python-maec Documentation

Release 4.1.0.13

The MITRE Corporation

March 07, 2017

Contents

Version: 4.1.0.13

The python-maec library provides an API for developing and consuming Malware Attribute Enumeration and Characterization (MAEC) content. Developers can leverage the API to create applications that create, consume, translate, or otherwise work with MAEC content.

CHAPTER 1

Versions

Each version of python-maec is designed to work with a single version of the MAEC Language. The table below shows the latest version the library for each version of MAEC.

MAEC Version	python-maec Version
4.1	4.1.0.12 (PyPI) (GitHub)
4.0	4.0.1.0 (PyPI) (GitHub)
3.0	3.0.0b1 (PyPI) (GitHub)

CHAPTER 2

Contents

Version: 4.1.0.13

Getting Started with python-maec

Note: The python-maec library is intended for developers who want to add MAEC support to existing programs or create new programs that handle MAEC content. Experience with Python development is assumed.

Other users should look at existing tools that support MAEC.

Understanding XML, XML Schema, and the MAEC language is also incredibly helpful when using python-maec in an application.

First, you should follow the *Installation* procedures.

Your First MAEC Application

Once you have installed python-maec, you can begin writing Python applications that consume or create STIX content!

Note: The *python-maec* library provides **bindings** and **APIs**, both of which can be used to parse and write MAEC XML files. For in-depth description of the *APIs, bindings, and the differences between the two*, please refer to APIs or bindings?

Creating a MAEC Package

```
from maec.package import Package                # Import the MAEC Package API
from maec.package import MalwareSubject         # Import the MAEC Malware Subject API

package = Package()                             # Create an instance of Package
malware_subject = MalwareSubject()              # Create an instance of MalwareSubject
package.add_malware_subject(malware_subject)    # Add the Malware Subject to the Package

print(package.to_xml())                         # Print the XML for this MAEC Package
```

Parsing MAEC XML

```
import maec                                    # Import the python-maec API

fn = 'sample_maec_package.xml'                 # generate by running examples\package_generation_example
maec_objects = maec.parse_xml_instance(fn)     # Parse using the from_xml() method
api_object = maec_objects['api']               # Get the API object from the parsed objects
```

Example Scripts

The python-maec repository contains several example scripts that help illustrate the capabilities of the APIs. These scripts are simple command line utilities that can be executed by passing the name of the script to a Python interpreter.

```
$ python package_generation_example.py
```

Writing Your Own Application

See the *Examples* page for more examples of using python-maec in your own application.

Version: 4.1.0.13

Installation

The installation of python-maec can be accomplished through a few different workflows.

Recommended Installation

Use pypi and pip:

```
$ pip install maec
```

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

Dependencies

The python-maec library relies on some non-standard Python libraries for the processing of MAEC content. Revisions of python-maec 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 python-maec:

- lxml - A Pythonic binding for the C libraries **libxml2** and **libxslt**.
- python-cybox - A library for consuming and producing CybOX content.
- python-dateutil - A library for parsing datetime information.

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.

Manual Installation

If you are unable to use pip, you can also install python-maec 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 MAEC installation. Make sure you check to ensure the versions you install are compatible with the version of MAEC you plan to install.
2. Download the desired version of MAEC from PyPI or the GitHub releases page. The steps below assume you are using the 4.1.0.13 release.
3. Extract the downloaded file. This will leave you with a directory named MAEC-4.1.0.13.

```
$ tar -zxf MAEC-4.1.0.13.tar.gz
$ ls
MAEC-4.1.0.13 MAEC-4.1.0.13.tar.gz
```

OR

```
$ unzip MAEC-4.1.0.13.zip
$ ls
MAEC-4.1.0.13 MAEC-4.1.0.13.zip
```

4. Run the installation script.

```
$ cd MAEC-4.1.0.13
$ python setup.py install
```

5. Test the installation.

```
$ python
Python 2.7.6 (default, Mar 22 2015, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import MAEC
>>>
```

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

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: 4.1.0.13

Overview

This page provides a quick overview needed to understand the inner workings of the python-maec library. If you prefer a more hands-on approach, browse the Examples.

MAEC Entities

Each type within MAEC is represented by a class which derives from *maec.Entity*. In general, there is one Python class per MAEC type, though in some cases classes which would have identical functionality have been reused rather than writing duplicating classes. One example of this is that many enumerated values are implemented using the `cybox.common.properties.String`, since values aren't checked to make sure they are valid enumeration values.

Note: Not all MAEC types have yet been implemented.

Version: 4.1.0.13

Examples

This page includes some basic examples of creating and parsing MAEC content.

There are a couple things we do in these examples for purposes of demonstration that shouldn't be done in production code:

- When calling `to_xml()`, we use `include_namespaces=False`. This is to make the example output easier to read, but means the resulting output cannot be successfully parsed. The XML parser doesn't know what namespaces to use if they aren't included. In production code, you should explicitly set `include_namespaces` to `True` or omit it entirely (`True` is the default).
- We use `set_id_method(IDGenerator.METHOD_INT)` to make IDs for Malware Subjects and Actions easier to read and cross-reference within the XML document. In production code, you should omit this statement, which causes random UUIDs to be created instead, or create explicit IDs yourself for Malware Subjects and Actions.

