Module Version: 0.0.1

Last Updated: 2016-12-16

NAME

ScarfXmlWriter - A Python module for writing SCARF XML files.

SYNOPSIS

```
from ScarfXmlWriter import ScarfXmlWriter

output_file = "/path/to/file"

writer = ScarfXmlWriter(output_file, 1, 1)
writer.setPretty(0)
writer.setErrorLevel(0)

errors = []
errors.append(writer.CheckStart(INITIALDETAILS))
errors.append(writer.CheckBug(BUG))
errors.append(writer.CheckMetric(METRIC))

writer.AddStartTag(INITIALDETAILS)
writer.AddBugInstance(BUG)
writer.AddMetric(METRIC)
writer.AddSummary()
writer.AddEndTag()
writer.Close()
```

DESCRIPTION

This module provides the ability to convert Python data structures into SWAMP Common Assessment Results Format (SCARF) XML formatted documents. It is dependant on the lxml library for writing.

The writer’s output is determined by the user through method calls that provide the data to be written. This data is to be structured as a hash containing all fields for a specific section. For details on how the data structures are formatted see below.

The user has the ability to set the error level to 0 (none), 1 (warnings), or 2 (exit). Both error levels 1 and 2 will print all error messages found during writing including misformatted elements, required elements not found, wrong value types, and ordering violations. Error level 2 will also exit the program as soon as an error is found to ensure validity of results.

METHODS

ScarfXmlWriter(outputFile, errorLevel, prettyEnable)

This is a class method used to instantiate the writer. outputFile is either an object that has a write() method or, a string file path. ErrorLevel is 0, 1, or 2 and sets the error level of the writer (default 2). PrettyEnable will enable or disable pretty printing of output with a true or false value respectively (default true). The default encoding is UTF-8.

SetEncoding(encoding)

Sets the encoding to the given type. Currently this can only be UTF-8 or ASCII.

GetEncoding()

Get the current encoding type.

GetHandle()

Returns the set handle.

GetErrorLevel()

Returns the current error level of the writer.

SetErrorLevel(errorLevel)

Allows changing the error level of the writer to 0 (none), 1 (print warnings), or 2 (print warnings and throw exception). If errorLevel does not equal one of these values does nothing.

GetPretty()

Returns the current value set to pretty print.

SetPretty(prettyEnable)

If prettyEnable is a true value then enable pretty printing, else disable pretty printing.

CheckStart(initialData)

Checks INITIALDATA for any errors. Returns an array of all found errors or an empty array if none are found. For details on valid data structures see below. For details on valid data structures see below.

CheckBug(bugData)

Checks BUGDATA for any errors. Returns an array of all found errors or an empty array if none are found. For details on valid data structures see below. For details on valid data structures see below.

CheckMetric(metricData)

Checks METRICDATA for any errors. Returns an array of all found errors or an empty array if none are found. For details on valid data structures see below. For details on valid data structures see below.

AddStartTag(initialData)

Writes a start tag to the file based on INITIALDATA. For details on valid data structures see below. Must be called exactly once before other 'Add' methods below.

AddBugInstance(bugData)

Writes a bug to the file based on BUGDATA. For details on valid data structures see below. May be called 0 or more times and interleaved with 'AddMetric' calls. Not allowed after 'AddSummary' and 'AddEndTag'.

AddMetric(metricData)

Writes a metric to the file based on METRICDATA. For details on valid data structures see below. May be called 0 or more times and interleaved with 'AddBug' calls. Not allowed after 'AddSummary' and 'AddEndTag'.

AddSummary()

Writes a summary to the file based on all bugs and metrics already written with this writer. May be called at most 1 time before 'AddEndTag'.

AddEndTag()

Writes an end tag to the file. Must be called exactly once after which no other 'Add' methods may be called.

DATA STRUCTURES

The following are the data structures used in the methods listed above. Undefined keys will not be written.

initialData

InitialData contains information regarding the tool used to test the package. All fields in this structure are required elements therefore must be included in the data structure.

```
{
    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. Other items listed are not required, but can be included and written to SCARF.

```
{
    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 => [
        {
            name => METHODNAMEVALUE,               # REQUIRED
            primary => PRIMARYVALUE                # REQUIRED
        },
        {
            name => METHODNAMEVALUE,
            primary => PRIMARYVALUE
        }
    ],
    BugLocations => [                              # REQUIRED
        {
            SourceFile => SOURCEVALUE,             # REQUIRED
            StartLine => STARTLINEVALUE,
            EndLine => ENDLINEVALUE,
            StartColumn => STARTCOLVALUE,
            EndColumn => ENDCOLVALUE,
            primary => PRIMARYVALUE,               # REQUIRED
            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. Other items listed are not required, but can be written to SCARF.

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

AUTHOR

Brandon Klein