# stix-validator Documentation

***Release 2.3.0***

**The MITRE Corporation**

June 26, 2015

Contents

**Version**: 2.3.0

The Structued Threat Information eXpression (STIX) and Cyber Observable eXpression (CybOX) are collaborative, community-driven efforts to define and develop standardized languages to represent structured cyber threat information and are currently implemented in XML Schema.

The **stix-validator** library and scripts helps validate STIX XML content using XML Schema, STIX Suggested Practices, and STIX Profile validation. The **stix-validator** library also provides CybOX XML validation capabilities!

For more information about STIX, please visit the STIX homepage and STIX Documentation website. For more information about CybOX, please visit the CybOX homepage and the CybOX Documentation website.

CHAPTER 1

# Contents

**Version**: 2.3.0

## 1.1 Installation

The installation of **stix-validator** can be accomplished through a few different work flows.

### 1.1.1 Recommended Installation

Use PyPI and pip:

```
$ pip install stix-validator [--upgrade]
```

You might also want to consider using a virtualenv. Please refer to the pip installation instructions for details regarding the installation of pip.

### 1.1.2 Dependencies

The **stix-validator** package relies on some non-standard Python libraries for the processing of XML content. Revisions of stix-validator may depend on particular versions of dependencies to function correctly. These versions are detailed within the distutils `setup.py` installation script.

The following libraries are required to use stix-validator:

- lxml - A Pythonic binding for the C libraries **libxml2** and **libxslt**.
- xlrd - A library for parsing Microsoft Excel documents
- ordereddict - A drop-in replacement for `collections.OrderedDict`, which is not available in Python 2.6.

Each of these can be installed with `pip` or by manually downloading packages from PyPI. On Windows, you will probably have the most luck using pre-compiled binaries for `lxml`. On Ubuntu (12.04 or 14.04), you should make sure the following packages are installed before attempting to compile `lxml` from source:

- libxml2-dev
- libxslt1-dev
- zlib1g-dev

**Warning:** Users have encountered errors with versions of `libxml2` (a dependency of `lxml`) prior to version 2.9.1. The default version of `libxml2` provided on Ubuntu 12.04 is currently 2.7.8. Users are encouraged to upgrade `libxml2` manually if they have any issues. Ubuntu 14.04 provides `libxml2` version 2.9.1.

### 1.1.3 Manual Installation

If you are unable to use pip, you can also install python-stix with setuptools. If you don't already have setuptools installed, please install it before continuing.

1. Download and install the *dependencies* above. Although setuptools will generally install dependencies automatically, installing the dependencies manually beforehand helps distinguish errors in dependency installation from errors in stix installation. Make sure you check to ensure the versions you install are compatible with the version of stix you plan to install.
2. Download the desired version of stix from PyPI or the GitHub releases page. The steps below assume you are using the 2.3.0 release.
3. Extract the downloaded file. This will leave you with a directory named stix-validator-2.3.0.

```
$ tar -zxf stix-validator-2.3.0.tar.gz
$ ls
stix-validator-2.3.0 stix-validator-2.3.0.tar.gz
```

OR

```
$ unzip stix-validator-2.3.0.zip
$ ls
stix-validator-2.3.0 stix-validator-2.3.0.zip
```

4. Run the installation script.

```
$ cd stix-validator-2.3.0
$ python setup.py install
```

5. Test the installation.

```
$ python
Python 2.7.8 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type ``help'', ``copyright'', ``credits'' or ``license'' for more information.
>>> import sdv
>>> print sdv.__version__
2.3.0
```

If you don't see an `ImportError`, the installation was successful.

### 1.1.4 Further Information

If you're new to installing Python packages, you can learn more at the Python Packaging User Guide, specifically the Installing Python Packages section.

**Version**: 2.3.0

## 1.2 Getting Started

This page gives an introduction to **stix-validator** scripts. Please note that this page is being actively worked on and feedback is welcome! If you have a suggestion or something doesn't look right, let us know: (stix@mitre.org).

Note that the GitHub repository is named `stix-validator`, but once installed, the library is imported using the `import sdv` statement.

### 1.2.1 Installation

To install **stix-validator** just run `pip install stix-validator`. If you have any issues, please refer to the instructions found on the Installation page.

### 1.2.2 Scripts

The **stix-validator** library comes with two scripts capable of performing the validation of STIX and CybOX documents: `stix_validator.py` and `cybox_validator.py`. These scripts can be found on your `PATH` after installing the **stix-validator**.

These instructions tell you how to validate STIX and CybOX content using the scripts bundled with **stix-validator**.

#### STIX Document Validator

The `stix_validator.py` script can be used to validate STIX content in a number of ways. The following sections describe the validation options and expected behavior of the `stix_validator.py` script.

##### Options

Running `stix_validator.py -h` displays the following:

```
$ stix_validator.py -h
usage: stix_validator.py [-h] [--stix-version STIX_VERSION]
                         [--schema-dir SCHEMA_DIR] [--use-schemaloc]
                         [--best-practices] [--profile PROFILE]
                         [--schematron-out SCHEMATRON] [--xslt-out XSLT]
                         [--quiet] [--json-results]
                         [FILES [FILES ...]]

STIX Document Validator v2.1

positional arguments:
  FILES                 A whitespace separated list of STIX files or
                        directories of STIX files to validate.

optional arguments:
  -h, --help            show this help message and exit
  --stix-version STIX_VERSION
                        The version of STIX to validate against
  --schema-dir SCHEMA_DIR
                        Schema directory. If not provided, the STIX schemas
                        bundled with the stix-validator library will be used.
  --use-schemaloc       Use schemaLocation attribute to determine schema
                        locations.
```

```
  --best-practices      Check that the document follows authoring best
                        practices
  --profile PROFILE     Path to STIX profile in excel
  --schematron-out SCHEMATRON
                        Path to converted STIX profile schematron file output.
  --xslt-out XSLT       Path to converted STIX profile schematron xslt output.
  --quiet               Only print results and errors if they occur.
  --json-results        Print results as raw JSON. This also sets --quiet.
```

#### Example STIX Schema Validation

To perform xml schema validation, just pass in a path to the STIX filename, filenames, and/or directories containing STIX content.

```
$ stix_validator.py stix-content.xml another-stix-doc.xml
```

If these documents were valid, the `stix_validator.py` script would print something like the following:

```
[-] Performing xml schema validation on stix-content.xml
[-] Performing xml schema validation on another-stix-doc.xml
============================================================
[-] Results: stix-content.xml
[+] XML Schema: True
============================================================
[-] Results: another-stix-doc.xml
[+] XML Schema: True
```

### CybOX Document Validator

The `cybox_validator.py` script can be used to perform XML Schema validation on one or more input CybOX documents. The following sections describe the validation options and expected behavior of the `cybox_validator.py` script.