Creating Packages

The most commonly used MAEC output format is the MAEC Package, which can contain one or more Malware Subjects. Malware Subjects (discussed in more detail below) encompass all of the data for a single malware instance, including that from different types of analysis.

```
from mixbox.idgen import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)

from maec.package import Package, MalwareSubject

p = Package()
ms = MalwareSubject()
p.add_malware_subject(ms)

print(p.to_xml(include_namespaces=False, encoding=None))
```

Which outputs:

```
<maecPackage:MAEC_Package id="example:package-1" schema_version="2.1">
    <maecPackage:Malware_Subjects>
        <maecPackage:Malware_Subject id="example:malware_subject-2">
        </maecPackage:Malware_Subject>
    </maecPackage:Malware_Subjects>
</maecPackage:MAEC_Package>
```

Creating Malware Subjects

The easiest way to create a Malware Subject is to construct one and then set various properties on it. The Malware_Instance_Object_Attributes field on a Malware Subject MUST be set in order to identify the particular malware instance that it is characterizing.

```
from mixbox.idgen import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)

from cybox.core import Object
from cybox.objects.file_object import File
from maec.package import MalwareSubject

ms = MalwareSubject()
ms.malware_instance_object_attributes = Object()
ms.malware_instance_object_attributes.properties = File()
ms.malware_instance_object_attributes.properties.file_name = "malware.exe"
ms.malware_instance_object_attributes.properties.file_path = "C:\Windows\Temp\malware.exe"
print(ms.to_xml(include_namespaces=False, encoding=None))
```

Which outputs:

```
<maecPackage:MalwareSubjectType id="example:malware_subject-1">
    <maecPackage:Malware_Instance_Object_Attributes id="example:Object-2">
        <cybox:Properties xsi:type="FileObj:FileObjectType">
            <FileObj:File_Name>malware.exe</FileObj:File_Name>
            <FileObj:File_Path>C:\Windows\Temp\malware.exe</FileObj:File_Path>
        </cybox:Properties>
    </maecPackage:Malware_Instance_Object_Attributes>
</maecPackage:MalwareSubjectType>
```

Creating Bundles

In MAEC, the `Bundle` represents a container for capturing the results from a particular malware analysis that was performed on a malware instance. While a `Bundle` is most commonly included as part of a Malware Subject, it can also be used a standalone output format when only malware analysis results for a malware instance wish to be shared. We'll cover both cases here.

Creating Standalone Bundles

Standalone Bundles function very similarly to Malware Subjects. Therefore, the easiest way to create a standalone Bundle is to construct one and then set various properties on it. The Malware_Instance_Object_Attributes field on a standalone Bundle MUST be set in order to identify the particular malware instance that it is characterizing.

```
from mixbox.idgen import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)
```

```
from cybox.core import Object
from cybox.objects.file_object import File
from maec.bundle import Bundle

b = Bundle()
b.malware_instance_object_attributes = Object()
b.malware_instance_object_attributes.properties = File()
b.malware_instance_object_attributes.properties.file_name = "malware.exe"
b.malware_instance_object_attributes.properties.file_path = "C:\Windows\Temp\malware.exe"

print(b.to_xml(include_namespaces=False, encoding=None))
```

Which outputs:

```
<maecBundle:MAEC_Bundle defined_subject="false" id="example:bundle-1" schema_version="4.1">
    <maecBundle:Malware_Instance_Object_Attributes id="example:Object-2">
        <cybox:Properties xsi:type="FileObj:FileObjectType">
            <FileObj:File_Name>malware.exe</FileObj:File_Name>
            <FileObj:File_Path>C:\Windows\Temp\malware.exe</FileObj:File_Path>
        </cybox:Properties>
    </maecBundle:Malware_Instance_Object_Attributes>
</maecBundle:MAEC_Bundle>
```

Creating and adding Bundles to a Malware Subject

Bundles in a Malware Subject are defined nearly identically to those of the standalone variety, with the sole exception that they do not require their Malware_Instance_Object_Attributes field to be set, since this would already be defined in their parent Malware Subject.

```
from mixbox.idgen import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)

from cybox.core import Object
from cybox.objects.file_object import File

from maec.package import MalwareSubject
from maec.bundle import Bundle

ms = MalwareSubject()
ms.malware_instance_object_attributes = Object()
ms.malware_instance_object_attributes.properties = File()
ms.malware_instance_object_attributes.properties.file_name = "malware.exe"
ms.malware_instance_object_attributes.properties.file_path = "C:\Windows\Temp\malware.exe"

b = Bundle()
ms.add_findings_bundle(b)

print(ms.to_xml(include_namespaces=False, encoding=None))
```

Which outputs:

```
<maecPackage:MalwareSubjectType id="example:malware_subject-1">
    <maecPackage:Malware_Instance_Object_Attributes id="example:Object-2">
        <cybox:Properties xsi:type="FileObj:FileObjectType">
            <FileObj:File_Name>malware.exe</FileObj:File_Name>
            <FileObj:File_Path>C:\Windows\Temp\malware.exe</FileObj:File_Path>
        </cybox:Properties>
```

```
    </maecPackage:Malware_Instance_Object_Attributes>
    <maecPackage:Findings_Bundles>
        <maecPackage:Bundle defined_subject="false" id="example:bundle-3" schema_version="4.1"/>
    </maecPackage:Findings_Bundles>
</maecPackage:MalwareSubjectType>
```

