Module Verison: 0.0.1

Last Update: 2016-08-23

NAME

ScarfXmlReader - A Python module for parsing SCARF XML files.

SYNOPSIS

```
from ScarfXmlReader import ScarfXmlReader

input = "/path/to/file"
reader = ScarfXmlReader(input)

reader.SetEncoding('UTF-8')
reader.SetInitialCallback(initialFunction)
reader.SetBugCallback(bugFunction)
reader.SetMetricCallback(metricFunction)
reader.SetBugSummaryCallback(bugSummaryFunction)
reader.SetMetricSummaryCallback(metricSummaryFunction)
reader.SetFinalCallback(finalFunction)
reader.SetCallbackData(data)

reader.parse()
```

DESCRIPTION

This module provides the ability to convert SWAMP Common Assessment Results Format (SCARF) files into Python data structures. It is dependent on xml.etree.ElementTree for parsing of the XML document.

The parser provides data to the user through the use of user provided callbacks. A callback will be called once the parser is finished parsing a section of the document. These sections are the beginning AnalyzerReport tag, an entire BugInstance or Metric, a complete BugSummary or MetricSummary, and the end AnalyzerReport tag.

All Callbacks except the FinalCallback receive as parameters a reference to a hash containing information on their section of parsed data and if set, the data passed to SetCallbackData. Upon reaching the end of an Analyzer Report the FinalCallback will be called with the error value returned from the previous call or undef if there was no error, and if set, the data passed to SetCallbackData.

While the parser does do minor checks to ensure the input file is a SCARF file, if an invalid SCARF file is passed in, the behavior is undefined. Additional validation routines can be found in the ScarfXmlWriter module. === METHODS ==== new(file) This is a class method used to instantiate the parser. FILE can be an open handle, a string containing the filename of a file, or a reference to a

string containing SCARF data.

SetEncoding(encoding)

This sets the encoding name. Current options for built in encodings are UTF-8, ISO-8859-1, UTF-16, US-ASCII. Other valid encodings can be found at https://www.iana.org/assignments/character-sets/character-sets.xhtml .

GetEncoding()

Accesses the current value stored as the encoding name.

Parse()

This method initiates the parsing of the XML. If parsing fails an exception is thrown with an error message detailing where in the file it failed. The return value of parse will be the return value of the FinalCallback if it is defined. Otherwise the return value will be the same as the last callback executed, or undef if there are no failures.

SetInitialCallback(initialFunction)

Sets the InitialCallback to be called after each start AnalyzerReport start tag is parsed.

SetBugCallback(bugFunction)

Sets the BugCallback to be called after each full BugInstance is parsed.

SetMetricCallback(metricFunction)

Sets the BugCallback to be called after each full BugInstance is parsed.

SetMetricSummaryCallback(metricSummaryFunction)

Sets the MetricSummaryCallback to be called after all metric summaries are parsed.

SetBugSummaryCallback(bugSummaryFunction)

Sets the BugSummaryCallback to be called after all metric summaries are parsed.

SetFinalCallback(finalFunction)

Sets the FinalCallback to be called after each AnalyzerReport end tag is parsed.

SetCallbackData(callbackData)

Sets the data to be used as an additional parameter to callbacks. CALLBACKDATA can be any valid Python scalar value, with the most useful being a reference to a hash. This data is passed to all callbacks and can be used to avoid global variables.

GetInitialCallback()

Access current value set to InitialCallback.

GetBugCallback()

Access current value set to BugCallback.

GetMetricCallback()

Access current value set to MetricCallback.

GetMetricSummaryCallback()

Access current value set to MetricSummaryCallback

GetBugSummaryCallback()

Access current value set to BugSummaryCallback.

GetFinalCallback()

Access current value set to FinalCallback.

GetCallbackData()

Access current value of CallbackData.

CALLBACKS()

The main purpose of this module is to interpret the events generated from XML::Parser and assemble them into a usable Python data structures. When parsing, the module will call the pre-defined callbacks upon completion of parsing an object of their respective type. If defined, all callbacks will receive the data contained in the optional key "CallbackData" as a parameter. For details on the structure of each individual Python data structure see below.

InitialCallback(initialData[, callbackData])

This is called just after the opening AnalyzerReport tag is parsed. Any defined return value will terminate parsing and skip to FinalCallback.

MetricCallback(metricData[, callbackData])

This is called every time a single Metric completes parsing. Any defined return value will terminate parsing and skip to FinalCallback.

BugCallback(metricData[, callbackData])

This is called every time a single BugInstance completes parsing. Any defined return value will terminate parsing and skip to FinalCallback.

BugSummaryCallback(bugSummaryData[, callbackData])

This is called after all BugSummaries have been parsed. Any defined return value will terminate parsing and skip to FinalCallback.

MetricSummaryCallback(metricSummaryData[, callbackData])

This is called once all MetricSummaries have been parsed. Any defined return value will terminate parsing and skip to FinalCallback.

FinalCallback(returnValue[, callbackData])

This is called after reaching an AnalayzerReport end tag. If one of the above callbacks terminates parsing with a defined return value, RETURNVALUE will equal that value, otherwise RETURNVALUE will be undef.