#### Options

The `cybox_validator.py` script provides CybOX XML Schema validation capabilities to your command line.

```
$ cybox_validator.py -h
usage: cybox_validator.py [-h] [--cybox-version LANG_VERSION]
                          [--schema-dir SCHEMA_DIR] [--use-schemaloc]
                          [--quiet] [--json-results] [--recursive]
                          [FILES [FILES ...]]

CybOX Document Validator v2.1

positional arguments:
  FILES                 A whitespace separated list of CybOX files or
                        directories of CybOX files to validate.

optional arguments:
  -h, --help            show this help message and exit
  --cybox-version LANG_VERSION
                        The version of CybOX to validate against
  --schema-dir SCHEMA_DIR
```

```
                        Schema directory. If not provided, the CybOX schemas
                        bundled with the stix-validator library will be used.
  --use-schemaloc       Use schemaLocation attribute to determine schema
                        locations.
  --quiet               Only print results and errors if they occur.
  --json-results        Print results as raw JSON. This also sets --quiet.
  --recursive           Recursively descend into input directories.
```

#### Example CybOX Schema Validation

To perform xml schema validation, just pass in a path to the CybOX filename, filenames, and/or directories containing CybOX content.

```
$ cybox_validator.py cybox-content.xml another-cybox-doc.xml
```

If these documents were valid, the `cybox_validator.py` script would print something like the following:

```
[-] Performing xml schema validation on cybox-content.xml
[-] Performing xml schema validation on another-cybox-doc.xml
============================================================
[-] Results: cybox-content.xml
[+] XML Schema: True
============================================================
[-] Results: another-cybox-doc.xml
[+] XML Schema: True
```

### Exit Codes

Exit status codes for the **stix-validator** bundled scripts are defined within *sdv.codes* module.

When invoking the `stix_validator.py` or `cybox_validator.py` scripts from another process, developers can inspect the exit code after execution to determine the results of the validation attempt. Exit status codes can be combined via bitmasks to convey multiple results (multiple files validated and/or multiple validation methods selected).

The following script demonstrates an example of invoking `stix-validator.py` from another Python script.

```
#!/usr/bin/env python

import subprocess
import sdv.codes as codes # STIX Document Validator exit codes

ARGS = [
    'stix_validator.py',
    '--best-practices',
    '--profile',
    'stix-profile.xlsx',
    'stix-document.xml'
]

# Run the stix_validator.py script as a subprocess. Redirect stdout.
results = subprocess.call(ARGS, stdout=subprocess.PIPE)

# Check exit status code(s)

if codes.EXIT_SUCCESS & results:
    print "Input document(s) were valid."
```

```
if codes.EXIT_SCHEMA_INVALID & results:
    print "One or more input files were schema-invalid."

if codes.EXIT_BEST_PRACTICE_INVALID & results:
    print "One or more input files were STIX Best Practices invalid."

if codes.EXIT_PROFILE_INVALID & results:
    print "One or more input files were STIX Profile invalid."

if codes.EXIT_VALIDATION_ERROR & results:
    print "A validation error occurred."

if codes.EXIT_FAILURE & results:
    print "An unknown, fatal error occurred."
```

**Note:** Invoking `stix_validator.py` or `cybox_validator.py` as a subprocess may not always be the best method for validating STIX documents from a Python script. The *sdv* module contains methods for performing STIX and CybOX validation!

CHAPTER 2

# API Documentation

**Version**: 2.3.0

## 2.1 API Reference

The **stix-validator** APIs provide methods for validating STIX and CybOX content. Listed below are the modules and packages provided by the **stix-validator** library.

For examples of how make use of all of this, check out the Code Examples page.

**Note:** The **stix-validator** APIs are currently under heavy development. Feel free to check out our issue tracker to see what we're working on!

**Version**: 2.3.0

### 2.1.1 `sdv` Module

`sdv.`**`validate_xml`**(*doc*, *version=None*, *schemas=None*, *schemaloc=False*, *klass=None*)

Performs XML Schema validation against a STIX or CybOX document.

**Parameters**

- **`doc`** – A STIX/CybOX document to validate. This can be a filename, file-like object, `etree._Element` or `etree._ElementTree` object.
- **`version`** – The version of the STIX/CybOX document being validated. If `None` an attempt will be made to extract the version from *doc*.
- **`schemas`** – A string path to a directory of STIX/CybOX schemas. If `None`, the validation code will leverage its bundled STIX/CybOX schemas.
- **`schemaloc`** – Use `xsi:schemaLocation` attribute on *doc* to perform validation.
- **`klass`** – **Internal use only**. The validator klass to use for validating *doc*.

**Note:** The first time running this for a given *schemas* (or no *schemas*) will take longer than following validation runs due to schema compilation time.

**Returns** An instance of *`XmlValidationResults`*.

**Raises**

- `IOError` – If *doc* is not a valid XML document or there is an issue processing *schemas*.
- *`UnknownSTIXVersionError`* – If *version* is `None` and *doc* does not contain a `@version` attribute value.
- *`UnknownCyboxVersionError`* – If *version* is `None` and *doc* does not contain CybOX version information.
- *`InvalidSTIXVersionError`* – If *version* or the `version` attribute in *doc* contains an invalid STIX version number.
- *`InvalidCyboxVersionError`* – if *version* or the version information on *doc* contain an invalid CybOX version number.
- *`ValidationError`* – If the class was not initialized with a schema directory and *schemaloc* is `False`.
- *`XMLSchemaImportError`* – If an error occurs while processing the schemas required for validation.
- *`XMLSchemaIncludeError`* – If an error occurs while processing `xs:include` directives.

`sdv.`**`validate_best_practices`**(*doc*, *version=None*)

Performs Best Practices validation against a STIX document.

---

**Note:** This should be used together with *`validate_xml()`* since this only checks best practices and not schema-conformance.

---

**Parameters**

- **`doc`** – A STIX document to validate. This can be a filename, file-like object, `etree._Element` or `etree._ElementTree` object.
- **`version`** – The version of the STIX document being validated. If `None` an attempt will be made to extract the version from *doc*.

**Returns** An instance of *`BestPracticeValidationResults`*.

**Raises**

- `IOError` – If *doc* is not a valid XML document.
- *`ValidationError`* – If *doc* is not a well-formed STIX document.
- *`UnknownSTIXVersionError`* – If *version* is `None` and *doc* does not contain version information.
- *`InvalidSTIXVersionError`* – If *version* or the `@version` attribute in *doc* contains an invalid STIX version number.

`sdv.`**`validate_profile`**(*doc*, *profile*)

Performs STIX Profile validation against a STIX document.

---

**Note:** This should be used together with *`validate_xml()`* since this only checks profile-conformance and not schema-conformance.

---

**Parameters**

- **`doc`** – A STIX document to validate. This can be a filename, file-like object, `etree._Element` or `etree._ElementTree` object.

- **profile** – A filename to a STIX Profile document.