Creating and adding Actions to a Bundle

MAEC uses its `MalwareAction` to capture the low-level dynamic entities, such as API calls or their abstractions, performed by malware. A `MalwareAction` is stored in a Bundle (either standalone or embedded in a Malware Subject, as discussed above). As with the other MAEC entities, the easiest way to use the `MalwareAction` is to instantiate it and then set various properties on it as needed.

```
from mixbox.idgen import IDGenerator, set_id_method
set_id_method(IDGenerator.METHOD_INT)

from cybox.core import Object, AssociatedObjects, AssociatedObject
from cybox.objects.file_object import File
from cybox.common import VocabString
from maec.bundle import Bundle
from maec.bundle import MalwareAction

b = Bundle()
a = MalwareAction()
ao = AssociatedObject()

ao.properties = File()
ao.properties.file_name = "badware.exe"
ao.properties.size_in_bytes = "123456"
ao.association_type = VocabString()
ao.association_type.value = 'output'
ao.association_type.xsi_type = 'maecVocabs:ActionObjectAssociationTypeVocab-1.0'

a.name = VocabString()
a.name.value = 'create file'
a.name.xsi_type = 'maecVocabs:FileActionNameVocab-1.0'
a.associated_objects = AssociatedObjects()
a.associated_objects.append(ao)

b.add_action(a)

print(b.to_xml(include_namespaces = False, encoding=None))
```

```
<maecBundle:MAEC_Bundle defined_subject="false" id="example:bundle-1" schema_version="4.1">
    <maecBundle:Actions>
        <maecBundle:Action id="example:action-2">
            <cybox:Name xsi:type="maecVocabs:FileActionNameVocab-1.0">create file</cybox:Name>
            <cybox:Associated_Objects>
                <cybox:Associated_Object id="example:Object-3">
                    <cybox:Properties xsi:type="FileObj:FileObjectType">
                        <FileObj:File_Name>badware.exe</FileObj:File_Name>
                        <FileObj:Size_In_Bytes>123456</FileObj:Size_In_Bytes>
                    </cybox:Properties>
                    <cybox:Association_Type xsi:type="maecVocabs:ActionObjectAssociationTypeVocab-1.0
                </cybox:Associated_Object>
            </cybox:Associated_Objects>
        </maecBundle:Action>
```

```
    </maecBundle:Actions>
</maecBundle:MAEC_Bundle>
```

Version: 4.1.0.13

APIs or bindings?

This page describes both the **APIs** and the **bindings** provided by the *python-maec* library.

Overview

The python-maec library provides APIs and utilities that aid in the creation, consumption, and processing of Structured Threat Information eXpression (MAEC) content. The APIs that drive much of the functionality of python-maec sit on top of a binding layer that acts as a direct connection between Python and the MAEC XML. Because both the APIs and the bindings allow for the creation and development of MAEC content, developers that are new to python-maec may not understand the differences between the two. This document aims to identify the purpose and uses of the APIs and bindings.

Bindings

The python-maec library leverages machine generated XML-to-Python bindings for the creation and processing of MAEC content. These bindings are created using the generateDS utility and can be found under maec.bindings within the package hierarchy.

The MAEC bindings allow for a direct, complete mapping between Python classes and MAEC XML Schema data structures. That being said, it is possible (though not advised) to use only the MAEC bindings to create MAEC documents. However, because the code is generated from XML Schema without contextual knowledge of relationships or broader organizational/developmental schemes, it is often a cumbersome and laborious task to create even the simplest of MAEC documents.

Developers within the python-maec team felt that the binding code did not lend itself to rapid development or natural navigation of data, and so it was decided that a higher-level API should be created.

APIs

The python-maec APIs are classes and utilities that leverage the MAEC bindings for the creation and processing of MAEC content. The APIs are designed to behave more naturally when working with MAEC content, allowing developers to conceptualize and interact with MAEC documents as pure Python objects and not XML Schema objects.

The APIs provide validation of inputs, multiple input and output formats, more Pythonic access of data structure internals and interaction with classes, and better interpretation of a developers intent through datatype coercion and implicit instantiation.

Note: The python-maec APIs are under constant development. Our goal is to provide full API coverage of the MAEC data structures, but not all structures are exposed via the APIs yet. Please refer to the API Documentation for API coverage details.

Brevity Wins

The two code examples show the difference in creating and printing a simple MAEC document consisting of only a MAEC Bundle with a single Malware Action using the python-maec and python-cybox bindings. Both examples will produce the same MAEC XML!

API Example

```
# Import the required APIs
from maec.bundle import Bundle, MalwareAction
from maec.utils import IDGenerator, set_id_method
from cybox.core import Object, AssociatedObjects, AssociatedObject
from cybox.objects.file_object import File
from cybox.common import VocabString

# Instantiate the MAEC/CybOX Entities
set_id_method(IDGenerator.METHOD_INT)
b = Bundle()
a = MalwareAction()
ao = AssociatedObject()

# Build the Associated Object for use in the Action
ao.properties = File()
ao.properties.file_name = "badware.exe"
ao.properties.size_in_bytes = "123456"
ao.association_type = VocabString()
ao.association_type.value = 'output'
ao.association_type.xsi_type = 'maecVocabs:ActionObjectAssociationTypeVocab-1.0'

# Build the Action and add the Associated Object to it
a.name = VocabString()
a.name.value = 'create file'
a.name.xsi_type = 'maecVocabs:FileActionNameVocab-1.0'
a.associated_objects = AssociatedObjects()
a.associated_objects.append(ao)

# Add the Action to the Bundle
b.add_action(a)

# Output the Bundle to stdout
print(b.to_xml(include_namespaces = False))
```

