l2tScaffolder Documentation

Log2timeline Developers

May 12, 2020

Contents

The l2t_scaffolder is a tool developed to speed up l2t development by automating the generation of plugins and parsers in various tools, such as Plaso and Timesketch.

At the moment there is no documentation to speak of, but it will be provided shortly.

The project's code is available from https://github.com/log2timeline/l2tscaffolder

Contents:

CHAPTER 1

User documentation

1.1 Installing the Tool

There are two ways to install the tool:

1. Use pip
2. Source from github

Let's cover both ways. But the first recommended step is to setup a virtualenv environment.

Follow the instructions here or a quick method:

```
$ virtualenv -p /usr/bin/python3 scaffolder
$ source scaffolder/bin/activate
```

Once the virtual environment is setup you can move on to the next step, either using pip or source installation.

1.1.1 Pip Install

To install the latest release of the scaffolder, use:

```
$ pip3 install --upgrade l2tscaffolder
```

1.1.2 Install From Sources

First fetch the latest source code from github:

```
$ git clone https://github.com/log2timeline/l2tscaffolder.git
```

Then install dependencies and compile and install the tool:

```
$ cd l2tscaffolder
$ pip3 install -r requirements.txt
$ python3 setup.py build && python3 setup.py install
```

1.2 Use the tool

1.2.1 Installation

The first step is to install the tool. Follow the instructions here

1.2.2 Preparation

Once the tool is installed you'll need to first fetch the source of whatever project you are adding code to, eg. Plaso. The way log2timeline development is done is that you need to first create a fork of the main project into your own account, clone your fork and then work from that one. Here is an example of fetching and syncing your personal fork:

```
$ git clone https://github.com/kiddinn/plaso.git
$ cd plaso
$ git remote add upstream https://github.com/log2timeline/plaso.git
$ git pull --rebase upstream master
$ git push
```

Once this is ready you can start using the l2t_scaffolder tool.

1.2.3 Using the Tool

The tool will guide you through its use, the parameters are fairly simple:

```
l2t_scaffolder.py [DEFINITION]
```

Where definition is an optional parameter of the name of the project, eg. `plaso`, `timesketch`, etc.

The simplest way to run the tool is to run it without any parameters and then follow the questions asked.

```
$ l2t_scaffolder.py
   == Starting the scaffolder ==
Gathering all required information.

Available definitions:
  [0] plaso
  [1] timesketch
Definition choice: 0
plaso chosen.

Path to the project root: plaso
Path [plaso] set as the project path.

Name of the module to be generated. This can be something like "foobar sqlite"
or "event analytics".

This will be used for class name generation and file name prefixes.
```

(continues on next page)

(continued from previous page)

```
Module Name:
...
```

After that it is a simple manner of following the instructions given by the tool.

Some notes:

- "*Name of the module*": this is used to create both the name of the class as well as filenames of the generated files, so if you choose something like: "New Awesome Parser" you'll end up with a parser/plugin file with the name of new_awesome_parser.py and a class name on the lines of NewAwesomeParserParser (depending on the scaffolder some text may be appended to the class name).
- Each scaffolder will determine what questions need to be asked in order to successfully generate files, some may ask more than others, eg. the SQLite plugin will ask for SQL commands, and names of functions. That will be used to generate the skeleton of the code.
- Once the tool has collected all answers to questions it will generate the required files, what it will do is:
  - Create a feature branch inside the git repository
  - Generate all the necessary files
  - Add those files to the git client

Once the tool completes it's run, you can go to the git repo of the project you just generated the files and start completing them. The tool uses a template, often filled with TODOs or missing parts that need to be completed before the plugin/parser is ready for use. However it should get you started by generating all the necessary files as well as filling out the boiler plate code that is often needed.

1.3 Setting up L2t scaffolder in a virtualenv

For development purposes, l2t_scaffolder can be installed using virtualenv (preferred method).

1.3.1 Fedora Core

Install virtualenv

To install virtualenv on Fedora Core (or equivalent) run:

```
$ sudo dnf install python3-virtualenv
```

Installing build dependencies

TODO add more text

1.3.2 Ubuntu

Installing virtualenv

To install virtualenv on Ubuntu (or equivalent) run:

```
$ sudo apt-get install python-virtualenv python3-virtualenv
```

Installing build dependencies

TODO add more text

```
$ sudo apt-get install libyaml-dev liblzma-dev
```

1.3.3 Setting up l2t_scaffolder in virtualenv

To create a virtualenv:

```
virtualenv -p PATH_TO_PYTHON3 scaffolderoenv
```

eg:

```
$ virtualenv -p /usr/bin/python3 scaffolderoenv
```

To activate the virtualenv:

```
$ source ./scaffolderenv/bin/activate
```

Note that using pip outside virtualenv is not recommended since it ignores your systems package manager.