**Returns** An instance of *ProfileValidationResults*.

**Raises**

- IOError – If *doc* is not a valid XML document.
- *ValidationError* – If the input document is not a well-formed STIX document.
- *ProfileParseError* – If an error occurred while attempting to parse the *profile*.

sdv.**profile_to_schematron**(*profile*)
Converts the STIX Profile *profile* into a Schematron representation.

**Parameters** **profile** – A filename to a STIX Profile document.

**Returns** An etree._ElementTree Schematron representation of *profile*.

**Raises** *ProfileParseError* – If an error occurred while attempting to parse the *profile*.

sdv.**profile_to_xslt**(*profile*)
Converts the STIX Profile *profile* into an XSLT representation.

**Parameters** **profile** – A filename to a STIX Profile document.

**Returns** An etree._ElementTree XSLT representation of *profile*.

**Raises** *ProfileParseError* – If an error occurred while attempting to parse the *profile*.

**Version**: 2.3.0

### 2.1.2 sdv.codes Module

This module defines exit status codes used by bundled scripts.

sdv.codes.**EXIT_SUCCESS = 0**
Execution finished successfully. All STIX documents were valid for all user- specified validation scenarios.

sdv.codes.**EXIT_FAILURE = 1**
Execution finished with fatal system error. Some unhandled system exception was raised during execution.

sdv.codes.**EXIT_SCHEMA_INVALID = 2**
Execution finished with at least one input document found to be schema- invalid.

sdv.codes.**EXIT_PROFILE_INVALID = 4**
Execution finished with at least one input document found to be profile invalid.

sdv.codes.**EXIT_BEST_PRACTICE_INVALID = 8**
Execution finished with at least one input document found to be best practice invalid.

sdv.codes.**EXIT_VALIDATION_ERROR = 16**
An error occurred while validating an instance document. This can be caused by malformed input documents or file names that do not resolve to actual files.

**Version**: 2.3.0

### 2.1.3 sdv.errors Module

**exception** sdv.errors.**IdrefLookupError**(*idref*, *message=None*)
Bases: *sdv.errors.ValidationError*

Raised when an attempt to resolve an ID reference fails. This can occur when the full STIX component definition resides outside of the input document.

**exception** sdv.errors.**InvalidCyboxVersionError**(*message*, *expected=None*, *found=None*)
Bases: *sdv.errors.InvalidVersionError*

Raised when an invalid version of CybOX is discovered within an instance document or is passed into a method which depends on CybOX version information.

**Parameters**

- **message** – The error message.
- **expected** – A version or list of expected versions.
- **found** – The CybOX version that was declared for an instance document or found within an instance document.

**exception** sdv.errors.**InvalidSTIXVersionError**(*message*, *expected=None*, *found=None*)
Bases: *sdv.errors.InvalidVersionError*

Raised when an invalid version of STIX is discovered within an instance document or is passed into a method which depends on STIX version information.

**Parameters**

- **message** – The error message.
- **expected** – A version or list of expected versions.
- **found** – The STIX version that was declared for an instance document or found within an instance document.

**exception** sdv.errors.**InvalidVersionError**(*message*, *expected=None*, *found=None*)
Bases: *sdv.errors.ValidationError*

Base Exception for errors raised as a result of invalid version information being declared for a document, or found within a document.

**exception** sdv.errors.**ProfileParseError**
Bases: *sdv.errors.ValidationError*

Raised when an error occurs during the parse or initialization of a STIX profile document.

**exception** sdv.errors.**UnknownCyboxVersionError**
Bases: *sdv.errors.UnknownVersionError*

Raised when no CybOX version information can be found in an instance document and no version information was provided to a method which requires version information.

**exception** sdv.errors.**UnknownNamespaceError**
Bases: *sdv.errors.ValidationError*

Raised when an unknown namespace is encountered in a function.

**exception** sdv.errors.**UnknownSTIXVersionError**
Bases: *sdv.errors.UnknownVersionError*

Raised when no STIX version information can be found in an instance document and no version information was provided to a method which requires version information.

**exception** sdv.errors.**UnknownVersionError**
Bases: *sdv.errors.ValidationError*

Base Exception for errors raised as a result of not being able to determine the version of an input document.

**exception** sdv.errors.**UnknownVocabularyError**
Bases: *sdv.errors.ValidationError*

Raised when an unknown controlled vocabulary name is discovered during best practice validation.

**exception** sdv.errors.**ValidationError**
Bases: exceptions.Exception

Base Exception for all validator-specific exceptions. This is used directly by some modules as a generic Exception.

**exception** sdv.errors.**XMLSchemaImportError**
Bases: *sdv.errors.ValidationError*

Raised when errors occur when generating `xs:import` directives for the "uber" schema, used to validate XML instance documents.

**exception** sdv.errors.**XMLSchemaIncludeError**
Bases: *sdv.errors.ValidationError*

Raised when errors occur during the processing of `xs:include` directives found within schema documents.

**Version**: 2.3.0

### 2.1.4 sdv.validators.cybox.schema Module

**class** sdv.validators.cybox.schema.**CyboxSchemaValidator**(*schema_dir=None*)
Bases: sdv.validators.base.BaseSchemaValidator

**validate**(*doc*, *version=None*, *schemaloc=False*)
Performs XML Schema validation against a CybOX document.

When validating against the set of bundled schemas, a CybOX version number must be declared for the input *doc*. If a user does not pass in a *version* parameter, an attempt will be made to collect the version from the input *doc*.

---

**Note:** If *schemaloc* is `True` or this class was initialized with a `schema_dir`, no version checking or verification will occur.

---

**Parameters**

- **doc** – The CybOX document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree` instance.
- **version** – The version of the CybOX document. If `None` an attempt will be made to extract the version from *doc*.
- **schemaloc** – If `True`, the `xsi:schemaLocation` attribute on *doc* will be used to drive the validation.

**Returns** An instance of *XmlValidationResults*.

**Raises**

- *UnknownCyboxVersionError* – If *version* is `None` and *doc* does not contain CybOX version information.
- *InvalidCyboxVersionError* – If *version* is an invalid CybOX version or *doc* contains an invalid CybOX version number.

- *ValidationError* – If the class was not initialized with a schema directory and *schemaloc* is `False`.
- *XMLSchemaImportError* – If an error occurs while processing the schemas required for validation.
- *XMLSchemaIncludeError* – If an error occurs while processing `xs:include` directives.
- *ValidationError* – If there are any issues parsing *doc*.

**Version**: 2.3.0

### 2.1.5 `sdv.validators.schematron` Module

**class** `sdv.validators.schematron.`**`SchematronValidator`**(*schematron*)

Performs schematron validation against an XML instance document.

**Parameters** **`schematron`** – A Schematron document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree` instance.

**`validate`**(*doc*)

Validates an XML instance document *doc* using Schematron rules.

**Parameters** **`doc`** – An XML instance document. This can be a filename, file-like object, `etree._Element` or `etree._ElementTree` instance.

**Returns** An instance of *SchematronValidationResults*.

**Raises** *ValidationError* – If there are any issues parsing *doc*.

**class** `sdv.validators.schematron.`**`SchematronValidationResults`**(*is_valid*, *doc=None*, *svrl_report=None*)

Bases: `sdv.validators.base.ValidationResults`

Used to hold results of a Schematron validation process.

**Parameters**

- **`is_valid`** – The boolean validation result.
- **`doc`** – The document which produced these validation results.
- **`svrl_report`** – The etree._ElementTree SVRL report produced during the validation run.

**`errors`**

A list of *SchematronError* instances representing errors found in the *svrl_report*.

**`is_valid`**

Returns `True` if the validation was successful and `False` otherwise.

**`as_dict`**()

A dictionary representation of the *SchematronValidationResults* instance.

**Keys:**

- `'result'`: The validation results. Values can be `True` or `False`.
- `'errors'`: A list of validation error dictionaries. The keys are `'message'` and `'line'`.

**Returns** A dictionary representation of an instance of this class.

**as_json**()
Returns a JSON representation of this class instance.

**is_valid**
Returns `True` if the validation attempt was successful and `False` otherwise.

**class** sdv.validators.schematron.**SchematronError**(*doc*, *error*)
Bases: sdv.validators.base.ValidationError

Represents an error found in a SVRL report.

**Parameters**

- **doc** – The instance document which was validated and produced this error.
- **error** – The `svrl:failed-assert` or `svrl:successful-report` `etree._Element` instance.

**message**
The validation error message.

**as_dict**()
Returns a dictionary representation.

**Keys:**

- `'message'`: The error message
- `'line'`: The line number associated with the error

**as_json**()
Returns a JSON representation of this class instance.

**line**
Returns the line number in the input document associated with this error.

This property is lazily evaluated, meaning the line number isn't known until the first time this property is accessed. Each subsequent call will return the cached line number.

**Version**: 2.3.0

### 2.1.6 sdv.validators.stix.best_practice Module

**class** sdv.validators.stix.best_practice.**STIXBestPracticeValidator**
Bases: object

Performs STIX Best Practice validation.

**validate**(*doc*, *version=None*)
Checks that a STIX document aligns with suggested authoring practices.

**Parameters**

- **doc** – The STIX document. Can be a filename, file-like object, lxml._Element, or lxml._ElementTree instance.
- **version** – The version of the STIX document. This will determine the set of best practice rules to check. If `None` an attempt will be made to extract the version from *doc*.

**Returns** An instance of *BestPracticeValidationResults*.

**Raises**

- *UnknownSTIXVersionError* – If *version* was `None` and *doc* did not contain any version information.

- *InvalidSTIXVersionError* – If discovered version or *version* argument contains an invalid STIX version number.
- *ValidationError* – If there are any issues parsing *doc*.

**class** sdv.validators.stix.best_practice.**BestPracticeWarning**(*node*, *message=None*)

Represents a best practice warning. These are built within best practice rule checking methods and attached to *BestPracticeWarningCollection* instances.

---

**Note:** This class acts like a dictionary and contains the following keys at a minimum:

- `'id'`: The id of a node associated with the warning.
- `'idref'`: The idref of a node associated with the warning.
- `'line'`: The line number of the offending node.
- `'message'`: A message associated with the warning.
- `'tag'`: The lxml tag for the offending node.

These keys can be retrieved via the *core_keys* property.

Instances of this class may attach additional keys. These *other keys* can be obtained via the *other_keys* property.

---

**Parameters**

- **node** – The `lxml._Element` node associated with this warning.
- **message** – A message for this warning.

**as_dict**()

Returns a dictionary representation of this class instance. This is implemented for consistency across other validation error types.

The *BestPracticeWarning* class extends `collections.MutableMapping`, so this method isn't really necessary.

**as_json**()

Returns a JSON representation of this class instance.

**core_keys**

Returns a `tuple` of the keys that can always be found on instance of this class.

**Returns**

A tuple including the following keys.

- `'id'`: The id of the warning node. The associated value may be `None`.
- `'idref'`: The idref of the warning node. The associated value may be `None`.
- `'line'`: The line number of the warning node in the input document. The associated value may be `None`.
- `'tag'`: The `{namespace}localname` value of the warning node.
- `'message'`: An optional message that can be attached to the warning. The associated value may be `None`.

**line**

Returns the line number of the warning node in the input document.

**other_keys**
Returns a `tuple` of keys attached to instances of this class that are not found in the *core_keys*.

**class** sdv.validators.stix.best_practice.**BestPracticeWarningCollection**(*name*)
Bases: _abcoll.MutableSequence

A collection of *BestPracticeWarning* instances for a given type of STIX Best Practice.

For example, all warnings about STIX constructs missing titles would go within an instance of this class.

**Note:** This class behaves like a mutable sequence, such as a `list`.

**Parameters** **name** – The name of the STIX best practice for this collection (e.g., 'Missing Titles').

**name**
The name of the STIX best practice for this collection (e.g., 'Missing Titles').

**as_dict**()
Returns a dictionary representation.

The key of the dictionary is the `name` of this collection. The associated value is a `list` of *BestPracticeWarning* dictionaries.

**insert**(*idx*, *value*)
Inserts *value* at *idx* into this *BestPracticeWarningCollection* instance.

**Note:** Values that evaluate to `False` will not be inserted.

**class** sdv.validators.stix.best_practice.**BestPracticeValidationResults**
Bases: sdv.validators.base.ValidationResults, _abcoll.MutableSequence

Represents STIX best practice validation results. This class behaves like a `list` and accepts instances of *BestPracticeWarningCollection*.

**as_dict**()
Returns a dictionary representation.

**Keys:**

- `'result'`: The result of the validation. Values can be `True` or `False` .
- `'errors'`: A list of *BestPracticeWarningCollection* dictionaries.

**as_json**()
Returns a JSON representation of this class instance.

**errors**
Returns a `list` of *BestPracticeWarningCollection* instances.

**is_valid**
Returns `True` if an instance of this class contains no warning collections or only contains only warning collections.

**Version**: 2.3.0

### 2.1.7 sdv.validators.stix.profile Module

**class** sdv.validators.stix.profile.**STIXProfileValidator**(*profile_fn*)
Bases: *sdv.validators.schematron.SchematronValidator*

Performs STIX Profile validation.

**Parameters** **profile_fn** – The filename of a `.xlsx` STIX Profile document.

**validate**(*doc*)
Validates an XML instance document against a STIX profile.

**Parameters** **doc** – The STIX document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree` instance.

**Returns** An instance of *ProfileValidationResults*.

**Raises** *ValidationError* – If there are any issues parsing *doc*.

**class** sdv.validators.stix.profile.**ProfileValidationResults**(*is_valid*, *doc=None*, *svrl_report=None*)
Bases: *sdv.validators.schematron.SchematronValidationResults*

Represents STIX profile validation results. This is returned from the *STIXProfileValidator.validate()* method.

**Parameters**

- **is_vaild** – `True` if the document was valid and `False` otherwise.
- **doc** – The document that was validated. This is an instance of lxml._Element.
- **svrl_report** – The SVRL report. This is an instance of `lxml.isoschematron.Schematron.validation_report`

**errors**
A list of *ProfileError* instances representing errors found in the *svrl_report*.

**as_dict**()
A dictionary representation of the *SchematronValidationResults* instance.

**Keys:**

- `'result'`: The validation results. Values can be `True` or `False`.
- `'errors'`: A list of validation error dictionaries. The keys are `'message'` and `'line'`.

**Returns** A dictionary representation of an instance of this class.

**as_json**()
Returns a JSON representation of this class instance.

**is_valid**
Returns `True` if the validation attempt was successful and `False` otherwise.

**class** sdv.validators.stix.profile.**ProfileError**(*doc*, *error*)
Bases: *sdv.validators.schematron.SchematronError*

Represents STIX profile validation error.

**Parameters**

- **doc** – The instance document which was validated and produced this error.
- **error** – The `svrl:failed-assert` or `svrl:successful-report` `etree._Element` instance.

**message**
The STIX Profile validation error message.

**as_dict**()
Returns a dictionary representation.

**Keys:**

- `'message'`: The error message
- `'line'`: The line number associated with the error

**as_json**()
Returns a JSON representation of this class instance.

**line**
Returns the line number in the input document associated with this error.

This property is lazily evaluated, meaning the line number isn't known until the first time this property is accessed. Each subsequent call will return the cached line number.

**Version**: 2.3.0

### 2.1.8 sdv.validators.stix.schema Module

**class** sdv.validators.stix.schema.**STIXSchemaValidator**(*schema_dir=None*)
Bases: sdv.validators.base.BaseSchemaValidator

**validate**(*doc*, *version=None*, *schemaloc=False*)
Performs XML Schema validation against a STIX document.

When validating against the set of bundled schemas, a STIX version number must be declared for the input *doc*. If a user does not pass in a *version* parameter, an attempt will be made to collect the version from the input *doc*.

---

**Note:** If *schemaloc* is `True` or this class was initialized with a `schema_dir`, no version checking or verification will occur.

---

**Parameters**

- **doc** – The STIX document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree` instance.
- **version** – The version of the STIX document. If `None` an attempt will be made to extract the version from *doc*.
- **schemaloc** – If `True`, the `xsi:schemaLocation` attribute on *doc* will be used to drive the validation.

**Returns** An instance of *XmlValidationResults*.

**Raises**

- *UnknownSTIXVersionError* – If *version* is `None` and *doc* does not contain STIX version information.
- *InvalidSTIXVersionError* – If *version* is an invalid STIX version or *doc* contains an invalid STIX version number.
- *ValidationError* – If the class was not initialized with a schema directory and *schemaloc* is `False`.
- *XMLSchemaImportError* – If an error occurs while processing the schemas required for validation.

- *XMLSchemaIncludeError* – If an error occurs while processing `xs:include` directives.
- *ValidationError* – If there are any issues parsing *doc*.

**Version**: 2.3.0

### 2.1.9 `sdv.validators.xml_schema` Module

**class** `sdv.validators.xml_schema.`**`XmlSchemaValidator`**(*schema_dir=None*)
Validates XML instance documents.

**Note:** If validating against a single XML schema document, use `lxml.etree.XMLSchema` instead.

**Parameters** **`schema_dir`** – A directory of schema files used to validate XML instance documents.

**`OVERRIDE_SCHEMALOC`**
Overrides the schemalocation for a given namespace that may be discovered when walking *schema_dir*. This does not alter the schemalocation of namespaces declared by `xsi:schemalLocation` attributes if validating via `xsi:schemaLocation`.

**`validate`**(*doc*, *schemaloc=False*)
Validates an XML instance document.

**Parameters**

- **`doc`** – An XML instance document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree`.
- **`schemaloc`** – If `True`, the document will be validated using the `xsi:schemaLocation` attribute found on the instance document root.

**Returns** An instance of *XmlValidationResults*.

**Raises**

- *ValidationError* – If the class was not initialized with a schema directory and *schemaloc* is `False` or if there are any issues parsing *doc*.
- *XMLSchemaIncludeError* – If an error occurs while processing the schemas required for validation.
- *XMLSchemaIncludeError* – If an error occurs while processing `xs:include` directives.

**class** `sdv.validators.xml_schema.`**`XmlValidationResults`**(*is_valid*, *errors=None*)
Results of XML schema validation. Returned from *XmlSchemaValidator.validate()*.

**Parameters**

- **`is_valid`** – The validation result.
- **`errors`** – A list of strings reported from the XML validation engine.

**`is_valid`**
`True` if the validation was successful and `False` otherwise.

**`as_dict`**()
A dictionary representation of the *XmlValidationResults* instance.

**Keys:**

- `'result'`: The validation results (`True` or `False`)
- `'errors'`: A list of validation errors.

**Returns** A dictionary representation of an instance of this class.

**as_json**()
Returns a JSON representation of this class instance.

**errors**
"A list of *XmlSchemaError* validation errors.

**is_valid**
Returns `True` if the validation attempt was successful and `False` otherwise.

**class** sdv.validators.xml_schema.**XmlSchemaError**(*error*)
Represents an XML Schema validation error.

**Parameters** **error** – An error returned from `etree` XML Schema validation error log.

**message**
The XML validation error message.

**as_dict**()
Returns a dictionary representation.

**Keys:**

- `'message'`: The error message
- `'line'`: The line number associated with the error

**as_json**()
Returns a JSON representation of this class instance.

**line**
Returns the line number associated with the error.

**Version**: 2.3.0

### 2.1.10 sdv.utils Module

sdv.utils.**children**(*node*)
Returns an iterable collection of etree Element nodes that are direct children of *node*.

sdv.utils.**descendants**(*node*)
Returns a list of etree Element nodes that are descendants of *node*.

sdv.utils.**get_document_namespaces**(*doc*)
Returns namespace dictionary for all the namespaces declared in the input *doc*.

**Parameters** **doc** – A read()-able XML document or etree node.

sdv.utils.**get_etree_root**(*doc*)
Returns an instance of lxml.etree._Element for the given *doc* input.

**Parameters** **doc** – The input XML document. Can be an instance of `lxml.etree._Element`, `lxml.etree._ElementTree`, a file-like object, or a string filename.

**Returns** An `lxml.etree._Element` instance for *doc*.

**Raises** *ValidationError* – If *doc* cannot be found or is not a well-formed XML document.

sdv.utils.**get_namespace**(*node*)
Returns the namespace for which *node* falls under.

**Parameters** **node** – An etree node.

sdv.utils.**get_schemaloc_pairs**(*node*)
Parses the xsi:schemaLocation attribute on *node*.

**Returns** A list of (ns, schemaLocation) tuples for the node.

**Raises** `KeyError` – If *node* does not have an xsi:schemaLocation attribute.

sdv.utils.**get_target_ns**(*doc*)
Returns the value of the `targetNamespace` attribute found on *doc*.

**Returns** The value of the `targetNamespace` attribute found at the root of *doc*.

**Raises**

- `KeyError` – If *doc* does not contain a `targetNamespace` attribute.
- *`ValidationError`* – If *doc* cannot be found or is not a well-formed XML document.

sdv.utils.**get_type_ns**(*doc*, *typename*)
Returns the namespace associated with the `xsi:type` *typename* found in the XML document *doc*.

**Parameters**

- **doc** – An XML document. This can be a filename, file-like object, `etree._Element`, or `etree._ElementTree` instance.
- **typename** – The `xsi:type` value for a given vocabulary instance.

sdv.utils.**get_xml_files**(*files*, *recursive=False*)
Returns a list of files to validate from *files*. If a member of *files* is a directory, its children with a `.xml` extension will be added to the return value.

**Parameters**

- **files** – A list of file paths and/or directory paths.
- **recursive** – If `true`, this will descend into any subdirectories of input directories.

**Returns** A list of file paths to validate.

sdv.utils.**get_xml_parser**(*encoding=None*)
Returns the global XML parser object. If no global XML parser has been set, one will be created and then returned.

**Parameters** **encoding** – The expected encoding of input documents. By default, an attempt will be made to determine the input document encoding.

**Returns** The global XML parser object.

sdv.utils.**has_content**(*node*)
Returns `True` if the *node* has children or text nodes.

---

**Note:** This will ignore whitespace and XML comments.

---

sdv.utils.**has_tzinfo**(*timestamp*)
Returns `True` if the *timestamp* includes timezone or UTC offset information.

sdv.utils.**ignored**(**args*, ***kwds*)
Allows you to ignore exceptions cleanly using context managers. This exists in Python 3.4 as `contextlib.suppress()`.

sdv.utils.**is_cybox**(*doc*)
Attempts to determine if the input *doc* is a CybOX XML instance document. If the root-level element falls under a namespace which starts with `http://cybox.mitre.org`, this will return True.

sdv.utils.**is_element**(*node*)
Returns `True` if *node* is an etree._Element instance.

sdv.utils.**is_equal_timestamp**(*ts1*, *ts2*)
Returns `True` if the timestamps *ts1* and *ts2* are equal.

**Parameters**

- **ts1** – Timestamp string/datetime or etree Element node with 'timestamp' attribute.
- **ts2** – Timestamp string/datetime or etree Element node with 'timestamp' attribute.

sdv.utils.**is_iterable**(*x*)
Returns `True` if *x* is an iterable collection.

---

**Note:** This will return `False` if *x* is a string type.

---

sdv.utils.**is_leaf**(*node*)
Returns `True` if *node* has no element children.

sdv.utils.**is_qname**(*string*)
Returns `True` if *string* is a valid QName.

sdv.utils.**is_stix**(*doc*)
Attempts to determine if the input *doc* is a STIX XML instance document. If the root-level element falls under a namespace which starts with `http://stix.mitre.org`, this will return True.

sdv.utils.**is_version_equal**(*x*, *y*)
Attempts to determine if the *x* amd *y* version numbers are semantically equivalent.

#### Examples

The version strings "2.1.0" and "2.1" represent semantically equivalent versions, despite not being equal strings.

**Parameters**

- **x** – A string version number. Ex: '2.1.0'
- **y** – A string version number. Ex: '2.1'

sdv.utils.**is_xml**(*fn*)
Returns `True` if the input filename *fn* ends with an XML extension.

sdv.utils.**iterchildren**(*node*)
Returns an iterator which yields direct child elements of *node*.

sdv.utils.**iterdescendants**(*node*)
Returns an iterator which yields descendant elements of *node*.

sdv.utils.**leaves**(*tree*)
Returns an iterable collection of leaf nodes under *tree*.

sdv.utils.**list_xml_files**(*directory*, *recursive=False*)
Returns a list of file paths for XML files contained within *dir_*.

**Parameters**

- **dir** – A path to a directory.

- **recursive** – If `True`, this function will descend into all subdirectories.

**Returns** A list of XML file paths directly under *dir_*.

`sdv.utils.`**`localname`**(*node*)
Returns the localname for an etree Element *node*.

`sdv.utils.`**`namespace`**(*node*)
Returns the namespace for an etree Element *node*.

`sdv.utils.`**`parse_timestamp`**(*value*)
Attempts to parse *value* into an instance of `datetime.datetime`. If *value* is `None`, this function will return `None`.

**Parameters** **value** – A timestamp. This can be a string or datetime.datetime value.

`sdv.utils.`**`remove_all`**(*list_*, *items*)
Removes all *items* from the *list_*.

`sdv.utils.`**`set_xml_parser`**(*parser*)
Set the XML parser to use internally. This should be an instance of `lxml.etree.XMLParser`.

---

**Note:** Setting *parser* to an object that is not an instance `lxml.etree.XMLParser` may result in undesired behaviors.

---

**Parameters** **parser** – An etree parser.

`sdv.utils.`**`strip_whitespace`**(*string*)
Returns a copy of *string* with all whitespace removed.

CHAPTER 3

# Code Examples

**Version**: 2.3.0

## 3.1 Code Examples

The following sections provide examples of how to perform XML Schema, STIX Profile, and STIX "Best Practices" validation with **stix-validator**.

**Version**: 2.3.0

### 3.1.1 XML Schema Validation

The **stix-validator** library provides methods and data types to help perform STIX and CybOX XML Schema validation. The **stix-validator** library bundles all versions of STIX and CybOX XML Schema files with it, but also supports the ability to validate validate against external directories of schemas or remote, web-accessible schema locations.

The following code examples demonstrate different ways you can utilize the STIX and CyOX XML Schema validation capabilities in **stix-validator**.

**Contents**



#### Validating STIX and CybOX Documents

The **stix-validator** *`sdv.validate_xml()`* method can be used to validate STIX and CybOX XML files or file-like objects.

```
import sdv

# Validate the 'xml-content.xml' STIX/CybOX document using bundled XML schemas.
results = sdv.validate_xml('xml-content.xml')
```

```
# Print the result!
print results.is_valid
```

When validating STIX content, The *sdv.validate_xml()* method acts as a proxy to the *STIXSchemaValidator* class and is equivalent to the following:

```
from sdv.validators import STIXSchemaValidator

# Create the validator instance
validator = STIXSchemaValidator()

# Validate 'xml-content.xml` STIX document using bundled STIX schemas
results = validator.validate('xml-content.xml')

# Print the results!
print results.is_valid
```

When validating CybOX content, the *sdv.validate_xml()* method passes its input to the *CyboxSchemaValidator* class.

