expression providing the relative location of a wave file. This can be set as the key to our Map.

When the attribute is the ‘srcexpr’ attribute we need to do further processing as described here.

Once we have this data, we invoke VXMLFiles.ProcessGrammarFiles(). This will ensure that each VXMLForm.GrammarFileReference is processed and updated to a clean relative url to the associated grammar file.

VXMLForm.GrammarFileReference contains either a relative url to a grammar file, or an expression to be dereferenced to a relative url. An example expression is:

AppState.GRAMMARFILEDIR + '/yesno.grxml'

In this example, we can see one variable:

* AppState.GRAMMARFILEDIR

When we detect variables, we refer to the value of the VXMLForm.Script parameter. If the value cannot be dereferenced here, we refer to the value of the VXMLData.Script parameter. Here is an example ‘script’ value:

var AppState = new Object();

…

AppState.GRAMMARFILEDIR = initSystemTypeVariable(‘GRAMMARFILEDIR’,'../Resources/Grammars’);

...

Use the same logic implemented for the audio element.

The values we capture then are:

* AppState.GRAMMARFILEDIR = ../Resources/Grammars

The relative path then is:

../Resources/Grammars/

The resulting relative url then is:

../Resources/Grammars/yesno.grxml

We update the VXMLForm.GrammarFileReference field then to this value.

#### MenuId

Set Menu Id to 0.

# Appendix A

## Site Configuration Options

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration Item | Value Range | Default / Example | Description |
| test\_parameter | 0 (off)  1 (on) | 1 | For now, simple read and set. Log key name and value at application exit. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |