King Phisher Documentation

Release 1.16.0b0

Spencer McIntyre

Jan 19, 2020

Technical Documentation

King Phisher is an open source Phishing Campaign Toolkit. This is its technical documentation intended for use by contributors. The source code is available on the GitHub homepage. Additionally documentation intended for use by users can be found in the King Phisher GitHub wiki.

The *Architecture Overview* development reference is available to help new users understand the basic project components and how they interact.

CHAPTER 1

The King Phisher Package

1.1 client

This package contains all packages and modules specific to the client application.

1.1.1 assistants

campaign

Classes

class CampaignAssistant(*application*, *campaign_id=None*)

Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display an assistant which walks the user through creating a new campaign or configuring an existing campaign. If no *campaign_id* is specified a new campaign will be created.

__init__(*application*, *campaign_id=None*)

Parameters

- **application** (*KingPhisherClientApplication*) – The application instance which this object belongs to.
- **campaign_id** – The ID of the campaign to edit.

campaign_name

The string value of the configured campaign name. This may be set even when the campaign was not created, which would be the case if the user closed the window.

1.1.2 dialogs

about

Classes

class AboutDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a Gtk.AboutDialog with information regarding the King Phisher client.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

campaign_selection

Classes

class CampaignSelectionDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog which allows a new campaign to be created or an existing campaign to be opened.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

load_campaigns(*cursor=None*)
Load campaigns from the remote server and populate the Gtk.TreeView.

clone_page

Classes

class ClonePageDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog for cloning a web page. The logic for the cloning operation is provided by the *web_cloner* module.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

company_editor

Classes

class CompanyEditorDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog which can be used to edit the various fields associated with a company object.

configuration

Classes

class ConfigurationDialog(**args*, ***kwargs*)

Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display the King Phisher client configuration dialog. Running this dialog via the `interact()` method will cause some server settings to be loaded.

__init__(**args*, ***kwargs*)

Parameters **application** (`Gtk.Application`) – The parent application for this object.

entry

Classes

class TextEntryDialog(**args*, ***kwargs*)

Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a `Gtk.Dialog` with a text entry suitable for prompting users for text input. If the user confirms the action, the text within the entry is returned. If the user cancels the action or closes the dialog, None is returned.

__init__(**args*, ***kwargs*)

Parameters **application** (`Gtk.Application`) – The parent application for this object.

classmethod build_prompt(*application*, *title*, *label_text*, *entry_text=None*, *entry_tooltip_text=None*)

Create a *TextEntryDialog* instance configured with the specified text prompts.

Parameters

- **application** (`Gtk.Application`) – The parent application for this object.
- **title** (*str*) – The title to set for the dialog window.
- **label_text** (*str*) – The text to display in the entry's label.
- **entry_text** (*str*) – Text to place in the entry.
- **entry_tooltip_text** (*str*) – Text to display in the tool tip of the entry.

Returns If the prompt is submitted by the user, the text within the entry is returned.

Return type str

exception

Functions

format_exception_details(*exc_type*, *exc_value*, *exc_traceback*, *error_uid=None*)

Format exception details to be show to a human. This should include enough information about the type of error that occurred and the system on which it was triggered to allow someone to attempt to debug and fix it. The first three parameters to this function directly correspond to the values returned from the `sys.exc_info()` function.

Parameters

- **exc_type** – The type of the exception.
- **exc_value** – The exception instance.
- **exc_traceback** – The traceback object corresponding to the exception.
- **error_uid** (str, uuid.UUID) – A unique identifier for this exception.

Returns A formatted message containing the details about the exception and environment.

Return type str

format_exception_name(*exc_type*)

Format the exception name into a more easily recognizable format.

Parameters **exc_type** – The type of the exception.

Returns The formatted exception name.

Return type str

Classes

class ExceptionDialog(*application*, *exc_info=None*, *error_uid=None*)

Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog which shows an error message for a python exception. The dialog includes useful details for reporting and debugging the exception which occurred.

__init__(*application*, *exc_info=None*, *error_uid=None*)

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **exc_info** (*tuple*) – The exception information as provided by sys.exc_info().
- **error_uid** (*str*) – An optional unique identifier for the exception that can be provided for tracking purposes.

login

Classes

class LoginDialogBase(**args*, ***kwargs*)

Bases: *king_phisher.client.gui_utilities.GladeGObject*

This object is basic login dialog object that can be inherited from and customized.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

class LoginDialog(**args*, ***kwargs*)

Bases: *king_phisher.client.dialogs.login.LoginDialogBase*

This object is the main King Phisher login dialog, it is used to prompt for connection information for the King Phisher server.

It allows the user to specify the host and port to connect to and credentials for authentication.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

class SMTPLoginDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.dialogs.login.LoginDialogBase*

This object is the King Phisher SMTP login dialog, it is used to prompt for connection information to an SMTP server.

It allows the user to specify the host and port to connect to and credentials for authentication.

class SSHLoginDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.dialogs.login.LoginDialogBase*

This object is the King Phisher SSH login dialog, it is used to prompt for connection information to an SSH server.

It allows the user to specify the host and port to connect to and credentials for authentication.

ssh_host_key

Classes

class BaseHostKeyDialog(*application*, *hostname*, *key*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

A base class for dialogs which show information about SSH host keys. It is assumed that the widgets defined in *dependencies* are present including one button to accept the host key, and one to reject. The class's default response can be set using *default_response*.

__init__(*application*, *hostname*, *key*)

Parameters

- **application** (*KingPhisherClientApplication*) – The application to associate this popup dialog with.
- **hostname** (*str*) – The hostname associated with the key.
- **key** (paramiko.pkey.PKey) – The host's SSH key.

default_response = None
The response that should be selected as the default for the dialog.

class HostKeyAcceptDialog(*application*, *hostname*, *key*)
Bases: *king_phisher.client.dialogs.ssh_host_key.BaseHostKeyDialog*

A dialog that shows an SSH host key for a host that has not previously had one associated with it.

class HostKeyWarnDialog(*application*, *hostname*, *key*)
Bases: *king_phisher.client.dialogs.ssh_host_key.BaseHostKeyDialog*

A dialog that warns about an SSH host key that does not match the one that was previously stored for the host.

class MissingHostKeyPolicy(*application*)
Bases: paramiko.client.MissingHostKeyPolicy

A host key policy for use with paramiko that will validate SSH host keys correctly. If a key is new, the user will be prompted with *HostKeyAcceptDialog* dialog to accept it or if the host key does not match the user will be warned with *HostKeyWarnDialog*. The host keys accepted through this policy are stored in an OpenSSH compatible "known_hosts" file using paramiko.