Make sure that pip is up-to-date:

```
$ pip3 install --upgrade pip
```

1.3.4 Deactivate Virtualenv

To deactivate the virtualenv run:

```
$ deactivate
```

1.3.5 Configuring Git Client

Before submitting code to the project, make sure that you have created a fork of the l2tscaffolder project, and check out your personal fork:

```
$ git clone https://github.com/USERNAME/l2tscaffolder.git
```

Add the upstream repo:

```
$ git remote add upstream https://github.com/log2timeline/l2tscaffolder.git
```

And then you can create a feature branch to work on.

```
$ git checkout -b my_feature
```

CHAPTER 2

l2tscaffolder package

2.1 Subpackages

2.1.1 l2tscaffolder.definitions package

Submodules

l2tscaffolder.definitions.interface module

Interface defining how a project class looks like.

class l2tscaffolder.definitions.interface.**ScaffolderDefinition**
Bases: object

Scaffolder definition interface.

NAME = 'undefined'

ValidatePath(*root_path: str*) → bool
Validates the path to the root directory of the project.

Parameters **root_path** (*str*) – the path to the root of the project directory.

Returns whether the given path is the correct root path of the project.

Return type bool

l2tscaffolder.definitions.manager module

The definition manager.

class l2tscaffolder.definitions.manager.**DefinitionManager**
Bases: object

The definition manager.

classmethod DeregisterDefinition(*definition_class: Type[l2tscaffolder.definitions.interface.ScaffolderDefinition]*)
Deregisters a definition class.

Definition classes are identified by their NAME attribute.

Parameters **definition_class** (*type*) – definition class (subclass of ScaffolderDefinition).

Raises KeyError – if definition class is not set for the corresponding name.

classmethod GetDefinitionByName(*name: str*) → Type[l2tscaffolder.definitions.interface.ScaffolderDefinition]
Returns a definition class based on registered name.

Parameters **name** (*str*) – name of the definition.

Returns

definition class or None if name is not registered.

Return type *interface.ScaffolderDefinition*

classmethod GetDefinitionNames() → Iterator[str]
Yields all names of registered definition classes.

Yields *str* – definition names.

classmethod GetDefinitionObjects() → Iterator[l2tscaffolder.definitions.interface.ScaffolderDefinition]
Yields instances of each registered definition class.

Yields *ScaffolderDefinition* – definition object.

classmethod RegisterDefinition(*definition_class: Type[l2tscaffolder.definitions.interface.ScaffolderDefinition]*)
Registers a definition class.

Definition classes are identified by their NAME attribute.

Parameters **definition_class** (ScaffolderDefinition) – definition class.

Raises KeyError – if definition class is already set for the corresponding name.

l2tscaffolder.definitions.plaso module

The plaso definition class.

class l2tscaffolder.definitions.plaso.**PlasoProject**
Bases: *l2tscaffolder.definitions.interface.ScaffolderDefinition*

Plaso project definition.

NAME = 'plaso'

ValidatePath(*root_path: str*) → bool
Validates the path to a Plaso development tree.

Parameters **root_path** (*str*) – the path to the root of the project directory.

Returns

whether the given path is the correct root path to a Plaso development tree.

Return type bool

l2tscaffolder.definitions.timesketch module

The Timesketch definition class.

class l2tscaffolder.definitions.timesketch.**TimesketchProject**
Bases: *l2tscaffolder.definitions.interface.ScaffolderDefinition*

Timesketch project definition.

NAME = 'timesketch'

ValidatePath(*root_path: str*) → bool
Validates the path to a Timesketch development tree.

Parameters **root_path** (*str*) – the path to the root of the project directory.

Returns

whether the given path is the correct root path to a Timesketch development tree.

Return type bool

l2tscaffolder.definitions.turbinia module

The Turbinia definition class.

class l2tscaffolder.definitions.turbinia.**TurbiniaProject**
Bases: *l2tscaffolder.definitions.interface.ScaffolderDefinition*

Turbinia project definition.

NAME = 'turbinia'

ValidatePath(*root_path: str*) → bool
Validates the path to a Timesketch development tree.

Parameters **root_path** (*str*) – the path to the root of the project directory.

Returns

whether the given path is the correct root path to a Timesketch development tree.

Return type bool

Module contents

This file imports Python modules that registers definitions.

2.1.2 l2tscaffolder.frontend package

Submodules

l2tscaffolder.frontend.cli_output_handler module

The output file handler for click

class l2tscaffolder.frontend.cli_output_handler.**OutputHandlerClick**
Bases: *l2tscaffolder.frontend.output_handler.BaseOutputHandler*

Output handler for click.

Confirm(*text: str*, *default=True*, *abort=True*)
Returns a bool from a yes/no question presented to the end user.

Parameters

- **text** (*str*) – the question presented to the end user.
- **default** (*bool*) – the default for the confirmation answer. If True the default is Y(es), if False the default is N(o)
- **abort** (*bool*) – if the program should abort if the user answer to the confirm prompt is no. The default is an abort.

Returns False if the user entered no, True if the user entered yes

Return type bool

PrintError(*text: str*)
Presents an error message.

Parameters **text** (*str*) – the error message to present.

PrintInfo(*text: str*)
Presents the user with an informational text.

Parameters **text** (*str*) – the text to present.