The examples above pass the `'xml-content.xml'` filename into *sdv.validate_xml()* and *STIXSchemaValidator.validate()*, but these methods (and all other validation methods) can also accept file-like objects (such as files on disk or `StringIO` instances), `etree._Element` instances, or `etree._ElementTree` instances. Super duper neato!

### Using Non-bundled Schemas

Some STIX data which utilizes STIX extensions may require non-STIX schemas (e.g, OVAL, OpenIOC, etc.) to perform validation. To validate a STIX document which includes non-STIX extension data users can provide a path to a directory containing all the schemas required for validation.

```
import sdv

# Path to a directory containing ALL schema files required for validation
SCHEMA_DIR = "/path/to/schemas/"

# Use the `schemas` parameter to use non-bundled schemas.
results = sdv.validate_xml('stix-content.xml', schemas=SCHEMA_DIR)
```

**Note:** Validating against external schema directories requires that **all** schemas necessary for validation be found under the directory. This includes STIX schemas!

### Using `xsi:schemaLocation`

XML content that contains an `xsi:schemaLocation` attribute referring to external schemas can be validated using the `xsi:schemaLocation` value by making use of the `schemaloc` parameter,

```
import sdv

# Use the xsi:schemaLocation attribute to resolve remote schemas
results = sdv.validate_xml('xml-content.xml', schemaloc=True)

# Print the results!
print results.is_valid
```

### STIX and CybOX Versions

The **stix-validator** is bundled with official STIX and CybOX schemas, which enables validation of STIX and CybOX XML documents without requiring users to provide schemas.

Because we bundle several versions of the STIX and CybOX schemas, the **stix-validator** needs to know what version of STIX or CybOX to validate input documents against. By default, the **stix-validator** will attempt to determine the version of the input STIX/CybOX document by inspecting the document for version information.

If the input document contains no version information, users must declare a version for the STIX/CybOX document via the `version` parameter:

```
import sdv

# Validate the 'stix-content.xml'.
# Declare that the STIX content is STIX v1.2
results = sdv.validate_xml('stix-content.xml', version='1.2')

# Print the result!
print results.is_valid
```

---

**Note:** No version information is required by the **stix-validator** when performing validation against *non-bundled schemas* or when validating using *xsi:schemaLocation*. Any `version` values that are passed in will be **ignored** by the API.

**However**, the schemas may define constraints or requirements for version numbers, so validation errors regarding invalid or missing version numbers are still reported as *XmlSchemaError* objects.

---

#### Unknown Versions

If a version is not passed in nor found on the document, one of the following *UnknownVersionError* implementations are raised:

- *UnknownSTIXVersionError* if validating a STIX document.
- *UnknownCyboxVersionError* if validating a CybOX document.

#### Invalid Versions

If an invalid version is passed in nor found on the document, one of the following *InvalidVersionError* implementations are raised:

- *InvalidSTIXVersionError* if validating a STIX document.
- *InvalidCyboxVersionError* if validating a CybOX document.

### Retrieving XML Schema Validation Errors

The following sections explain how to retrieve XML Schema validation errors from the *XmlValidationResults* class.

#### The XmlValidationResults Class

XML Schema validation results are communicated via the *XmlValidationResults* and *XmlSchemaError* classes. The *sdv.validate_xml()* returns an instance of *XmlValidationResults*.

To determine if a document was valid, users only need to inspect the `is_valid` property:

```
import sdv

# Validate the 'xml-content.xml' input document using bundled schemas.
results = sdv.validate_xml('xml-content.xml')

# Print the result!
print results.is_valid
```

If the `is_valid` property is `False`, users can inspect the `errors` property to retrieve specific validation errors.

The `errors` property on *XmlValidationResults* contains a list of *XmlSchemaError* instances, which hold details about the validation errors and methods for accessing those details.

```
import sdv

results = sdv.validate_xml('xml-content.xml')

# If 'xml-content.xml' is invalid, print each error
if not results.is_valid:
    for error in results.errors:
        print "Line Number:", error.line
        print "Error Message:", error
```

#### Dictionaries and JSON

Users wanting to work with dictionaries or pass around JSON blobs can make use of the *XmlValidationResults.as_dict()* and *XmlValidationResults.as_json()* methods.

```
import sdv

# Validate 'xml-content.xml'
results = sdv.validate_xml('xml-content.xml')

# Retrieve results as dictionary
result_dictionary = results.as_dict()  # returns {'result': True} if valid

# Retrieve results as JSON
result_json = results.as_json() # returns '{"result": true}' JSON if valid
```

**Version**: 2.3.0

### 3.1.2 STIX "Best Practices" Validation

The **stix-validator** library provides methods and data types to help perform STIX Best Practices validation.

> **Warning:** The STIX Best Practices validation capabilities are under active development and do not cover all STIX Best Practices.

The following code examples demonstrate different ways you can utilize the STIX Best Practices validation capabilities in **stix-validator**.

### Validating STIX Documents

The **stix-validator** *sdv.validate_best_practices()* method can be used to validate STIX XML files or file-like objects against STIX Best Practices.

```
import sdv

# Check the 'stix-content.xml' document for STIX Best Practices conformance
results = sdv.validate_best_practices('stix-content.xml')

# Print the result!
print results.is_valid
```

The *sdv.validate_best_practices()* method acts as a proxy to the *STIXBestPracticeValidator* class and is equivalent to the following:

```
from sdv.validators import STIXBestPracticeValidator

# Create the validator instance
validator = STIXBestPracticeValidator()

# Validate 'stix-content.xml` STIX document
results = validator.validate('stix-content.xml')

# Print the results!
print results.is_valid
```

The examples above pass the `'stix-content.xml'` filename into *sdv.validate_profile()* and *STIXProfileValidator.validate()*, but these methods can also accept file-like objects (such as files on disk or `StringIO` instances), `etree._Element` instances, or `etree._ElementTree` instances. Neato!

### Retrieving STIX Best Practice Validation Errors

The following sections explain how to retrieve STIX Best Practices validation errors from the *BestPracticeValidationResults* class.

#### The BestPracticeValidationResults Class

STIX Best Practices validation results are communicated via the *BestPracticeValidationResults*, *BestPracticeWarningCollection*, and *BestPracticeWarning* classes.

The *sdv.validate_best_practices()* and *STIXBestPracticeValidator.validate()* methods both return an instance of *BestPracticeValidationResults*.

To determine if a document was valid, users only need to inspect the `is_valid` property:

```
import sdv

# Check the 'stix-content.xml' document for STIX Best Practices conformance
results = sdv.validate_best_practices('stix-content.xml')

# Print the result!
print results.is_valid
```

If the `is_valid` property is `False`, users can inspect the `errors` property to retrieve specific validation errors, or iterate over the `BestPracticeValidationResults` class directly.