Binding Example

```
import sys
# Import the required bindings
import maec.bindings.maec_bundle as bundle_binding
import cybox.bindings.cybox_core as cybox_core_binding
import cybox.bindings.cybox_common as cybox_common_binding
import cybox.bindings.file_object as file_binding

# Instantiate the MAEC/CybOX Entities
b = bundle_binding.BundleType(id="bundle-1")
a = bundle_binding.MalwareActionType(id="action-1")
ao = cybox_core_binding.AssociatedObjectType(id="object-1")

# Build the Associated Object for use in the Action
f = file_binding.FileObjectType()
f_name = cybox_common_binding.StringObjectPropertyType(valueOf_="badware.exe")
```

```
f.set_File_Name(f_name)
f_size = cybox_common_binding.UnsignedLongObjectPropertyType(valueOf_="123456")
f.set_Size_In_Bytes(f_size)
f.set_xsi_type = "FileObj:FileObjectType"
ao.set_Properties(f)
ao_type = cybox_common_binding.ControlledVocabularyStringType(valueOf_="output")
ao_type.set_xsi_type("maecVocabs:ActionObjectAssociationTypeVocab-1.0")
ao.set_Association_Type(ao_type)

# Build the Action and add the Associated Object to it
a_name = cybox_common_binding.ControlledVocabularyStringType(valueOf_="create file")
a_name.set_xsi_type("maecVocabs:FileActionNameVocab-1.0")
a.set_Name(a_name)
as_objects = cybox_core_binding.AssociatedObjectsType()
as_objects.add_Associated_Object(ao)
a.set_Associated_Objects(as_objects)

# Add the Action to the Bundle
action_list = bundle_binding.ActionListType()
action_list.add_Action(a)
b.set_Actions(action_list)

# Output the Bundle to stdout
b.export(sys.stdout, 0)
```

Feedback

If there is a problem with the APIs or bindings, or if there is functionality missing from the APIs that forces the use of the bindings, let us know in the python-maec issue tracker

Version: 4.1.0.13

Contributing

If you notice a bug, have a suggestion for a new feature, or find that that something just isn't behaving the way you'd expect it to, please submit an issue to our issue 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. Users who want to contribute code to the python-maec repository should be familiar with git and the GitHub pull request process.

CHAPTER 3

API Reference

Version: 4.1.0.13

API Documentation

The *python-maec* APIs are the recommended tools for reading, writing, and manipulating MAEC XML documents.

Note: The python-maec APIs are currently under development. As such, API coverage of MAEC data constructs is incomplete; please bear with us as we work toward complete coverage. This documentation also serves to outline current API coverage.

MAEC – Modules located in the base maec package

Version: 4.1.0.13

`maec` Module

Classes

class `maec.`**`Entity`**

Bases: `mixbox.entities.Entity`

Base class for all classes in the MAEC SimpleAPI.

`to_xml_file`(*file*, *namespace_dict=None*, *custom_header=None*)

Export an object to an XML file. Only supports Package or Bundle objects at the moment.

Parameters

- **`file`** – the name of a file or a file-like object to write the output to.
- **`namespace_dict`** – a dictionary of mappings of additional XML namespaces to prefixes.
- **`custom_header`** – a string, list, or dictionary that represents a custom XML header to be written to the output.

class `maec.`**`EntityList`**(**args*)

Bases: `_abcoll.MutableSequence`, `mixbox.entities.Entity`

An EntityList is an Entity that behaves like a mutable sequence.

EntityList implementations must define one multiple TypedField which has an Entity subclass type. EntityLists can define other TypedFields that are not multiple.

The MutableSequence methods are used to interact with the multiple TypedField.

classmethod **list_from_object**(*entitylist_obj*)
Convert from object representation to list representation.

classmethod **object_from_list**(*entitylist_list*)
Convert from list representation to object representation.

MAEC Bundle – Modules located in the maec.bundle package

Version: 4.1.0.13

maec.bundle.action_reference_list Module

Classes

class maec.bundle.action_reference_list.**ActionReferenceList**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.bundle.av_classification Module

Classes

class maec.bundle.av_classification.**AVClassification**(*classification=None*, *tool_name=None*, *tool_vendor=None*)
Bases: cybox.common.tools.ToolInformation, *maec.Entity*

class maec.bundle.av_classification.**AVClassifications**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.bundle.behavior Module

Classes

class maec.bundle.behavior.**Behavior**(*id=None*, *description=None*)
Bases: *maec.Entity*

class maec.bundle.behavior.**BehavioralActionEquivalenceReference**
Bases: *maec.Entity*

class maec.bundle.behavior.**BehavioralActionReference**(*action_id=None*)
Bases: cybox.core.action_reference.ActionReference

class maec.bundle.behavior.**BehavioralAction**
Bases: *maec.Entity*

class maec.bundle.behavior.**BehavioralActions**
Bases: *maec.Entity*

class maec.bundle.behavior.**PlatformList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.behavior.**CVEVulnerability**
Bases: *maec.Entity*

class maec.bundle.behavior.**Exploit**
Bases: *maec.Entity*

class maec.bundle.behavior.**BehaviorPurpose**
Bases: *maec.Entity*

class maec.bundle.behavior.**AssociatedCode**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.bundle.behavior_reference Module