PrintNewLine()
Adds a new or blank line to the output.

PrintOutput(*text: str*)
Presents the user with output from the tool.

Parameters **text** (*str*) – the text to present the user with.

PromptError(*text: str*) → str
Presents the user with an error message and return back the answer.

Parameters **text** (*str*) – the text to prompt

Returns the user input

Return type str

PromptInfo(*text: str*) → str
Presents the user with a message prompt and return back the answer.

Parameters **text** (*str*) – the text to prompt

Returns the user input

Return type str

PromptInfoWithDefault(*text: str*, *input_type: type*, *default: object*) → object
Presents the user with a prompt with a default return value and a type.

The prompt can have a default value to be chosen as well as a defined type of the returned data.

Parameters

- **text** (*str*) – the text to prompt
- **input_type** (*type*) – the type of the input
- **default** (*object*) – the default value

Returns the user input, using the supplied input type.

Return type object

l2tscaffolder.frontend.frontend module

The scaffolder frontend.

class l2tscaffolder.frontend.frontend.**ScaffolderFrontend**(*output_handler: l2tscaffolder.frontend.output_handler.BaseOutput*

Bases: object

A frontend implementation for the scaffolder project.

CreateGitFeatureBranch(*project_path: str*, *module_name: str*)
Creates a feature branch inside the git project.

Creates a feature branch inside the git project path to store all the generated files in.

Parameters

- **project_path** (*str*) – path to the git project folder.
- **module_name** (*str*) – name of the output module.

GatherScaffolderAnswers(*scaffolder*, *scaffolder_engine*)
Asks all questions that scaffolder requires and store the results in it.

Parameters

- **scaffolder** (*scaffolder_interface.Scaffolder*) – the scaffolder that stores all required questions and stores all results as well.
- **scaffolder_engine** (*scaffolder_engine.ScaffolderEngine*) – the scaffolder engine object, needed to store answers from questions asked.

Raises UnableToConfigure – if the answer causes the scaffolder not to be configured properly.

GetDefinition(*definition_string: str*) → l2tscaffolder.definitions.interface.ScaffolderDefinition
Returns the definition object as chosen by the user.

Parameters **definition_string** (*str*) – definition name, read from user input.

Returns the chosen definition object.

Return type definition_interface.ScaffolderDefinition

GetModuleName() → str
Returns the module name as chosen by the user.

GetProjectPath(*definition: l2tscaffolder.definitions.interface.ScaffolderDefinition*) → str
Returns the path to the project's root folder as chosen by the user.

Parameters **definition** (*definition_interface.ScaffolderDefinition*) – the chosen definition. Used to validate the project path.

Returns the path to the project's root folder.

Return type str

Raises errors.WrongCliInput – when no valid project path has been provided.

GetScaffolder(*definition: l2tscaffolder.definitions.interface.ScaffolderDefinition*) → l2tscaffolder.scaffolders.interface.Scaffolder
Returns the scaffolder as chosen by the user.

Parameters **definition** (*definition_interface.ScaffolderDefinition*) – the chosen definition. Used to determine available scaffolders.

Returns the chosen scaffolder object.

Return type scaffolder_interface.ScaffolderCli

Start(*definition_value*)
Start the CLI.

Parameters **definition_value** (*str*) – the definition string chosen by UI.

l2tscaffolder.frontend.output_handler module

The output file handler.

This file defines the interface of how an output handler should operate. An output handler is used as a UI element, for two things: 1. Relay information back to the user. 2. Gather input from an end user and presenting it back to the tool.

class l2tscaffolder.frontend.output_handler.**BaseOutputHandler**
Bases: object

Interface for the output handler.

Confirm(*text: str*, *default=True*, *abort=True*)
Returns a bool from a yes/no question presented to the end user.

Parameters

- **text** (*str*) – the question presented to the end user.
- **default** (*bool*) – the default for the confirmation answer. If True the default is Y(es), if False the default is N(o)
- **abort** (*bool*) – if the program should abort if the user answer to the confirm prompt is no. The default is an abort.

Returns False if the user entered no, True if the user entered yes

Return type bool

PrintError(*text: str*)
Presents an error message.

Parameters **text** (*str*) – the error message to present.

PrintInfo(*text: str*)
Presents the user with an informational text.

Parameters **text** (*str*) – the text to present.

PrintNewLine()
Adds a new or blank line to the output.

PrintOutput(*text: str*)
Presents the user with output from the tool.

Parameters **text** (*str*) – the text to present the user with.

PromptError(*text: str*) → str
Presents the user with an error message prompt and returns the answer.

Parameters **text** (*str*) – the text to prompt

Returns the user input.

Return type str

PromptInfo(*text: str*) → str
Presents the user with a message prompt and return back the answer.

Parameters **text** (*str*) – the text to prompt

Returns the user input.

Return type str

PromptInfoWithDefault(*text: str*, *input_type: type*, *default: object*) → str
Presents the user with a prompt with a default return value and a type.

The prompt can have a default value to be chosen as well as a defined type of the returned data.