__init__(*application*)

Parameters **application** (*KingPhisherClientApplication*) – The application which is using this policy.

missing_host_key(*client*, *hostname*, *key*)
Called when an *.SSHClient* receives a server key for a server that isn't in either the system or local *.HostKeys* object. To accept the key, simply return. To reject, raised an exception (which will be passed to the calling application).

tag_editor

Classes

class TagEditorDialog(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog which can be used to edit the various tags that are present on the remote server. This can be used to rename tags and modify their descriptions.

1.1.3 tabs

This package contains modules for providing the content of the top level tabs used by the main application window.

campaign

This module provides the contents of the tab representing the campaign information in client's graphical interface.

Classes

class CampaignViewCredentialsTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTableTab*

Display campaign information regarding submitted credentials.

format_node_data(*node*)
This method is overridden by subclasses to format the raw node data returned from the server. The length of the list must equal the number of columns in the table. This method is called for each node in the remote table by the loader thread.

Parameters **node** (*dict*) – The node from a GraphQL query representing data for this table.

Returns The formatted row data.

Return type list

class CampaignViewDashboardTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTab*

Display campaign information on a graphical dash board.

graphs = None
The *CampaignGraph* classes represented on the dash board.

label_text = 'Dashboard'
The tabs label for display in the GUI.

load_campaign_information(*force=True*)
Load the necessary campaign information from the remote server. Unless *force* is True, the *last_load_time* is compared with the *refresh_frequency* to check if the information is stale. If the local data is not stale, this function will return without updating the table.

Parameters **force** (*bool*) – Ignore the load life time and force loading the remote data.

loader_idle_routine()
The routine which refreshes the campaign data at a regular interval.

loader_thread_routine()
The loading routine to be executed within a thread.

class CampaignViewDeaddropTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTableTab*

Display campaign information regarding dead drop connections.

format_node_data(*connection*)
This method is overridden by subclasses to format the raw node data returned from the server. The length of the list must equal the number of columns in the table. This method is called for each node in the remote table by the loader thread.

Parameters **node** (*dict*) – The node from a GraphQL query representing data for this table.

Returns The formatted row data.

Return type list

class CampaignViewGenericTab(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

This object is meant to be subclassed by all of the tabs which load and display information about the current campaign.

label = None
The Gtk.Label representing this tab with text from *label_text*.

label_text = 'Unknown'
The label of the tab for display in the GUI.

last_load_time = None
The last time the data was loaded from the server.

loader_thread = None
The thread object which loads the data from the server.

loader_thread_lock = None
The threading.Lock object used for synchronization between the loader and main threads.

loader_thread_stop = None
The threading.Event object used to request that the loader thread stop before completion.

refresh_frequency = None
The lifetime in seconds to wait before refreshing the data from the server.

class CampaignViewGenericTableTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTab*

This object is meant to be subclassed by tabs which will display campaign information of different types from specific database tables. The data in this object is refreshed when multiple events occur and it uses an internal timer to represent the last time the data was refreshed.

export_table_to_csv(*filtered=False*)
Export the data represented by the view to a CSV file.

export_table_to_xlsx_worksheet(*worksheet*, *title_format*)
Export the data represented by the view to an XLSX worksheet.

Parameters

- **worksheet** (xlsxwriter.worksheet.Worksheet) – The destination sheet for the store's data.
- **title_format** (xlsxwriter.format.Format) – The formatting to use for the title row.

format_node_data(*node*)
This method is overridden by subclasses to format the raw node data returned from the server. The length of the list must equal the number of columns in the table. This method is called for each node in the remote table by the loader thread.

Parameters **node** (*dict*) – The node from a GraphQL query representing data for this table.

Returns The formatted row data.

Return type list

load_campaign_information(*force=True*)
Load the necessary campaign information from the remote server. Unless *force* is True, the *last_load_time* is compared with the *refresh_frequency* to check if the information is stale. If the local data is not stale, this function will return without updating the table.

Parameters **force** (*bool*) – Ignore the load life time and force loading the remote data.

loader_thread_routine(*store*)
The loading routine to be executed within a thread.

Parameters **store** (Gtk.ListStore) – The store object to place the new data.

node_query = None
The GraphQL query used to load a particular node from the remote table. This query is provided with a single parameter of the node's id.

popup_menu = None
The Gtk.Menu object which is displayed when right-clicking in the view area.

table_name = ''
The database table represented by this tab.

table_query = None
The GraphQL query used to load the desired information from the remote table. This query is provided with the following three parameters: campaign, count and cursor.

class CampaignViewMessagesTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTableTab*

Display campaign information regarding sent messages.

format_node_data(*node*)
This method is overridden by subclasses to format the raw node data returned from the server. The length of the list must equal the number of columns in the table. This method is called for each node in the remote table by the loader thread.

Parameters **node** (*dict*) – The node from a GraphQL query representing data for this table.

Returns The formatted row data.

Return type list

class CampaignViewTab(*parent*, *application*)
Bases: object

The King Phisher client top-level 'View Campaign' tab. This object manages the sub-tabs which display all the information regarding the current campaign.

__init__(*parent*, *application*)

Parameters

- **parent** (Gtk.Window) – The parent window for this object.
- **application** (Gtk.Application) – The main client application instance.

label = None
The Gtk.Label representing this tabs name.

notebook = None
The Gtk.Notebook for holding sub-tabs.

tabs = None
A dict object holding the sub tabs managed by this object.

class CampaignViewVisitsTab(**args*, ***kwargs*)
Bases: *king_phisher.client.tabs.campaign.CampaignViewGenericTableTab*

Display campaign information regarding incoming visitors.

format_node_data(*node*)
This method is overridden by subclasses to format the raw node data returned from the server. The length of the list must equal the number of columns in the table. This method is called for each node in the remote table by the loader thread.

Parameters **node** (*dict*) – The node from a GraphQL query representing data for this table.

Returns The formatted row data.

Return type list

mail

This module provides the contents of the tab used to create and send messages as part of a campaign.

Functions

test_webserver_url(*target_url*, *secret_id*)
Test the target URL to ensure that it is valid and the server is responding.

Parameters

- **target_url** (*str*) – The URL to make a test request to.
- **secret_id** (*str*) – The King Phisher Server secret id to include in the test request.

Classes

class MailSenderConfigurationTab(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

This is the tab which allows the user to configure and set parameters for sending messages as part of a campaign.

label = None
The Gtk.Label representing this tabs name.

objects_load_from_config()
Iterate through *gobjects* and set the GObject's value from the corresponding value in the *config*.

class MailSenderEditTab(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

This is the tab which adds basic text edition for changing an email template.

label = None
The Gtk.Label representing this tabs name.

load_html_file()
Load the contents of the configured HTML file into the editor.

save_html_file(*force_prompt=False*)
Save the contents from the editor into an HTML file if one is configured otherwise prompt to user to select a file to save as. The user may abort the operation by declining to select a file to save as if they are prompted to do so.