The `errors` property on *`BestPracticeValidationResults`* contains a list of *`BestPracticeWarningCollection`* instances, which hold details about the validation errors and methods for accessing those details.

#### BestPracticeWarnings and Collections

Every deviation from STIX Best Practices within an instance document is represented as an instance of *`BestPracticeWarning`*. These violations are categorized and collected within instances of *`BestPracticeWarningCollection`* instances, which are each assigned names, such as `"Missing Titles"`, or `"Duplicate IDs"`.

The `errors` property on *`BestPracticeValidationResults`* contains a list of *`BestPracticeWarningCollection`* instances, which hold details about the validation errors and methods for accessing those details.

```
import sdv

# Check the 'stix-content.xml' document for STIX Best Practices conformance
results = sdv.validate_best_practices('stix-content.xml')

# If 'stix-content.xml' is invalid, print each error
if not results.is_valid:
    for coll in results.errors:
        print_best_practice_collection(coll)  # User-defined print method
```

The example above iterates over the `result.errors` property, and calls a user-defined, `print_best_practice_collection()`.

This function *could* be defined as the following:

```
def print_best_practice_collection(coll):
    """
    Prints BestPracticeWarningCollection info to stdout.
    """

    # Print the Best Practice Warning collection name
    print coll.name

    # Print the line and XML tag for each non-conformant node in the
    # warning collection.
    for warning in coll:
        print warning.line, warning.tag
```

#### Dictionaries and JSON

Users wanting to work with dictionaries or pass around JSON blobs can make use of the *`BestPracticeValidationResults.as_dict()`* and *`BestPracticeValidationResults.as_json()`* methods.

```
import sdv

# Check the 'stix-content.xml' document for STIX Best Practices conformance
results = sdv.validate_best_practices('stix-content.xml')
```

```
# Retrieve results as dictionary
result_dictionary = results.as_dict()  # returns {'result': True} if valid

# Retrieve results as JSON
result_json = results.as_json() # returns '{"result": true}' JSON if valid
```

**Version**: 2.3.0

### 3.1.3 STIX Profile Validation

The **stix-validator** library provides methods and data types to help perform STIX Profile validation.

**Warning:** The STIX Profile validation capabilities should be considered **experimental** and may change considerably over time.

The following code examples demonstrate different ways you can utilize the STIX Profile validation capabilities in **stix-validator**.

#### Validating STIX Documents

The **stix-validator** *sdv.validate_profile()* method can be used to validate STIX XML files or file-like objects agains a STIX Profile.

```
import sdv

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Validate the 'stix-content.xml' STIX document against the PROFILE doc
results = sdv.validate_profile('stix-content.xml', PROFILE)

# Print the result!
print results.is_valid
```

The *sdv.validate_profile()* method acts as a proxy to the *STIXProfileValidator* class and is equivalent to the following:

```
from sdv.validators import STIXProfileValidator

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Create the validator instance for PROFILE
validator = STIXProfileValidator(PROFILE)

# Validate 'stix-content.xml` STIX document against the PROFILE
results = validator.validate('stix-content.xml')

# Print the results!
print results.is_valid
```

**Note:** When validating multiple documents against a STIX Profile, using the *STIXProfileValidator* will be faster than *sdv.validate_profile()* since *sdv.validate_profile()* needs to parse the STIX Profile with each invocation.

The examples above pass the `'stix-content.xml'` filename into *sdv.validate_profile()* and *STIXProfileValidator.validate()*, but these methods can also accept file-like objects (such as files on disk or `StringIO` instances), `etree._Element` instances, or `etree._ElementTree` instances. Neato!

### Retrieving STIX Profile Validation Errors

The following sections explain how to retrieve STIX Profile validation errors from the *ProfileValidationResults* class.

#### The ProfileValidationResults Class

STIX Profile validation results are communicated via the *ProfileValidationResults* and *ProfileError* classes.

The *sdv.validate_profile()* and *STIXProfileValidator.validate()* methods both return an instance of *ProfileValidationResults*.

To determine if a document was valid, users only need to inspect the `is_valid` property:

```
import sdv

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Validate the 'stix-content.xml' STIX document against the PROFILE doc
results = sdv.validate_profile('stix-content.xml', PROFILE)

# Print the result!
print results.is_valid
```

If the `is_valid` property is `False`, users can inspect the `errors` property to retrieve specific validation errors.

The `errors` property on *ProfileValidationResults* contains a list of *ProfileError* instances, which hold details about the validation errors and methods for accessing those details.

```
import sdv

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Validate the 'stix-content.xml' STIX document against the PROFILE doc
results = sdv.validate_profile('stix-content.xml', PROFILE)

# If 'stix-content.xml' is invalid, print each error
if not results.is_valid:
    for error in results.errors:
        print "Line Number:", error.line
        print "Error Message:", error
```

#### Dictionaries and JSON

Users wanting to work with dictionaries or pass around JSON blobs can make use of the *ProfileValidationResults.as_dict()* and *ProfileValidationResults.as_json()* methods.

```
import sdv

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Validate the 'stix-content.xml' STIX document against the PROFILE doc
results = sdv.validate_profile('stix-content.xml', PROFILE)

# Retrieve results as dictionary
result_dictionary = results.as_dict()  # returns {'result': True} if valid

# Retrieve results as JSON
result_json = results.as_json() # returns '{"result": true}' JSON if valid
```

### Converting STIX Profiles to XSLT and Schematron

STIX Profiles are currently defined using multi-worksheet Excel documents. The **stix-validator** API provides methods for converting Excel documents into ISO Schematron and XSLT documents.

```
import sdv

# STIX Profile filename
PROFILE = "/path/to/stix/profile.xlsx"

# Convert the STIX Profile into a Schematron document
schematron = sdv.profile_to_schematron(PROFILE)

# Convert the STIX Profile into an XSLT document
xslt = sdv.profile_to_xslt(PROFILE)

# Write the returned Scheamtron document to a file
schematron.write(
    "/path/to/output/filename.sch",  # Output Schematron file path
    pretty_print=True,               # Pretty print the file (not necessary)
    xml_declaration=True,            # Write out <?xml version="1.0" encoding="UTF-8"?>
    encoding="UTF-8"                 # Set the encoding to UTF-8
)

# Write out the returned XSLT document to a file
xslt.write(
    "/path/to/output/filename.xslt", # Output XSLT file path
    pretty_print=True,               # Pretty print the file (not necessary)
    xml_declaration=True,            # Write out <?xml version="1.0" encoding="UTF-8"?>
    encoding="UTF-8"                 # Set the encoding to UTF-8
)
```

CHAPTER 4

# Contributing

If a bug is found, a feature is missing, or something just isn't behaving the way you'd expect it to, please submit an issue to our tracker. If you'd like to contribute code to our repository, you can do so by issuing a pull request and we will work with you to try and integrate that code into our repository.

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

## S

## A



## B



## C



## D



## E



## G

## V



## X