Parameters

- **text** (*str*) – the text to prompt
- **input_type** (*type*) – the type of the input
- **default** (*object*) – the default value

Returns the user input, using the supplied input type.

Return type object

Module contents

2.1.3 l2tscaffolder.helpers package

Submodules

l2tscaffolder.helpers.cli module

Helper for command line functions.

class l2tscaffolder.helpers.cli.**CLIHelper**(*mock_responses=None*)
Bases: object

Command line interface (CLI) helper.

mock_responses
mappings of commands to responses.

Type dict[str, str]

preferred_encoding
preferred encoding of output.

Type str

RunCommand(*command: str*)
Runs a command.

Parameters **command** (*str*) – command to run.

Returns

exit code, output that was written to stdout and stderr.

Return type tuple[int, str, str]

Raises AttributeError – if the command is not recognized.

l2tscaffolder.helpers.git module

Git helper for the scaffolder project.

This file provides a class to assist with git operations.

class l2tscaffolder.helpers.git.**GitHelper**(*project_path: str*)
Bases: *l2tscaffolder.helpers.cli.CLIHelper*

Helper class for git operations.

project_path
path to the git project folder.

AddFileToTrack(*file_path: str*)
Add a file to those that are tracked by the git repo.

Parameters **file_path** (*str*) – path to the file to be added to tracked files by this git repo.

Raises errors.UnableToConfigure – when the tool is not able to add newly added files to the git repo.

CreateBranch(*branch: str*) → int
Creates a git branch and returns the exit code of the command.

Parameters **branch** (*str*) – the name of the git branch.

Returns the exit code from the git command.

Return type int

GenerateBranchName(*module_name: str*) → str
Generates a git branch name.

Parameters **module_name** (*str*) – module name to generate a git branch name from.

Returns git branch name.

Return type str

GetActiveBranch() → str
Determines the active branch of the git project.

Returns the active branch of the git project.

Return type str

Raises errors.UnableToConfigure – when the tool is not able to get the active branch of the git project.

HasBranch(*branch_name: str*) → bool
Tests for the existence of a specific branch.

Parameters **branch_name** (*str*) – the name of the branch to test for.

Returns True if the branch exists.

Return type bool

RunCommand(*command: str*) → Tuple[int, str, str]
Runs a command.

Parameters **command** (*str*) – command to run.

Returns

exit code, output that was written to stdout and stderr.

Return type tuple[int, str, str]

SwitchToBranch(*branch: str*) → int
Switches the git branch and returns the exit code of the command.

Parameters **branch** (*str*) – the name of the git branch.

Returns the exit code from the git command.

Return type int

Module contents

2.1.4 l2tscaffolder.lib package

Submodules

l2tscaffolder.lib.code_formatter module

Formatter for generated code.

class l2tscaffolder.lib.code_formatter.**CodeFormatter**(*yapf_path: str*)
Bases: object

Formats code in files.

Format(*code: str*) → str
Formats the code.

Parameters **code** (*str*) – code to format

Returns the formatted code

Return type str

l2tscaffolder.lib.definitions module

The format specification classes.

l2tscaffolder.lib.engine module

The scaffolder engine.

class l2tscaffolder.lib.engine.**ScaffolderEngine**
Bases: object

The engine, responsible for file handling and setting up scaffolders.

GenerateFiles() → Iterator[str]
Generates needed files.

Raises errors.EngineNotConfigured – when not all attributes have been configured.

Yields *str* – the full path to a file that was generated and written to disk.

SetModuleName(*module_name: str*)
Sets the module name as chosen by the user.

Parameters **module_name** (*str*) – name of the module to be generated by the scaffolder.

SetProjectRootPath(*root_path: str*)
Sets the path to the root of the project tree.

Raises errors.NoValidDefinition – when root path is not identified as a valid definition path.

SetScaffolder(*scaffolder: l2tscaffolder.scaffolders.interface.Scaffolder*)
Stores and initializes the scaffolder object in the engine.

Parameters **scaffolder** (*scaffolder_interface.Scaffolder*) – the scaffolder class that the engine will use to generate files.

StoreScaffolderAttribute(*name: str, value: object, value_type: Type[CT_co]*)
Stores an attribute read from the CLI.

Parameters

- **name** (*str*) – the attribute name.
- **value** (*object*) – the attribute value.
- **value_type** (*type*) – the attribute type.

Raises

- KeyError – if the attribute name is already defined.
- ScaffolderNotConfigured – if the scaffolder has not yet been set.
- ValueError – if the value is not of the correct type.

l2tscaffolder.lib.errors module

This file contains the error classes.

exception l2tscaffolder.lib.errors.**EngineNotConfigured**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when the scaffolder engine has not been configured correctly.

exception l2tscaffolder.lib.errors.**Error**
Bases: Exception

Base error class.

exception l2tscaffolder.lib.errors.**FileHandlingError**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when the file handler is unable to do file operation.

exception l2tscaffolder.lib.errors.**NoValidDefinition**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when no valid project definition has been identified.

exception l2tscaffolder.lib.errors.**ScaffolderNotConfigured**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when the scaffolder has not been configured correctly.

exception l2tscaffolder.lib.errors.**UnableToConfigure**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when the scaffolder tool has issues with configuration.

exception l2tscaffolder.lib.errors.**WrongCliInput**
Bases: *l2tscaffolder.lib.errors.Error*

Raised when wrong input is entered into the CLI.

l2tscaffolder.lib.file_handler module

The file handler.

class l2tscaffolder.lib.file_handler.**FileHandler**
Bases: object

Handles the creation of files.