Parameters **force_prompt** – Force prompting the user to select the file to save as.

Return type bool

Returns Whether the contents were saved or not.

show_tab()
Load the message HTML file from disk and configure the tab for editing.

textbuffer = None
The Gtk.TextBuffer used by the :py:attr:textview‘ attribute.

textview = None
The Gtk.TextView object of the editor.

class MailSenderPreviewTab(*application*)
Bases: object

This tab uses the WebKit engine to render the HTML of an email so it can be previewed before it is sent.

__init__(*application*)

Parameters **application** (*KingPhisherClientApplication*) – The application instance.

label = None
The Gtk.Label representing this tabs name.

load_html_file()
Load the configured HTML file into the WebKit engine so the contents can be previewed.

show_tab()
Configure the webview to preview the the message HTML file.

webview = None
The *WebKitHTMLView* object used to render the message HTML.

class MailSenderSendTab(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

This allows the *MailSenderThread* object to be managed by the user through the GUI. These two classes are very interdependent

label = None
The Gtk.Label representing this tabs name.

notify_sent(*emails_done*, *emails_total*)
A call back use by *MailSenderThread* to notify when an email has been successfully sent to the SMTP server.

Parameters

- **emails_done** (*int*) – The number of email messages that have been sent.
- **emails_total** (*int*) – The total number of email messages that need to be sent.

notify_status(*message*)
A call back use by *MailSenderThread* to update general status information.

Parameters **message** (*str*) – The status message.

notify_stopped()
A callback used by *MailSenderThread* to notify when the thread has stopped.

precheck_routines = ('settings', 'attachment', 'required-files', 'campaign', 'url', 's
The built-in precheck routines that are executed before sending messages.

progressbar = None
The Gtk.ProgressBar instance which is used to display progress of sending messages.

sender_start_failure(*message=None*, *text=None*, *retry=False*)
Handle a failure in starting the message sender thread and perform any necessary clean up.

Parameters

- **message** (*text*) – A message to shown in an error popup dialog.
- **message** – A message to be inserted into the text buffer.
- **retry** (*bool*) – The operation will be attempted again.

sender_thread = None
The *MailSenderThread* instance that is being used to send messages.

text_insert(*message*)
Insert text into the *textbuffer*.

Parameters **message** (*str*) – The text to insert.

textbuffer = None
The Gtk.TextBuffer instance associated with *textview*.

textview = None
The Gtk.TextView object that renders text status messages.

class MailSenderTab(*parent*, *application*)
Bases: GObject.GObject

The King Phisher client top-level 'Send Messages' tab. This object manages the sub-tabs which display useful information for configuring, previewing and sending messages as part of a campaign.

GObject Signals *Mail Tab Signals*

__init__(*parent*, *application*)

Parameters

- **parent** (Gtk.Window) – The parent window for this object.
- **application** (Gtk.Application) – The main client application instance.

export_message_data(*path=None*)
Gather and prepare the components of the mailer tab to be exported into a King Phisher message (KPM) archive file suitable for restoring at a later point in time. If *path* is not specified, the user will be prompted to select one and failure to do so will prevent the message data from being exported. This function wraps the emission of the `message-data-export` signal.

Parameters **path** (*str*) – An optional path of where to save the archive file to.

Returns Whether or not the message archive file was written to disk.

Return type bool

import_message_data()
Process a previously exported message archive file and restore the message data, settings, and applicable files from it. This function wraps the emission of the `message-data-import` signal.

Returns Whether or not the message archive file was loaded from disk.

Return type bool

label = None
The Gtk.Label representing this tabs name.

notebook = None
The Gtk.Notebook for holding sub-tabs.

tabs = None
A dict object holding the sub tabs managed by this object.

1.1.4 widget

extras

This module contains miscellaneous extra GTK widgets.

Classes

Cell Renderers

class CellRendererPythonText(**args*, ***kwargs*)
Bases: Gtk.CellRendererText

A base Gtk.CellRendererText class to facilitate rendering native Python values into strings of various formats.

render_python_value(*value*)
The method to render *value* into a string to be displayed within the cell.

Parameters **value** – The Python value to render.

Return type str

Returns Either the value rendered as a string or None. Returning None will cause the cell to be displayed as empty.

class CellRendererBytes(**args*, ***kwargs*)
Bases: *king_phisher.client.widget.extras.CellRendererPythonText*

A custom *CellRendererPythonText* to render numeric values representing bytes.

class CellRendererDatetime(**args*, ***kwargs*)
Bases: *king_phisher.client.widget.extras.CellRendererPythonText*

A custom *CellRendererPythonText* to render numeric values representing bytes.

class CellRendererInteger(**args*, ***kwargs*)
Bases: *king_phisher.client.widget.extras.CellRendererPythonText*

A custom *CellRendererPythonText* to render numeric values with comma separators.

Column Definitions

class ColumnDefinitionBase(*title*, *width*)
Bases: object

A base class for defining attributes of columns to be displayed within TreeView instances.

__init__(*title*, *width*)
Initialize self. See help(type(self)) for accurate signature.

cell_renderer = None
The CellRenderer to use for renderering the content.

g_type = None
The type to specify in the context of GObjects.

name
The title converted to lowercase and with spaces replaced with underscores.

python_type = None
The type to specify in the context of native Python code.

sort_function = None
An optional custom sort function to use for comparing values. This is necessary when *g_type* is not something that can be automatically sorted. If specified, this function will be passed to Gtk.TreeSortable.set_sort_func().

title
The title of the column to be displayed within the TreeView instance.

width
An integer specifying the width of the column.

class ColumnDefinitionBytes(*title*, *width=25*)
Bases: *king_phisher.client.widget.extras.ColumnDefinitionBase*

class ColumnDefinitionDatetime(*title*, *width=25*)
Bases: *king_phisher.client.widget.extras.ColumnDefinitionBase*

class ColumnDefinitionInteger(*title*, *width=15*)
Bases: *king_phisher.client.widget.extras.ColumnDefinitionBase*

class ColumnDefinitionString(*title*, *width=30*)
Bases: *king_phisher.client.widget.extras.ColumnDefinitionBase*

Miscellaneous

class FileChooserDialog(*title*, *parent*, ***kwargs*)
Bases: Gtk.FileChooserDialog

Display a file chooser dialog with additional convenience methods.