Classes

class maec.bundle.behavior_reference.**BehaviorReference**(*behavior_idref=None*)
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.bundle Module

Classes

class maec.bundle.bundle.**Bundle**(*id=None*, *defined_subject=False*, *schema_version='4.1'*, *content_type=None*, *malware_instance_object=None*)
Bases: *maec.Entity*

add_action(*action*, *action_collection_name=None*)
Add an Action to an existing named Action Collection in the Collections entity. If it does not exist, add it to the top-level Actions entity.

add_av_classification(*av_classification*)
Add an AV Classification to the top-level AV_Classifications entity in the Bundle.

add_behavior(*behavior*, *behavior_collection_name=None*)
Add a Behavior to an existing named Behavior Collection in the Collections entity. If it does not exist, add it to the top-level Behaviors entity.

add_candidate_indicator(*candidate_indicator*, *candidate_indicator_collection_name=None*)
Add a Candidate Indicator to an existing named Candidate Indicator Collection in the Collections entity. If it does not exist, add it to the top-level Candidate Indicators entity.

add_capability(*capability*)
Add a Capability to the top-level Capabilities entity in the Bundle.

add_named_action_collection(*collection_name*, *collection_id=None*)
Add a new named Action Collection to the top-level Collections entity in the Bundle.

add_named_behavior_collection(*collection_name*, *collection_id=None*)
Add a new named Behavior Collection to the Collections entity in the Bundle.

add_named_candidate_indicator_collection(*collection_name*, *collection_id=None*)
Add a new named Candidate Indicator Collection to the Collections entity in the Bundle.

add_named_object_collection(*collection_name*, *collection_id=None*)
Add a new named Object Collection to the Collections entity in the Bundle.

add_object(*object*, *object_collection_name=None*)
Add an Object to an existing named Object Collection in the Collections entity. If it does not exist, add it to the top-level Object entity.

classmethod compare(*bundle_list*, *match_on=None*, *case_sensitive=True*)
Compare the Bundle to a list of other Bundles, returning a BundleComparator object.

deduplicate()
Deduplicate all Objects in the Bundle. Add duplicate Objects to new "Deduplicated Objects" Object Collection, and replace duplicate entries with references to corresponding Object.

dereference_objects(*extra_objects=[]*)
Dereference any Objects in the Bundle by replacing them with the entities they reference.

get_action_objects(*action_name_list*)
Get all Objects corresponding to one or more types of Actions, specified via a list of Action names.

get_all_actions(*bin=False*)
Return a list of all Actions in the Bundle.

get_all_actions_on_object(*object*)
Return a list of all of the Actions in the Bundle that operate on a particular input Object.

get_all_multiple_referenced_objects()
Return a list of all Objects in the Bundle that are referenced more than once.

get_all_non_reference_objects()
Return a list of all Objects in the Bundle that are not references (i.e. all of the actual Objects in the Bundle).

get_all_objects(*include_actions=False*)
Return a list of all Objects in the Bundle.

get_object_by_id(*id*, *extra_objects=[]*, *ignore_actions=False*)
Find and return the Entity (Action, Object, etc.) with the specified ID.

get_object_history()
Build and return the Object history for the Bundle.

normalize_objects()
Normalize all Objects in the Bundle, using the CybOX normalize module.

set_malware_instance_object_attributes(*malware_instance_object*)
Set the top-level Malware Instance Object Attributes entity in the Bundle.

set_process_tree(*process_tree*)
Set the Process Tree, in the top-level <Process_Tree> element.

class maec.bundle.bundle.**ActionList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.bundle.**BehaviorList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.bundle.**ObjectList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.bundle.**BaseCollection**(*name=None*)
Bases: *maec.Entity*

class maec.bundle.bundle.**ActionCollection**(*name=None*, *id=None*)
Bases: *maec.bundle.bundle.BaseCollection*

add_action(*action*)
Add an input Action to the Collection.

class maec.bundle.bundle.**BehaviorCollection**(*name=None*, *id=None*)
Bases: *maec.bundle.bundle.BaseCollection*

add_behavior(*behavior*)
Add an input Behavior to the Collection.

class maec.bundle.bundle.**ObjectCollection**(*name=None*, *id=None*)
Bases: *maec.bundle.bundle.BaseCollection*

add_object(*object*)
Add an input Object to the Collection.

class maec.bundle.bundle.**CandidateIndicatorCollection**(*name=None*, *id=None*)
Bases: *maec.bundle.bundle.BaseCollection*

add_candidate_indicator(*candidate_indicator*)
Add an input Candidate Indicator to the Collection.

class maec.bundle.bundle.**BehaviorCollectionList**
Bases: mixbox.entities.EntityList

get_named_collection(*collection_name*)
Return a specific named Collection from the list, based on its name.

has_collection(*collection_name*)
Checks for the existence of a specific named Collection in the list, based on the its name.

class maec.bundle.bundle.**ActionCollectionList**
Bases: mixbox.entities.EntityList

get_named_collection(*collection_name*)
Return a specific named Collection from the list, based on its name.

has_collection(*collection_name*)
Checks for the existence of a specific named Collection in the list, based on the its name.

class maec.bundle.bundle.**ObjectCollectionList**
Bases: mixbox.entities.EntityList

get_named_collection(*collection_name*)
Return a specific named Collection from the list, based on its name.