AddContent(*source: str*, *content: str*) → str
Adds content to a file and create file if non existing.

Parameters

- **source** (*str*) – path of the file to edit.
- **content** (*str*) – content to append to the file.

Returns path of the edited file.

Return type str

AddImportToInit(*path: str*, *entry: str*)
Adds an import into an init file in the correct order.

Parameters

- **path** (*str*) – path to the __init__ file.
- **entry** (*str*) – the import statement.

CopyFile(*source: str*, *destination: str*) → str
Copies a file.

Parameters

- **source** (*str*) – path of the file to copy
- **destination** (*str*) – path to copy the file to.

Returns the path of the copied file

Return type str

Raises errors.FileHandlingError – when file copy operation fails.

CreateFile(*directory_path: str*, *file_name: str*, *filename_extension: str*) → str
Creates a empty file.

Parameters

- **directory_path** (*str*) – path to the directory the file should be created in.
- **file_name** (*str*) – name of the new file.
- **filename_extension** (*str*) – extension of the new file.

Returns path of the created file

Return type str

CreateFileFromPath (*file_path: str*) → str
Creates a empty file.

Parameters **file_path** (*str*) – path to the file.

Returns the path of the created file

Return type str

classmethod CreateFilePath (*path: str*, *name: str*, *extension: str*) → str
Creates the file path from the directory path, filename and suffix.

Parameters

- **path** (*str*) – path to the file directory.
- **name** (*str*) – filename.
- **extension** (*str*) – file extension.

Returns the path to the file.

Return type str

CreateFolderForFilePathIfNotExist (*file_path: str*) → str
Creates folders for the given file if it does not exist.

Parameters **file_path** (*str*) – path to the file

Returns directory path of the created directory

Return type str

CreateOrModifyFileWithContent (*source: str*, *content: str*)
Adds content to a file and create the file and path if non existing.

Parameters

- **source** (*str*) – path of the file to edit.
- **content** (*str*) – content to append to the file.

Returns path of the edited file.

Return type str

l2tscaffolder.lib.mapping_helper module

Helper methods for mapping.

class l2tscaffolder.lib.mapping_helper.**MappingHelper** (*template_path: str = ''*, *formatter_path: str = ''*)
Bases: object

Mapping helper for scaffolders.

GenerateClassName (*scaffolder_name: str*) → str
Generates a class name from the scaffolder name for file generation.

Parameters **scaffolder_name** (*str*) – name of the scaffolder

Returns name of the class

Return type str

RenderTemplate (*template_filename: str*, *context: dict*) → str
Renders the template with the context to return a string.

Parameters

- **template_filename** (*str*) – the name of the template
- **context** (*dict*) – the context of the template as a dictionary

Returns the rendered template as a string

Return type str

Module contents

Library code for l2tscaffolder.

2.1.5 l2tscaffolder.scaffolders package

Submodules

l2tscaffolder.scaffolders.interface module

The scaffolder interface classes.

class l2tscaffolder.scaffolders.interface.**BaseQuestion**(*attribute: str*, *prompt: str*)

Bases: object

Scaffolder question.

attribute
the name of the attribute the question prompts for.

Type str

prompt
help string that is displayed before the question is asked.

Type str

TYPE = None

ValidateAnswer(*answer: object*)
Validate an answer to a question.

Parameters **answer** (*object*) – the answer to the question asked.

Raises errors.UnableToConfigure – if the answer is invalid.

class l2tscaffolder.scaffolders.interface.**DictQuestion**(*attribute*, *prompt*, *key_prompt*, *value_prompt*)

Bases: *l2tscaffolder.scaffolders.interface.BaseQuestion*

Scaffolder dict question.

attribute
the name of the attribute the question prompts for.

Type str

prompt
help string that is displayed before the question is asked.

Type str

key_prompt
the help string that is displayed before asking for each key.

Type str

value_prompt
the help string that is displayed before asking for each value in the dict.

Type str

TYPE
alias of builtins.dict

class l2tscaffolder.scaffolders.interface.**IntQuestion**(*attribute: str*, *prompt: str*)
Bases: *l2tscaffolder.scaffolders.interface.BaseQuestion*

Scaffolder integer question.

TYPE
alias of builtins.int

class l2tscaffolder.scaffolders.interface.**ListQuestion**(*attribute: str*, *prompt: str*)
Bases: *l2tscaffolder.scaffolders.interface.BaseQuestion*

Scaffolder list question.

TYPE
alias of builtins.list

class l2tscaffolder.scaffolders.interface.**Scaffolder**
Bases: object

The scaffolder interface.