__init__(*title*, *parent*, ***kwargs*)

Parameters

- **title** (*str*) – The title for the file chooser dialog.
- **parent** (Gtk.Window) – The parent window for the dialog.

quick_add_filter(*name*, *patterns*)
Add a filter for displaying files, this is useful in conjunction with *run_quick_open()*.

Parameters

- **name** (*str*) – The name of the filter.
- **patterns** (*list, str*) – The pattern(s) to match.

run_quick_open()
Display a dialog asking a user which file should be opened. The value of target_path in the returned dictionary is an absolute path.

Returns A dictionary with target_uri and target_path keys representing the path chosen.

Return type dict

run_quick_save(*current_name=None*)
Display a dialog which asks the user where a file should be saved. The value of target_path in the returned dictionary is an absolute path.

Parameters **current_name** (*set*) – The name of the file to save.

Returns A dictionary with target_uri and target_path keys representing the path choosen.

Return type dict

run_quick_select_directory()
Display a dialog which asks the user to select a directory to use. The value of target_path in the returned dictionary is an absolute path.

Returns A dictionary with target_uri and target_path keys representing the path chosen.

Return type dict

class MultilineEntry(*args, **kwargs)
Bases: Gtk.Frame

A custom entry widget which can be styled to look like Gtk.Entry but accepts multiple lines of input.

__init__(*args, **kwargs)
Initialize self. See help(type(self)) for accurate signature.

class WebKitHTMLView
Bases: WebKitX.WebView

A WebView widget with additional convenience methods for rendering simple HTML content from either files or strings. If a link is opened within the document, the webview will emit the 'open-uri' signal instead of navigating to it.

__init__()
Initialize self. See help(type(self)) for accurate signature.

load_html_data(*html_data*, *html_file_uri=None*)
Load arbitrary HTML data into the WebKit engine to be rendered.

Parameters

- **html_data** (*str*) – The HTML data to load into WebKit.
- **html_file_uri** (*str*) – The URI of the file where the HTML data came from.

load_html_file(*html_file*)
Load arbitrary HTML data from a file into the WebKit engine to be rendered.

Parameters **html_file** (*str*) – The path to the file to load HTML data from.

load_markdown_data(*md_data*, *html_file_uri=None*, *gh_flavor=True*, *template=None*, *template_vars=None*)
Load markdown data, render it into HTML and then load it in to the WebKit engine. When *gh_flavor* is enabled, the markdown data is rendered using partial GitHub flavor support as provided by `PartialGithubFlavoredMarkdownExtension`. If *template* is specified, it is used to load a Jinja2 template using *template_env* into which the markdown data is passed in the variable `markdown` along with any others specified in the *template_vars* dictionary.

Parameters

- **md_data** (*str*) – The markdown data to render into HTML for displaying.
- **html_file_uri** (*str*) – The URI of the file where the HTML data came from.
- **gh_flavor** (*bool*) – Whether or not to enable partial GitHub markdown syntax support.
- **template** (*str*) – The name of a Jinja2 HTML template to load for hosting the rendered markdown.
- **template_vars** – Additional variables to pass to the Jinja2 `Template` when rendering it.

Returns

load_markdown_file(*md_file*, ***kwargs*)
Load markdown data from a file and render it using *load_markdown_data()*.

Parameters

- **md_file** (*str*) – The path to the file to load markdown data from.
- **kwargs** – Additional keyword arguments to pass to *load_markdown_data()*.

template_env = <king_phisher.templates.TemplateEnvironmentBase object>
The *TemplateEnvironmentBase* instance to use when rendering template content. The environment uses the *FindFileSystemLoader* loader.

managers

This module contains classes used for high level management of some GTK widgets.

Classes

class ButtonGroupManager(*glade_gobject*, *widget_type*, *group_name*)
Bases: object

Manage a set of buttons. The buttons should all be of the same type (such as "checkbutton" or "radiobutton") and include a common group name prefix. The intent is to make managing buttons of similar functionality easier by grouping them together.

__init__(*glade_gobject*, *widget_type*, *group_name*)

Parameters

- **glade_gobject** (*GladeGObject*) – The gobject which has the radio buttons set.
- **group_name** (*str*) – The name of the group of buttons.

class MenuManager(*menu=None*)
Bases: object

A class that wraps Gtk.Menu objects and facilitates managing their respective items.

__init__(*menu=None*)

Parameters **menu** (Gtk.Menu) – An optional menu to start with. If a menu is specified it is used as is, otherwise a new instance is used and is set to be visible using show().

append(*label*, *activate=None*, *activate_args=()*)
Create and append a new Gtk.MenuItem with the specified label to the menu.

Parameters

- **label** (*str*) – The label for the new menu item.
- **activate** – An optional callback function to connect to the new menu item's activate signal.

Returns Returns the newly created and added menu item.

Return type Gtk.MenuItem

append_item(*menu_item*, *set_show=True*)
Append the specified menu item to the menu.

Parameters

- **menu_item** (Gtk.MenuItem) – The item to append to the menu.
- **set_show** (*bool*) – Whether to set the item to being visible or leave it as is.

append_submenu(*label*)
Create and append a submenu item, then return a new menu manager instance for it.

Parameters **label** (*str*) – The label for the new menu item.

Returns Returns the newly created and added menu item.

Return type Gtk.MenuManager

class RadioButtonGroupManager(*glade_gobject*, *group_name*)
Bases: *king_phisher.client.widget.managers.ButtonGroupManager*

Manage a group of Gtk.RadioButton objects together to allow the active one to be easily set and identified. The buttons are retrieved from a *GladeGObject* instance and must be correctly named in the *dependencies* attribute as 'radiobutton_group_name_button_name'.

__init__(*glade_gobject*, *group_name*)

Parameters

- **glade_gobject** (*GladeGObject*) – The gobject which has the radio buttons set.
- **group_name** (*str*) – The name of the group of buttons.

get_active()
Return the name of the active button if one in the group is active. If no button in the group is active, None is returned.

Returns The name of the active button.

Return type str

set_active(*button*)
Set a button in the group as active.

Parameters **button** (*str*) – The name of the button to set as active.

class TimeSelectorButtonManager(*application*, *button*, *value=None*)
Bases: object

A manager class to convert a ToggleButton to be used for showing a time selector py:class:~.*Gtk.Popover* object with inputs for setting the hour and minutes. This then exposes the selected time through the *time* attribute.

time

__init__(*application*, *button*, *value=None*)

Parameters

- **button** (Gtk.ToggleButton) – The button used for activation.
- **application** – The application instance which owns this object.
- **value** (datetime.time) – The present datetime value (defaults to 00:00).

time
This property represents the current time value and when set, updates the associated button.