has_collection(*collection_name*)
Checks for the existence of a specific named Collection in the list, based on the its name.

class maec.bundle.bundle.**CandidateIndicatorCollectionList**
Bases: mixbox.entities.EntityList

get_named_collection(*collection_name*)
Return a specific named Collection from the list, based on its name.

has_collection(*collection_name*)
Checks for the existence of a specific named Collection in the list, based on the its name.

class maec.bundle.bundle.**Collections**
Bases: *maec.Entity*

add_named_action_collection(*action_collection_name*, *collection_id=None*)
Add a new named Action Collection to the Collections instance.

add_named_behavior_collection(*behavior_collection_name*, *collection_id=None*)
Add a new named Behavior Collection to the Collections instance.

add_named_candidate_indicator_collection(*candidate_indicator_collection_name*, *collection_id=None*)
Add a new named Candidate Indicator Collection to the Collections instance.

add_named_object_collection(*object_collection_name*, *collection_id=None*)
Add a new named Object Collection to the Collections instance.

has_content()
Returns true if any Collections instance inside of the Collection has len > 0.

class maec.bundle.bundle.**BehaviorReference**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.bundle_reference Module

Classes

class maec.bundle.bundle_reference.**BundleReference**(*bundle_idref=None*)
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.candidate_indicator Module

Classes

class maec.bundle.candidate_indicator.**CandidateIndicator**(*id=None*)
Bases: *maec.Entity*

class maec.bundle.candidate_indicator.**CandidateIndicatorList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.candidate_indicator.**CandidateIndicatorComposition**
Bases: *maec.Entity*

class maec.bundle.candidate_indicator.**MalwareEntity**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.capability Module

Classes

class maec.bundle.capability.**Capability**(*id=None*, *name=None*)
Bases: *maec.Entity*

add_strategic_objective(*strategic_objective*)
Add a Strategic Objective to the Capability.

add_tactical_objective(*tactical_objective*)
Add a Tactical Objective to the Capability.

class maec.bundle.capability.**CapabilityObjective**(*id=None*)
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityProperty**
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityRelationship**
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityObjectiveRelationship**
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityReference**
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityObjectiveReference**
Bases: *maec.Entity*

class maec.bundle.capability.**CapabilityList**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.malware_action Module

Classes

class maec.bundle.malware_action.**MalwareAction**
Bases: cybox.core.action.Action

class maec.bundle.malware_action.**ActionImplementation**
Bases: *maec.Entity*

class maec.bundle.malware_action.**APICall**
Bases: *maec.Entity*

class maec.bundle.malware_action.**ParameterList**(**args*)
Bases: mixbox.entities.EntityList

class maec.bundle.malware_action.**Parameter**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.bundle.object_history Module

Classes

class maec.bundle.object_history.**ObjectHistory**
Bases: object

classmethod build(*bundle*)
Build the Object History for a Bundle

class maec.bundle.object_history.**ObjectHistoryEntry**(*object=None*)
Bases: object

get_action_context()
Return a list of the Actions that operated on the Object, via their names, along with the Association_Type used in the Action.

get_action_names()
Return a list of the Actions that operated on the Object, via their names

Version: 4.1.0.13

maec.bundle.object_reference Module

Classes

class maec.bundle.object_reference.**ObjectReference**(*object_idref=None*)
Bases: *maec.Entity*

class maec.bundle.object_reference.**ObjectReferenceList**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.bundle.process_tree Module

Classes

class maec.bundle.process_tree.**ProcessTree**(*root_process=None*)
Bases: *maec.Entity*

set_root_process(*root_process*)
Set the Root Process node of the Process Tree entity.

class maec.bundle.process_tree.**ProcessTreeNode**(*id=None*, *parent_action_idref=None*)
Bases: cybox.objects.process_object.Process

add_initiated_action(*action_id*)
Add an initiated Action to the Process Tree node, based on its ID.

add_injected_process(*process_node*, *process_id=None*)
Add an injected process to the Process Tree node, either directly or to a particular process embedded in the node based on its ID.

add_spawned_process(*process_node*, *process_id=None*)
Add a spawned process to the Process Tree node, either directly or to a particular process embedded in the node based on its ID.

find_embedded_process(*process_id*)
Find a Process embedded somewhere in the Process Tree node tree, based on its ID.

set_id(*id*)
Set the ID of the Process Tree node.

set_parent_action(*parent_action_id*)
Set the ID of the parent action of the Process Tree node.

MAEC Package – Modules located in the maec.package package

Version: 4.1.0.13

maec.package.action_equivalence Module

Classes

class maec.package.action_equivalence.**ActionEquivalence**
Bases: *maec.Entity*

class maec.package.action_equivalence.**ActionEquivalenceList**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.package.analysis Module

Classes

class maec.package.analysis.**Analysis**(*id=None*, *method=None*, *type=None*, *findings_bundle_reference=[]*)
Bases: *maec.Entity*

class maec.package.analysis.**AnalysisEnvironment**
Bases: *maec.Entity*

class maec.package.analysis.**NetworkInfrastructure**
Bases: *maec.Entity*

class maec.package.analysis.**CapturedProtocolList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.analysis.**CapturedProtocol**
Bases: *maec.Entity*

class maec.package.analysis.**AnalysisSystemList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.analysis.**AnalysisSystem**
Bases: cybox.objects.system_object.System

class maec.package.analysis.**InstalledPrograms**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.analysis.**HypervisorHostSystem**
Bases: cybox.objects.system_object.System

class maec.package.analysis.**DynamicAnalysisMetadata**
Bases: *maec.Entity*

class maec.package.analysis.**ToolList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.analysis.**CommentList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.analysis.**Comment**(*value=None*)
Bases: cybox.common.structured_text.StructuredText

is_plain()
Whether this can be represented as a string rather than a dictionary

class maec.package.analysis.**Source**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.package.grouping_relationship Module