DESCRIPTION = ''

GenerateFiles() → Iterator[Tuple[str, str]]
Generates files this scaffolder provides.

Yields *list* – file name and content of the file to be written to disk.

GetFilesToCopy() → Iterator[Tuple[str, str]]
Return a list of files that need to be copied.

If not overwritten this will return an emtpy iterator.

Yields *tuple (str, str)* – file name of source and destination.

GetInitFileChanges() → Iterator[Tuple[str, str]]
Generate a list of init files that need changing and the changes to them.

Yields *tuple (str, str)* – path to the init file and the entry to add to it.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

GetQuestions() → List[l2tscaffolder.scaffolders.interface.BaseQuestion]
Returns scaffolder questions.

Returns questions to prompt the user with.

Return type list[*BaseQuestion*]

NAME = 'base_parser'

PROJECT = 'plaso'

QUESTIONS = []

RaiseIfNotReady()
Checks to see if all attributes are set to start generating files.

By default this function only checks to see if all attributes defined in questions and Jinja2 context have values and are not empty.

Raises ScaffolderNotConfigured – if the scaffolder is not fully configured.

SetAttribute(*name: str*, *value: object*, *value_type: type*)
Stores an attribute read from the CLI.

Parameters

- **name** (*str*) – the attribute name.
- **value** (*object*) – the attribute value.
- **value_type** (*type*) – the attribute type.

Raises

- ValueError – if the value is not of the correct type.
- KeyError – If the attribute is not configured for this scaffolder.

SetOutputName(*output_name: str*)
Sets the name of the output module.

This is the name of the generated output module this scaffolder implements.

Parameters **output_name** (*str*) – the name of the output that the scaffolder generates, whether that is an output module, plugin, parser, analyzer or something else.

class l2tscaffolder.scaffolders.interface.**StringQuestion**(*attribute: str*, *prompt: str*)
Bases: *l2tscaffolder.scaffolders.interface.BaseQuestion*

Scaffolder string question.

TYPE
alias of builtins.str

l2tscaffolder.scaffolders.manager module

The scaffolder manager.

class l2tscaffolder.scaffolders.manager.**ScaffolderManager**
Bases: object

The scaffolder manager.

classmethod DeregisterScaffolder(*scaffolder_class: Type[l2tscaffolder.scaffolders.interface.Scaffolder]*)
Deregisters a scaffolder class.

The scaffolder classes are identified based on their lower case name.

Parameters **scaffolder_class** (*type*) – scaffolder class (subclass of Scaffolder).

Raises KeyError – if scaffolder class is not set for the corresponding name.

classmethod GetScaffolderClasses() → Iterator[Type[l2tscaffolder.scaffolders.interface.Scaffolder]]
Generates a list of all registered scaffolder classes.

classmethod GetScaffolderInformation() → Iterator[Tuple[str, str]]
Retrieves the scaffolder information.

Yields *tuple[str, str]* – pairs of scaffolder names and descriptions.

classmethod GetScaffolderNames() → Iterator[str]
Retrieves the scaffolder names.

Yields *str* – scaffolder names.

classmethod GetScaffolderObjectByName(*scaffolder_name*) → Optional[l2tscaffolder.scaffolders.interface.Scaffolder]
Retrieves a specific scaffolder object by its name.

Parameters **scaffolder_name** (*str*) – name of the scaffolder.

Returns scaffolder object or None.

Return type *Scaffolder*

classmethod GetScaffolderObjects() → Dict[str, l2tscaffolder.scaffolders.interface.Scaffolder]
Retrieves the scaffolder objects.

Returns scaffolders per name.

Return type dict[str, *Scaffolder*]

classmethod GetScaffolderQuestionByName(*scaffolder_name: str*) → List[l2tscaffolder.scaffolders.interface.BaseQuestion]
Retrieve a list of questions asked by a scaffolder based on name.

Parameters **scaffolder_name** (*str*) – name of the scaffolder.

Returns

a list with all the questions needed to setup the scaffolder. If scaffolder_name is not registered an empty list will be returned.

Return type list

classmethod GetScaffolderQuestions() → List[l2tscaffolder.scaffolders.interface.BaseQuestion]
Retrieves all the questions asked by scaffolders.

Returns questions asked by all scaffolders.

Return type list[*interface.BaseQuestion*]

classmethod GetScaffolders() → Iterator[Tuple[str, Type[l2tscaffolder.scaffolders.interface.Scaffolder]]]
Retrieves the registered scaffolders.

Retrieves a dictionary of all registered scaffolders.

Yields *tuple* – contains:

- str: name of the scaffolder:
- type: scaffolder class (subclass of Scaffolder).

classmethod RegisterScaffolder(*scaffolder_class: Type[l2tscaffolder.scaffolders.interface.Scaffolder]*)
Registers a scaffolder class.

The scaffolder classes are identified based on their lower case name.

Parameters **scaffolder_class** (*type*) – scaffolder class (subclass of Scaffolder).