Returns The current time value.

Return type datetime.time

class ToggleButtonGroupManager(*glade_gobject*, *widget_type*, *group_name*)
Bases: *king_phisher.client.widget.managers.ButtonGroupManager*

Manage a mapping of button names to a boolean value indicating whether they are active or not.

get_active()
Get the button names and whether or not they are active.

Returns A mapping of button names to whether or not they are active.

Return type dict

set_active(*buttons*)
Set the specified buttons to active or not.

Parameters **buttons** (*dict*) – A mapping of button names to boolean values.

class TreeViewManager(*treeview*, *selection_mode=None*, *cb_delete=None*, *cb_refresh=None*)
Bases: object

A class that wraps Gtk.TreeView objects that use *Gtk.ListStore* models with additional functions for conveniently displaying text data.

If *cb_delete* is specified, the callback will be called with the treeview instance, and the selection as the parameters.

If *cb_refresh* is specified, the callback will be called without any parameters.

__init__(*treeview*, *selection_mode=None*, *cb_delete=None*, *cb_refresh=None*)

Parameters

- **treeview** (Gtk.TreeView) – The treeview to wrap and manage.
- **selection_mode** (Gtk.SelectionMode) – The selection mode to set for the treeview.
- **cb_delete** (*function*) – An optional callback that can be used to delete entries.

cb_delete = None
An optional callback for deleting entries from the treeview's model.

cb_refresh = None
An optional callback for refreshing the data in the treeview's model.

column_titles = None
An ordered dictionary of storage data columns keyed by their respective column titles.

column_views = None
A dictionary of column treeview's keyed by their column titles.

get_popup_copy_submenu()
Create a Gtk.Menu with entries for copying cell data from the treeview.

Returns The populated copy popup menu.

Return type Gtk.Menu

get_popup_menu(*handle_button_press=True*)
Create a Gtk.Menu with entries for copying and optionally delete cell data from within the treeview. The delete option will only be available if a delete callback was previously set.

Parameters **handle_button_press** (*bool*) – Whether or not to connect a handler for displaying the popup menu.

Returns The populated popup menu.

Return type Gtk.Menu

set_column_color(*background=None*, *foreground=None*, *column_titles=None*)
Set a column in the model to be used as either the background or foreground RGBA color for a cell.

Parameters

- **background** (*int*) – The column id of the model to use as the background color.
- **foreground** (*int*) – The column id of the model to use as the foreground color.
- **column_titles** (*str, tuple*) – The columns to set the color for, if None is specified all columns will be set.

set_column_titles(*column_titles*, *column_offset=0*, *renderers=None*)
Populate the column names of a GTK TreeView and set their sort IDs. This also populates the *column_titles* attribute.

Parameters

- **column_titles** (*list*) – The titles of the columns.
- **column_offset** (*int*) – The offset to start setting column names at.

- **renderers** (*list*) – A list containing custom renderers to use for each column.

Returns A dict of all the `Gtk.TreeViewColumn` objects keyed by their column id.

Return type dict

treeview = None
The `Gtk.TreeView` instance being managed.

resources

This module contains resources useful to GTK widgets.

Data

font_desc_italic
A `Pango.FontDescription` configured for representing italicized text.

renderer_text_desc
A `Gtk.CellRendererText` instance which is configured to be suitable for showing descriptions of various object.

Classes

class CompanyEditorGrid(*destination*)
Bases: *king_phisher.client.gui_utilities.GladeProxy*

An embeddable widget which contains the necessary widgets to edit the various fields of a company object.

children = ('combobox_company_industry', 'entry_company_industry', 'entry_company_name
The children widgets that can be used to edit the fields of the company.

name = 'CompanyEditorGrid'
The name of the top level widget in the GTK Builder data file.

completion_providers

This module contains classes for custom auto completion for GtkSourceCompletion. It provides support to recognize special characters and suggests syntax completion.

Functions

get_proposal_terms(*search*, *tokens*)
Used to iterate through the *search* dictionary definition representing tokens for completion. Terms within this dictionary have a hierarchy to their definition in which keys are always terms represented as strings and values are either sub-dictionaries following the same pattern or None in the case that the term is a leaf node.

Parameters

- **search** (*dict*) – The dictionary to iterate through looking for proposals.
- **tokens** (*list, str*) – List of tokens split on the hierarchy delimiter.

Returns A list of strings to be used for completion proposals.

Return type list

Classes

CustomCompletionProviderBase
alias of king_phisher.utilities.

HTMLCompletionProvider
alias of king_phisher.utilities.

JinjaCompletionProvider
alias of king_phisher.utilities.

JinjaEmailCompletionProvider
alias of king_phisher.utilities.

JinjaPageCompletionProvider
alias of king_phisher.utilities.

1.1.5 windows

This package contains modules for providing GTK Window objects used by the client application.

campaign_import

This module provides the window through which the user can import King Phisher campaigns from xml files previously exported with the export module.

Classes

class ImportCampaignWindow(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

Display a dialog which allows a new campaign to be created or an existing campaign to be opened.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

preprep_xml_data()
This function provides the actions required to see if required IDs are already in the database. If they are not it will clear them out and set subelement.attrib['type'] to null. If the element is required it will set it to a default value. This will normalize the data and ready it for import into the database.

remove_import_campaign()
Used to delete the imported campaign on failure or early exit of the import window, if the user selects to have it removed.

select_xml_campaign()
Prompts the user with a file dialog window to select the King Phisher Campaign XML file to import. Validates the file to make sure it is a Campaign exported from King Phisher and is the correct version to import.

signal_entry_change(_)
When there is a change in the campaign entry field it will check to see if the name is already in use. If it is not in use it will change the sensitivity of the button_import_campaign to allow the user to start the import process.

signal_import_button(_)
This will check to see if the campaign information is present. If campaign information is present it will launch an py:class:*ImportThread* to import the campaign in the background, freeing up the GUI for the user to conduct other functions.

signal_window_delete_event(_, *event*)
Checks to make sure the import campaign thread is closed before closing the window.

compare_campaigns

This module provides the window through which the user can compare campaigns across multiple data points in graph format

Classes

class CampaignCompWindow(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

The window which allows the user to select campaigns and compare the data using graphical representation.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

load_campaigns()
Load campaigns from the remote server and populate the Gtk.TreeView.

html

This module provides a window which shows HTML content.

Classes

class HTMLWindow(*application*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

This basic window contains a *WebKitHTMLView* widget for rendering and displaying HTML content.

__init__(*application*)

Parameters **application** (Gtk.Application) – The parent application for this object.

webview = None
The *WebKitHTMLView* widget instance.

main

This module provides the main window used by the client application.