DATA STRUCTURES

The following are the data structures used in the callbacks listed above. If a keys value is not defined in the SCARF file, then the corresponding key will not exist in the data structures.

initialData

InitialData contains information regarding the tool used to test the package. All fields in this structure are required elements in the AnalyzerReport start tag therefore they should always be present.

```
{
    assess_fw              => ASSESSMENT_FRAMEWORK,
    assess_fw_version      => ASSESSMENT_FRAMEWORK_VERSION,
    assessment_start_ts    => ASSESSMENT_START_TIMESTAMP_SINCE_JAN_1_1970,
    build_fw               => BUILD_FRAMEWORK,
    build_fw_version       => BUILD_FRAMEWORK_VERSION,
    build_root_dir         => PACKAGE_DIRECTORY,
    package_name           => PACKAGE_NAME,
    package_root_dir       => DIRECTORY_CONTAINING_PACKAGE,
    package_version        => PACKAGE_VERSION,
    parser_fw              => PARSER_FRAMEWORK,
    parser_fw_version      => PARSER_FRAMEWORK_VERSION,
    platform_name          => PLATFORM_NAME_AND_VERSION,
    tool_name              => TOOLNAMEVALUE,
    tool_verison           => TOOLVERSIONVALUE,
    uuid                   => UUIDVALUE
}
```

bugData

BugData contains information on one BugInstance from the SCARF file. All items listed as required should always be present in the data structure. Items listed as optional will only be present if they exist in the SCARF file.

```
{
    BugId => BUGIDVALUE,                            # REQUIRED
    BugGroup => GROUPVALUE,
    BugCode => CODEVALUE,
    BugMessage => BUGMESSAGEVALUE,                  # REQUIRED
    BugRank => BUGRANKVALUE,
    BugSeverity => SEVERITYVALUE,
    ResolutionSuggestion => RESOLUTIONSUGGESTIONVALUE,
    AssessmentReportFile => ASSESSREPORTVALUE,    # REQUIRED
    BuildId => BUILDIDVALUE,                        # REQUIRED
    InstanceLocation => {
        Xpath => XPATHVALUE,
        LineNum => {
            Start = STARTVALUE,                   # REQUIRED
            End = ENDVALUE                          # REQUIRED
        }
    },
    CweIds => [
        CWEIDVALUE, CWEIDVALUE, CWEIDVALUE
        ],
    ClassName => CLASSVALUE,
    Methods => [
        {
            MethodId => METHODIDVALUE,              # REQUIRED
            name => METHODNAMEVALUE,                # REQUIRED
            primary => PRIMARYVALUE                 # REQUIRED
        },
        {
            MethodId => METHODIDVALUE,
            name => METHODNAMEVALUE,
            primary => PRIMARYVALUE
        }
    ],
    BugLocations => [                               # REQUIRED
        {
            LocationId => LOCIDVALUE,             # REQUIRED
            SourceFile => SOURCEVALUE,            # REQUIRED
            StartLine => STARTLINEVALUE,
            EndLine => ENDLINEVALUE,
            StartColumn => STARTCOLVALUE,
            EndColumn => ENDCOLVALUE,
            primary => PRIMARYVALUE,
            Explanation => EXPLANVALUE
        }
    ],
}
```

metricData

MetricData contains information on one Metric from the SCARF file. All items listed as required should always be present in the data structure. Items listed as optional will only be present if they exist in the SCARF file.

```
{
    Value => VALUE,              # REQUIRED
    Type => TYPEVALUE,           # REQUIRED
    Method => METHODVALUE,
    Class => CLASSVALUE,
    SourceFile => SOURCEVALUE,   # REQUIRED
    MetricId => METRICIDVALUE     # REQUIRED
}
```

bugSummaryData

BugSummaryData contains information on all of the BugSummaries listed in the SCARF file. All elements in this data structure are required therefore all tags will always be present. If a bug was missing either a BugGroup or BugCode, the bug is categorized as undefined for that grouping key.

```
{
BugGroup => {
    BugCode => {
        bytes => BYTESVALUE,
        count => COUNTVALUE
        }
    BugCode => {
        bytes => BYTESVALUE,
        count => COUNTVALUE
        }
    }
BugGroup => {
    BugCode => {
        bytes => BYTESVALUE,
        count => COUNTVALUE
        }
    }
}
```

metricSummaryData

MetricSummaryData contains information on all of the MetricSummaries listed in the SCARF file. All elements in this data structure are required therefore should always be present in the data structure. The only exceptions to this is if the Type of the metric is "language" or if a value of a metric in the Type was not a number, in which case only the Type and Count will be present in the summary.

```
{
MetricSummaries => [{
    Type => TYPEVALUE,
    Count => COUNTVALUE,
    Sum => SUMVALUE,
    SumOfSquares => SUMOFSQVALUE,
    Maximum => MAXVALUE,
    Minimum => MINIMUMVALUE,
    Average => AVERAGEVALUE,
    StandardDeviation => STDDEVIATIONVALUE
    },
    {
    Type => TYPEVALUE,
    Count => COUNTVALUE,
    Sum => SUMVALUE,
    SumOfSquares => SUMOFSQVALUE,
    Maximum => MAXVALUE,
    Minimum => MINIMUMVALUE,
    Average => AVERAGEVALUE,
    StandardDeviation => STDDEVIATIONVALUE
    }
}
```

Author

Brandon Klein