Raises KeyError – if scaffolder class is already set for the corresponding name.

classmethod RegisterScaffolders(*scaffolder_classes: List[Type[l2tscaffolder.scaffolders.interface.Scaffolder]]*)
Registers scaffolder classes.

The scaffolder classes are identified based on their lower case name.

Parameters **scaffolder_classes** (*list[type]*) – scaffolders classes (subclasses of Scaffolder).

Raises KeyError – if scaffolder class is already set for the corresponding name.

l2tscaffolder.scaffolders.plaso module

Plaso scaffolder that generates plaso parser and plugins.

class l2tscaffolder.scaffolders.plaso.**PlasoBaseScaffolder**
Bases: *l2tscaffolder.scaffolders.interface.Scaffolder*

The plaso base scaffolder interface.

class_name
class name of the plaso parser or plugin to be generated.

Type str

test_file
name of the file used for testing the parser or plugin.

Type str

test_file_path
path to the test file.

Type str

DESCRIPTION = 'This is a scaffolder for plaso parsers and/or plugins'

GenerateFiles() → Iterator[Tuple[str, str]]
Generates all the files required for a plaso parser or a plugin.

Yields *list[tuple]* –

containing: str: file name. str: file content.

GetFilesToCopy() → Iterator[Tuple[str, str]]
Return a list of files that need to be copied.

Raises IOError – when the test file does not exist.

Yields *tuple* –

containing: str: file name of source. str: file name of destination.

GetInitFileChanges() → Iterator[Tuple[str, str]]
Generate a list of init files that need changing and the changes to them.

Yields *Tuple[str, str]* – path to the init file and the entry to add to it.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

GetQuestions() → List[l2tscaffolder.scaffolders.interface.BaseQuestion]
Returns scaffolder questions as well as adding plaso related ones.

Returns questions to prompt the user with.

Return type list[*interface.BaseQuestion*]

NAME = 'plaso_base'

PROJECT = 'plaso'

QUESTIONS = []

RaiseIfNotReady()
Checks to see if all attributes are set to start generating files.

Raises ScaffolderNotConfigured – if the scaffolder is not fully configured.

TEMPLATE_FORMATTER_FILE = 'generic__plaso_formatter.jinja2'

TEMPLATE_FORMATTER_TEST = 'generic_plaso_formatter_test.jinja2'

TEMPLATE_PARSER_FILE = 'generic_plaso_parser.jinja2'

TEMPLATE_PARSER_TEST = 'generic_plaso_parser_test.jinja2'

class l2tscaffolder.scaffolders.plaso.**PlasoParserScaffolder**
Bases: *l2tscaffolder.scaffolders.plaso.PlasoBaseScaffolder*

Scaffolder for generating plaso parsers.

parser_name
name of the parser to be generated.

Type str

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

class l2tscaffolder.scaffolders.plaso.**PlasoPluginScaffolder**
Bases: *l2tscaffolder.scaffolders.plaso.PlasoBaseScaffolder*

Scaffolder for generating plaso plugins.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

class l2tscaffolder.scaffolders.plaso.**TestFileQuestion**(*attribute: str*, *prompt: str*)
Bases: *l2tscaffolder.scaffolders.interface.StringQuestion*

Test file question.

ValidateAnswer(*answer: str*)
Validates the answer to the test file question.

Parameters **answer** (*str*) – path to a test file.

Raises errors.UnableToConfigure – if the answer is invalid.

l2tscaffolder.scaffolders.plaso_sqlite module

The scaffolder interface classes.

class l2tscaffolder.scaffolders.plaso_sqlite.**PlasoSQLiteScaffolder**
Bases: *l2tscaffolder.scaffolders.plaso.PlasoPluginScaffolder*

The plaso SQLite plugin scaffolder.

database_name
name of the test SQLite database for the plugin.

Type str

database_schema
a dict containing all table names (keys) and the SQL statement used to create the table (value), derived from the test database.

Type dict

data_types
a dict containing all the data types generated for the parser, the key is the name for each SQL statement run against the database and the value is the data type used for each generated event resulting from that SQL statement.

Type dict

queries
a dict containing query name and SQL statements or queries run against the database.

Type dict

query_columns
for each SQL statement run against the database, with the key being query name and value being a list of all SQL column names that are returned for each query.

Type dict

required_tables
a list of all required tables needed for the plugin to parse this particular database.

Type list

timestamp_columns
a dict containing a list of all columns with timestamp values, with query names as the key.

Type dict

DESCRIPTION = 'Provides a scaffolder to generate a plaso SQLite plugin.'

GenerateFiles() → Iterator[Tuple[str, str]]
Generates files required for the SQLite plugin.

Yields *tuple* – file name and content of the file to be written to disk.