Classes

class MainAppWindow(*config*, *application*)
Bases: Gtk.ApplicationWindow

This is the top level King Phisher client window. This is also the parent window for most GTK objects.

__init__(*config*, *application*)

Parameters

- **config** (*dict*) – The main King Phisher client configuration.
- **application** (*KingPhisherClientApplication*) – The application instance to which this window belongs.

config = None
The main King Phisher client configuration.

export_campaign_visit_geojson()
Export the current campaign visit information to a GeoJSON data file.

export_campaign_xlsx()
Export the current campaign to an Excel compatible XLSX workbook.

export_campaign_xml()
Export the current campaign to an XML data file.

notebook = None
The primary Gtk.Notebook that holds the top level taps of the client GUI.

rpc = None
The *KingPhisherRPCClient* instance.

class MainMenuBar(*application*, *window*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

The main menu bar for the primary application window. This configures any optional menu items as well as handles all the menu item signals appropriately.

__init__(*application*, *window*)

Parameters **application** (Gtk.Application) – The parent application for this object.

plugin_manager

This module provides the window through which the user can enable and disable plugins.

Classes

class PluginDocumentationWindow(*application*, *plugin_id*)
Bases: *king_phisher.client.windows.html.HTMLWindow*

A window for displaying plugin documentation from their respective README.md files. If the documentation file can not be found a FileNotFoundError exception will be raised on initialization. The contents of the README.md file is then rendered as markdown data and displayed using an *HTMLWindow*. The plugin must be loaded into the *plugin_manager* but does not have to be enabled for documentation to be displayed.

__init__(*application*, *plugin_id*)

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **plugin_id** (*str*) – The identifier of this plugin.

refresh()
Refresh the contents of the documentation. This will reload both the markdown content from README.md as well as the HTML template file.

template = 'plugin-documentation.html'
The Jinja2 HTML template to load for hosting the rendered markdown documentation.

class PluginManagerWindow(**args*, ***kwargs*)
Bases: *king_phisher.client.gui_utilities.GladeGObject*

The window which allows the user to selectively enable and disable plugins for the client application. This also handles configuration changes, so the enabled plugins will persist across application runs.

__init__(**args*, ***kwargs*)

Parameters **application** (Gtk.Application) – The parent application for this object.

rpc_terminal

This module provides the RPC Terminal window used by the client application to give the user raw access to the RPC interface.

Data

has_vte = True
Whether the Vte module is available or not.

Classes

class RPCTerminal(*application*)
Bases: object

A terminal using VTE that allows raw RPC methods to be called from within the King Phisher client. This is primarily useful for unofficial and advanced features or debugging and development.

__init__(*application*)

Parameters **application** (*KingPhisherClientApplication*) – The application instance to which this window belongs.

1.1.6 application

This module provides the top level GTK application object representing the client application.

Data

GTK3_DEFAULT_THEME = 'Adwaita'
The default GTK3 Theme for style information.

USER_DATA_PATH = 'king-phisher'
The default folder name for user specific data storage.

Classes

class ServerUser(*id*, *name*)
Bases: tuple

A named tuple representing the user that is authenticated on the remote server.

id
The user's unique identifier.

name
The user's name.

class KingPhisherClientApplication(*config_file=None*, *use_plugins=True*, *use_style=True*)
Bases: Gtk.Application

This is the top level King Phisher client object. It contains the custom GObject signals, keeps all the GUI references, and manages the RPC client object. This is also the parent window for most GTK objects.

GObject Signals *Signal Flags*

__init__(*config_file=None*, *use_plugins=True*, *use_style=True*)
Initialize self. See help(type(self)) for accurate signature.

add_reference(*ref_object*)
Add *ref_object* to the *references* so the object won't be garbage collected. The object must either be a *GladeGObject* or Gtk.Widget instance so a cleanup function can be attached to a destroy signal to remove the reference automatically.

Parameters **ref_object** (*GladeGObject*, Gtk.Widget) – The object to store a reference to.

config = None
The primary King Phisher client configuration.

config_file = None
The file containing the King Phisher client configuration.

do_campaign_delete(*campaign_id*)
Delete the campaign on the server. A confirmation dialog will be displayed before the operation is performed. If the campaign is deleted and a new campaign is not selected with *show_campaign_selection()*, the client will quit.

do_config_load(*load_defaults*)
Load the client configuration from disk and set the *config* attribute.

Check the proxy environment variable and set them appropriately.

Parameters **load_defaults** (*bool*) – Load missing options from the template configuration file.

do_server_disconnected()
Clean up the connections to the server and disconnect. This logs out of the RPC, closes the server event socket, and stops the SSH forwarder.

do_sftp_client_start()
Start the client's preferred sftp client application in a new process.

get_graphql_campaign(*campaign_id=None*)
Retrieve the GraphQL representation of the specified campaign. If *campaign_id* is not specified, then the data for the current campaign is retrieved.

Parameters **campaign_id** (*str*) – The ID for the campaign whose information should be retrieved.

Returns The campaign's GraphQL representation.

Return type dict

load_server_config()
Load the necessary values from the server's configuration.

main_window = None
The primary top-level *MainAppWindow* instance.

merge_config(*config_file*, *strict=True*)
Merge the configuration information from the specified configuration file. Only keys which exist in the currently loaded configuration are copied over while non-existent keys are skipped. The contents of the new configuration overwrites the existing.

Parameters

- **strict** (*bool*) – Do not try remove trailing commas from the JSON data.
- **config_file** (*str*) – The path to the configuration file to merge.

plugin_manager = None
The *ClientPluginManager* instance to manage the installed client plugins.

quit(*optional=False*)
Quit the client and perform any necessary clean up operations. If *optional* is False then the exit-confirm signal will not be sent and there will not be any opportunities for the client to cancel the operation.

Parameters **optional** (*bool*) – Whether the quit is request is optional or not.

references = None
A list to store references to arbitrary objects in for avoiding garbage collection.

rpc = None
The *KingPhisherRPCClient* instance for the application.

server_connect(*username*, *password*, *otp=None*, *window=None*)
Initialize the connection to the King Phisher server.

Parameters

- **username** (*str*) – The username to authenticate with.
- **password** (*str*) – The password to authenticate with.
- **otp** (*str*) – The optional one-time password to authenticate with.
- **window** (*Gtk.Window*) – The GTK window to use as the parent for error dialogs.

Return type tuple

server_events = None
The *ServerEventSubscriber* instance for the application to receive server events.

server_user = None
The *ServerUser* instance for the authenticated user.

show_campaign_graph(*graph_name*)
Create a new *CampaignGraph* instance and make it into a window. *graph_name* must be the name of a valid, exported graph provider.