Classes

class maec.package.grouping_relationship.**GroupingRelationship**
Bases: *maec.Entity*

class maec.package.grouping_relationship.**GroupingRelationshipList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.grouping_relationship.**ClusteringMetadata**
Bases: *maec.Entity*

class maec.package.grouping_relationship.**ClusteringAlgorithmParameters**
Bases: *maec.Entity*

class maec.package.grouping_relationship.**ClusterComposition**
Bases: *maec.Entity*

class maec.package.grouping_relationship.**ClusterEdgeNodePair**
Bases: *maec.Entity*

Version: 4.1.0.13

maec.package.malware_subject Module

Classes

class maec.package.malware_subject.**MalwareSubject**(*id=None*, *malware_instance_object_attributes=None*)
Bases: *maec.Entity*

deduplicate_bundles()
DeDuplicate all Findings Bundles in the Malware Subject. For now, only handles Objects

dereference_bundles()
Dereference all Findings Bundles in the Malware Subject. For now, only handles Objects

normalize_bundles()
Normalize all Findings Bundles in the Malware Subject. For now, only handles Objects

class maec.package.malware_subject.**MalwareSubjectList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.malware_subject.**MalwareConfigurationDetails**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareConfigurationObfuscationDetails**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareConfigurationObfuscationAlgorithm**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareConfigurationStorageDetails**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareBinaryConfigurationStorageDetails**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareConfigurationParameter**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareDevelopmentEnvironment**
Bases: *maec.Entity*

class maec.package.malware_subject.**FindingsBundleList**
Bases: *maec.Entity*

class maec.package.malware_subject.**MetaAnalysis**
Bases: *maec.Entity*

class maec.package.malware_subject.**MalwareSubjectRelationshipList**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.malware_subject.**MalwareSubjectRelationship**
Bases: *maec.Entity*

class maec.package.malware_subject.**Analyses**(**args*)
Bases: mixbox.entities.EntityList

class maec.package.malware_subject.**MinorVariants**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.package.malware_subject_reference Module

Classes

class maec.package.malware_subject_reference.**MalwareSubjectReference**(*malware_subject_idref=None*)
Bases: *maec.Entity*

Version: 4.1.0.13

maec.package.object_equivalence Module

Classes

class maec.package.object_equivalence.**ObjectEquivalence**
Bases: *maec.Entity*

class maec.package.object_equivalence.**ObjectEquivalenceList**(**args*)
Bases: mixbox.entities.EntityList

Version: 4.1.0.13

maec.package.package Module

Classes

class maec.package.package.**Package**(*id=None*, *schema_version='2.1'*, *timestamp=None*)
Bases: *maec.Entity*

deduplicate_malware_subjects()
DeDuplicate all Malware_Subjects in the Package. For now, only handles Objects in Findings Bundles

static from_xml (*xml_file*)
Returns a tuple of (api_object, binding_object). Parameters: xml_file - either a filename or a stream object

MAEC Utils – Modules located in the maec.utils package

Version: 4.1.0.13

maec.utils.comparator Module

Classes

class maec.utils.comparator.**BundleComparator**
Bases: object

class maec.utils.comparator.**SimilarObjectCluster**
Bases: dict

class maec.utils.comparator.**ObjectHash**
Bases: object

class maec.utils.comparator.**ComparisonResult** (*bundle_list*, *lookup_table*)
Bases: object

Version: 4.1.0.13

maec.utils.deduplicator Module

Classes

class maec.utils.deduplicator.**BundleDeduplicator**
Bases: object

classmethod add_unique_objects (*bundle*, *all_objects*)
Add the unique Objects to the collection and perform the properties replacement.

classmethod cleanup (*bundle*)
Cleanup and remove and Objects that may be referencing the re-used Objects. Otherwise, this can create Object->Object->Object etc. references which don't make sense.

classmethod deduplicate (*bundle*)
Deduplicate the input Bundle.

classmethod find_matching_object (*obj*)
Find a matching object, if it exists.

classmethod get_object_values (*obj*, *ignoreCase=False*)
Get the values specified for an Object's properties as a set.

classmethod get_typedfield_values (*val*, *name*, *values*, *ignoreCase=False*)
Returns the value contained in a TypedField or its nested members, if applicable.

classmethod handle_duplicate_objects (*bundle*, *all_objects*)
Replace all of the duplicate Objects with references to the unique object placed in the "Re-used Objects" Collection.

classmethod handle_unique_objects (*bundle*, *all_objects*)
Add a new Object collection to the Bundle for storing the unique Objects. Add the Objects to the collection.

classmethod **map_objects**(*all_objects*)
Map the non-unique Objects to their unique (first observed) counterparts.