Raises errors.UnableToConfigure – if it is not possible to generate the files.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

NAME = 'sqlite'

QUESTIONS = [<l2tscaffolder.scaffolders.plaso_sqlite.SQLQuestion object>, <l2tscaffold

SCHEMA_QUERY = 'SELECT tbl_name, sql FROM sqlite_master WHERE type = "table" AND tbl_n

TEMPLATE_FORMATTER_FILE = 'sqlite_plugin_formatter.jinja2'

TEMPLATE_FORMATTER_TEST = 'sqlite_plugin_formatter_test.jinja2'

TEMPLATE_PARSER_FILE = 'sqlite_plugin.jinja2'

TEMPLATE_PARSER_TEST = 'sqlite_plugin_test.jinja2'

class l2tscaffolder.scaffolders.plaso_sqlite.**SQLQuestion**(*attribute*, *prompt*, *key_prompt*, *value_prompt*)
Bases: *l2tscaffolder.scaffolders.interface.DictQuestion*

SQL Query question.

ValidateAnswer(*answer: dict*)
Validates the answer to the SQL query question.

The answer should be a dict that has query names as key values and valid SQLite commands as values. This function attempts to verify that the SQL commands do not have syntax errors in them by attempting to run it against an empty SQLite database stored in memory.

The function also makes sure the key value confirms to the style guide of plaso, to be in the form of CamelCase, eg. BookmarkRow.

Parameters **answer** (*dict*) – the answer to the question asked.

Raises errors.UnableToConfigure – if the answer is invalid.

l2tscaffolder.scaffolders.timesketch module

Timesketch scaffolder that generates analyzer plugins.

class l2tscaffolder.scaffolders.timesketch.**TimesketchBaseScaffolder**
Bases: *l2tscaffolder.scaffolders.interface.Scaffolder*

The Timesketch base scaffolder interface.

class_name
class name of the Timesketch analyzer to be generated.

Type str

DESCRIPTION = 'This is a scaffolder for Timesketch analyzers'

GenerateFiles() → Iterator[Tuple[str, str]]
Generates all the files required for a Timesketch analyzer plugin.

Yields *list[tuple]* –

containing: str: file name. str: file content.

GetInitFileChanges() → Iterator[Tuple[str, str]]
Generate a list of init files that need changing and the changes to them.

Yields *Tuple[str, str]* – path to the init file and the entry to add to it.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

NAME = 'timesketch_base'

PROJECT = 'timesketch'

QUESTIONS = []

TEMPLATE_PLUGIN_FILE = ''

TEMPLATE_PLUGIN_TEST = ''

l2tscaffolder.scaffolders.timesketch_index module

Timesketch index analyzer scaffolder.

class l2tscaffolder.scaffolders.timesketch_index.**TimesketchIndexScaffolder**
Bases: *l2tscaffolder.scaffolders.timesketch.TimesketchBaseScaffolder*

The Timesketch index analyzer plugin scaffolder.

DESCRIPTION = 'Provides a scaffolder to generate a Timesketch index analyzer plugin.'

NAME = 'index_analyzer'

TEMPLATE_PLUGIN_FILE = 'ts_index_analyzer.jinja2'

TEMPLATE_PLUGIN_TEST = 'ts_index_analyzer_test.jinja2'

l2tscaffolder.scaffolders.timesketch_sketch module

Timesketch sketch analyzer scaffolder.

class l2tscaffolder.scaffolders.timesketch_sketch.**TimesketchSketchScaffolder**
Bases: *l2tscaffolder.scaffolders.timesketch.TimesketchBaseScaffolder*

The Timesketch sketch analyzer plugin scaffolder.

DESCRIPTION = 'Provides a scaffolder to generate a Timesketch sketch analyzer plugin.'

NAME = 'sketch_analyzer'

TEMPLATE_PLUGIN_FILE = 'ts_sketch_analyzer.jinja2'

TEMPLATE_PLUGIN_TEST = 'ts_sketch_analyzer_test.jinja2'

l2tscaffolder.scaffolders.turbinia module

Turbinia component scaffolder.

class l2tscaffolder.scaffolders.turbinia.**TurbiniaJobTaskScaffolder**
Bases: *l2tscaffolder.scaffolders.interface.Scaffolder*

The Turbinia base scaffolder interface.

class_name
class name of the Turbinia job and task to be generated.

Type str

DESCRIPTION = 'Provides a scaffolder to generate a Turbinia job and task plugins.'

GenerateFiles() → Iterator[Tuple[str, str]]
Generates all the files required for a Turbinia component.

Yields *list[tuple]* –

containing: str: file name. str: file content.

GetInitFileChanges() → Iterator[Tuple[str, str]]
Generate a list of init files that need changing and the changes to them.

Yields *Tuple[str, str]* – path to the init file and the entry to add to it.

GetJinjaContext() → Dict[str, object]
Returns a dict that can be used as a context for Jinja2 templates.

Returns

containing: str: name of Jinja argument. object: Jinja argument value.

Return type dict

NAME = 'turbinia_job_and_task'

PROJECT = 'turbinia'

TEMPLATE_JOB_FILE = 'turbinia_job.jinja2'

TEMPLATE_TASK_FILE = 'turbinia_task.jinja2'

Module contents

This file imports Python modules that registers scaffolders.

2.1.6 l2tscaffolder.templates package

Module contents

2.2 Module contents

defining the version

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W