Parameters **graph_name** (*str*) – The name of the graph to make a window of.

show_campaign_selection()
Display the campaign selection dialog in a new *CampaignSelectionDialog* instance.

Returns Whether or not a campaign was selected.

Return type bool

show_preferences()
Display a *dialogs.configuration.ConfigurationDialog* instance and saves the configuration to disk if cancel is not selected.

stop_remote_service()
Stop the remote King Phisher server. This will request that the server stop processing new requests and exit. This will display a confirmation dialog before performing the operation. If the remote service is stopped, the client will quit.

user_data_path = None
The path to a directory where user data files can be stored. This path must be writable by the current user.

The default value is platform dependant:

Linux `~/.config/king-phisher`

Windows `%LOCALAPPDATA%\king-phisher`

1.1.7 client_rpc

This module facilitates communication with the server application over the RPC interface.

Data

UNRESOLVED = UNRESOLVED
A sentinel value used for values in rows to indicate that the data has not been loaded from the server.

Functions

vte_child_routine(*config*)
This is the method which is executed within the child process spawned by VTE. It expects additional values to be set in the *config* object so it can initialize a new *KingPhisherRPCClient* instance. It will then drop into an interpreter where the user may directly interact with the rpc object.

Parameters **config** (*str*) – A JSON encoded client configuration.

Classes

class KingPhisherRPCClient(**args*, ***kwargs*)
Bases: advancedhttpserver.RPCClientCached

The main RPC object for communicating with the King Phisher Server over RPC.

New in version 1.14.0: Asynchronous Methods

This RPC object provides a few methods for asynchronously making RPC calls to the server. This makes it easier to issue and RPC call and then process the results without having to either wait (and by extension lock the GUI thread) or start and manage a separate thread. These methods use the name `async_` prefix and have many of the same arguments.

In all cases, the callback parameters *on_success* and *on_error* are called with the signature `callback(*(cb_args + (results,)), **cb_kwargs)` where `results` is either the return value of the RPC method in the case of *on_success* or the exception instance in the case of *on_error*. The *when_idle* parameter can be used to specify that the callbacks must be executed within the main GUI thread and can thus access GObjects such as widgets.

async_call(*method*, *args=None*, *kwargs=None*, *on_success=None*, *on_error=None*, *when_idle=False*, *cb_args=None*, *cb_kwargs=None*)

Perform an asynchronous RPC call to the server. This will queue a work item for a thread to issue the RPC call and then specifies the behavior for completion. See *Asynchronous Methods* for more information.

New in version 1.14.0.

Parameters

- **method** (*str*) – The RPC method name to call.
- **args** (*tuple*) – The arguments to the RPC method.
- **kwargs** (*tuple*) – The keyword arguments to the RPC method.
- **on_success** – A callback function to be called after the RPC method returns successfully.
- **on_error** – A callback function to be called if the RPC method raises an exception.
- **when_idle** – Whether or not the *on_success* and *on_error* callback functions should be called from the main GUI thread while it is idle.
- **cb_args** – The arguments to the *on_success* and *on_error* callback functions.
- **cb_kwargs** – The keyword arguments to the *on_success* and *on_error* callback functions.

async_graphql(*query*, *query_vars=None*, *on_success=None*, *on_error=None*, *when_idle=False*, *cb_args=None*, *cb_kwargs=None*)

Perform an asynchronous RPC GraphQL query to the server. This will queue a work item for a thread to issue the RPC call and then specifies the behavior for completion. See *Asynchronous Methods* for more information.

New in version 1.14.0.

Parameters

- **query** (*str*) – The GraphQL query string to execute asynchronously.
- **query_vars** (*dict*) – Any variable definitions required by the GraphQL query.
- **on_success** – A callback function to be called after the RPC method returns successfully.
- **on_error** – A callback function to be called if the RPC method raises an exception.
- **when_idle** – Whether or not the *on_success* and *on_error* callback functions should be called from the main GUI thread while it is idle.
- **cb_args** – The arguments to the *on_success* and *on_error* callback functions.
- **cb_kwargs** – The keyword arguments to the *on_success* and *on_error* callback functions.

async_graphql_file(*file_or_path*, **args*, ***kwargs*)

Perform an asynchronous RPC GraphQL query from a file on the server. This will queue a work item for a thread to issue the RPC call and then specifies the behavior for completion. See *Asynchronous Methods* for more information.

New in version 1.14.0.

Parameters **file_or_path** – The file object or path to the file from which to read.

geoip_lookup(*ip*)
Look up the geographic location information for the specified IP address in the server's geoip database.

Parameters **ip** (ipaddress.IPv4Address, str) – The IP address to lookup.

Returns The geographic location information for the specified IP address.

Return type *GeoLocation*

geoip_lookup_multi(*ips*)
Look up the geographic location information for the specified IP addresses in the server's geoip database. Because results are cached for optimal performance, IP addresses to be queried should be grouped and sorted in a way that is unlikely to change, i.e. by a timestamp.

Parameters **ips** (*list*, *set*, *tuple*) – The IP addresses to lookup.

Returns The geographic location information for the specified IP address.

Return type dict

get_tag_model(*tag_table*, *model=None*)
Load tag information from a remote table into a Gtk.ListStore instance. Tables compatible with the tag interface must have id, name and description fields. If no *model* is provided a new one will be created, else the current model will be cleared.

Parameters

- **tag_table** (*str*) – The name of the table to load tag information from.
- **model** (Gtk.ListStore) – The model to place the information into.

Returns The model with the loaded data from the server.

Return type Gtk.ListStore

graphql(*query*, *query_vars=None*)
Execute a GraphQL query on the server and return the results. This will raise *KingPhisherGraphQLQueryError* if the query fails.

Parameters

- **query** (*str*) – The GraphQL query string to execute.
- **query_vars** – Any variable definitions required by the GraphQL *query*.

Returns The query results.

Return type dict

graphql_file(*file_or_path*, *query_vars=None*)
This method wraps *graphql()* to provide a convenient way to execute GraphQL queries from files.

Parameters

- **file_or_path** – The file object or path to the file from which to read.
- **query_vars** – The variables for *query*.

Returns The query results.

Return type dict

graphql_find_file(*query_file*, ***query_vars*)
This method is similar to *graphql_file()*. The first argument (*query_file*) is the name of a query file that will be located using `find.data_file()`. Additional keyword arguments are passed as the variables to the query.

Parameters

- **query_file** (*str*) – The name of the query file to locate.
- **query_vars** – These keyword arguments are passed as the variables to the query.

Returns The query results.