Version: 4.1.0.13

maec.utils.merge Module

Functions

maec.utils.merge.**merge_documents**(*input_list*, *output_file*)
Merge a list of input MAEC documents and write them to an output file

maec.utils.merge.**merge_malware_subjects**(*malware_subject_list*)
Merge a list of input Malware Subjects

maec.utils.merge.**merge_packages**(*package_list*, *namespace=None*)
Merge a list of input MAEC Packages and return a merged Package instance.

maec.utils.merge.**update_relationships**(*malware_subject_list*, *id_mappings*)
Update any existing Malware Subject relationships to account for merged Malware Subjects

maec.utils.merge.**merge_binned_malware_subjects**(*merged_malware_subject*, *binned_list*, *id_mappings_dict*)
Merge a list of input binned (related) Malware Subjects

maec.utils.merge.**create_mappings**(*mapping_dict*, *original_malware_subject_list*, *merged_malware_subject*)
Map the IDs of a list of existing Malware Subjects to the new merged Malware Subject

maec.utils.merge.**merge_findings_bundles**(*findings_bundles_list*)
Merge two or more Malware Subject Findings Bundles

maec.utils.merge.**deduplicate_vocabulary_list**(*entity_list*, *value_name='value'*)
Deduplicate a simple list of MAEC/CybOX vocabulary entries

maec.utils.merge.**merge_entities**(*entity_list*)
Merge a list of MAEC/CybOX entities

maec.utils.merge.**bin_malware_subjects**(*malware_subject_list*, *default_hash_type='md5'*)
Bin a list of Malware Subjects by hash Default = MD5

maec.utils.merge.**dict_merge**(*target*, **args*)
Merge multiple dictionaries into one

Version: 4.1.0.13

maec.utils.parser Module

Classes

class maec.utils.parser.**EntityParser**
Bases: mixbox.parser.EntityParser

MAEC Analytics – Modules located in the maec.analytics package

Version: 4.1.0.13

maec.analytics.distance Module

Classes

class maec.analytics.distance.**Distance**(*maec_entity_list*)
Bases: object

Calculates distance between two or more MAEC entities. Currently supports only Packages or Malware Subjects.

add_log(*number*, *log_list*)
Added a log'd (log-ized??) number to a list

bin_list(*numeric_value*, *numeric_list*, *n=10*)
Bin a numeric value into a bucket, based on a parent list of values. N = number of buckets to use (default = 10).

build_string_vector(*string_list*, *superset_string_list*, *ignore_case=True*)
Build a vector from an input list of strings and superset list of strings.

calculate()
Calculate the distances between the input Malware Subjects.

create_dynamic_result_vector(*dynamic_vector*)
Construct the dynamic result (matching) vector for a corresponding feature vector

create_static_result_vector(*static_vector*)
Construct the static result (matching) vector for a corresponding feature vector

create_superset_vectors()
Calculate vector supersets from the feature vectors

euclidean_distance(*vector_1*, *vector_2*)
Calculate the Euclidean distance between two input vectors

flatten_vector(*vector_entry_list*)
Generate a single, flattened vector from an input list of vectors or values.

generate_feature_vectors(*merged_subjects*)
Generate a feature vector for the binned Malware Subjects

normalize_numeric(*numeric_value*, *numeric_list*, *normalize=True*, *scale_log=True*)
Scale a numeric value, based on a parent list of values. Return the scaled/normalized form.

normalize_numeric_list(*value_list*, *numeric_list*, *normalize=True*, *scale_log=True*)
Scale a list of numeric values, based on a parent list of numeric value lists. Return the scaled/normalized form.

normalize_vectors(*vector_1*, *vector_2*)
Normalize two input vectors so that they have similar composition.

perform_calculation()
Perform the actual distance calculation. Store the results in the distances dictionary.

populate_hashes_mapping(*malware_subject_list*)
Populate and return the Malware Subject -> Hashes mapping from an input list of Malware Subjects.

preprocess_entities(*dereference=True*)
Pre-process the MAEC entities

print_distances(*file_object*, *default_label='md5'*, *delimiter=','*)
Print the distances between the Malware Subjects in delimited matrix format to a File-like object.

Try to use the MD5s of the Malware Subjects as the default label. Uses commas as the default delimiter, for CSV-like output.

class maec.analytics.distance.**StaticFeatureVector**(*malware_subject*, *deduplicator*)

Bases: object

Generate a feature vector for a Malware Subject based on its static features

create_object_vector(*object*, *static_feature_dict*, *callback_function=None*)
Create a vector from a single Object

create_static_vectors(*malware_subject*)
Create a vector of static features for an input Malware Subject

extract_features(*malware_subject*)
Extract the static features from the Malware Subject

get_unique_features()
Calculates the unique set of static features for the Malware Subject

class maec.analytics.distance.**DynamicFeatureVector**(*malware_subject*, *deduplicator*, *ignored_object_properties*, *ignored_actions*)

Bases: object

Generate a feature vector for a Malware Subject based on its dynamic features

create_action_vector(*action*)
Create a vector from a single Action

create_dynamic_vectors(*malware_subject*)
Create a vector of unique action/object pairs for an input Malware Subject

extract_features(*malware_subject*)
Extract the dynamic features from the Malware Subject

get_unique_features()
Calculates the unique set of dynamic features for the Malware Subject

prune_dynamic_features(*min_length=2*)
Prune the dynamic features based on ignored Object properties/Actions

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

m

A

B

C

D

E

F

G

H

I

L

M

N

O

P

S

T

U