Return type dict

login(*username*, *password*, *otp=None*)
Authenticate to the remote server. This is required before calling RPC methods which require an authenticated session.

Parameters

- **username** (*str*) – The username to authenticate with.
- **password** (*str*) – The password to authenticate with.
- **otp** (*str*) – An optional one time password as a 6 digit string to provide if the account requires it.

Returns The login result and an accompanying reason.

Return type tuple

ping()
Call the ping RPC method on the remote server to ensure that it is responsive. On success this method will always return True, otherwise an exception will be thrown.

Returns True

Return type bool

reconnect()
Reconnect to the remote server.

remote_row_resolve(*row*)
Take a *RemoteRow* instance and load all fields which are *UNRESOLVED*. If all fields are present, no modifications are made.

Parameters **row** – The row who's data is to be resolved.

Return type *RemoteRow*

Returns The row with all of it's fields fully resolved.

Return type *RemoteRow*

remote_table(*table*, *query_filter=None*)
Iterate over a remote database table hosted on the server. Rows are yielded as named tuples whose fields are the columns of the specified table.

Parameters **table** (*str*) – The table name to retrieve.

Returns A generator which yields rows of named tuples.

Return type tuple

remote_table_row(*table*, *row_id*, *cache=False*, *refresh=False*)
Get a row from the specified table by it's id, optionally caching it.

Parameters

- **table** (*str*) – The table in which the row exists.
- **row_id** – The value of the row's id column.
- **cache** (*bool*) – Whether to use the cache for this row.
- **refresh** (*bool*) – If *cache* is True, get the current row value and store it.

Returns The remote row as a named tuple of the specified table.

Return type tuple

remote_table_row_set(*table*, *row_id*, *attributes*)

shutdown()

class RemoteRow(*rpc*, **args*, ***kwargs*)

Bases: king_phisher.client.client_rpc._RemoteRow

A generic class representing a row of data from the remote King Phisher server.

commit()

Send this object to the server to update the remote instance.

1.1.8 export

This module provides functionality for exporting information from the client application into a variety of formats.

Functions

campaign_credentials_to_msf_txt(*rpc*, *campaign_id*, *target_file*)

Export credentials into a format that can easily be used with Metasploit's USERPASS_FILE option.

Parameters

- **rpc** (*KingPhisherRPCClient*) – The connected RPC instance to load the information with.
- **campaign_id** – The ID of the campaign to load the information for.
- **target_file** (*str*) – The destination file for the credential data.

campaign_to_xml(*rpc*, *campaign_id*, *xml_file*, *encoding='utf-8'*)

Load all information for a particular campaign and dump it to an XML file.

Parameters

- **rpc** (*KingPhisherRPCClient*) – The connected RPC instance to load the information with.
- **campaign_id** – The ID of the campaign to load the information for.
- **xml_file** (*str*) – The destination file for the XML data.
- **encoding** (*str*) – The encoding to use for strings.

campaign_visits_to_geojson(*rpc*, *campaign_id*, *geojson_file*)

Export the geo location information for all the visits of a campaign into the GeoJSON format.

Parameters

- **rpc** (*KingPhisherRPCClient*) – The connected RPC instance to load the information with.
- **campaign_id** – The ID of the campaign to load the information for.
- **geojson_file** (*str*) – The destination file for the GeoJSON data.

convert_value(*table_name*, *key*, *value*)
Perform any conversions necessary to neatly display the data in XML format.

Parameters

- **table_name** (*str*) – The table name that the key and value pair are from.
- **key** (*str*) – The data key.
- **value** – The data value to convert.

Returns The converted value.

Return type str

message_data_from_kpm(*target_file*, *dest_dir*, *encoding='utf-8'*)
Retrieve the stored details describing a message from a previously exported file.

Parameters

- **target_file** (*str*) – The file to load as a message archive.
- **dest_dir** (*str*) – The directory to extract data and attachment files to.
- **encoding** (*str*) – The encoding to use for strings.

Returns The restored details from the message config.

Return type dict

message_data_to_kpm(*message_config*, *target_file*, *encoding='utf-8'*)
Save details describing a message to the target file.

Parameters

- **message_config** (*dict*) – The message details from the client configuration.
- **target_file** (*str*) – The file to write the data to.
- **encoding** (*str*) – The encoding to use for strings.

liststore_export(*store*, *columns*, *cb_write*, *cb_write_args*, *row_offset=0*, *write_columns=True*)
A function to facilitate writing values from a list store to an arbitrary callback for exporting to different formats. The callback will be called with the row number, the column values and the additional arguments specified in **cb_write_args*.

```
cb_write(row, column_values, *cb_write_args).
```

Parameters

- **store** (*Gtk.ListStore*) – The store to export the information from.
- **columns** (*dict*) – A dictionary mapping store column ids to the value names.
- **cb_write** (*function*) – The callback function to be called for each row of data.
- **cb_write_args** (*tuple*) – Additional arguments to pass to *cb_write*.
- **row_offset** (*int*) – A modifier value to add to the row numbers passed to *cb_write*.
- **write_columns** (*bool*) – Write the column names to the export.

Returns The number of rows that were written.

Return type int

liststore_to_csv(*store*, *target_file*, *columns*)
Write the contents of a Gtk.ListStore to a csv file.

Parameters

- **store** (Gtk.ListStore) – The store to export the information from.
- **target_file** (*str*) – The destination file for the CSV data.
- **columns** (*dict*) – A dictionary mapping store column ids to the value names.

Returns The number of rows that were written.

Return type int

liststore_to_xlsx_worksheet(*store*, *worksheet*, *columns*, *title_format*, *xlsx_options=None*)
Write the contents of a Gtk.ListStore to an XLSX workseet.

Parameters

- **store** (Gtk.ListStore) – The store to export the information from.
- **worksheet** (xlsxwriter.worksheet.Worksheet) – The destination sheet for the store's data.
- **columns** (*dict*) – A dictionary mapping store column ids to the value names.
- **xlsx_options** (XLSXWorksheetOptions) – A collection of additional options for formatting the Excel Worksheet.

Returns The number of rows that were written.

Return type int

1.1.9 graphs

This module provides the functionality to support the client application's graphing capabilities.

Data

has_matplotlib = False
Whether the matplotlib module is available.

has_matplotlib_basemap = False
Whether the mpl_toolkits.basemap module is available.

Functions

export_graph_provider(*cls*)
Decorator to mark classes as valid graph providers. This decorator also sets the name attribute.

Parameters **cls** (*class*) – The class to mark as a graph provider.

Returns The *cls* parameter is returned.

get_graph(*graph_name*)
Return the graph providing class for *graph_name*. The class providing the specified graph must have been previously exported using *export_graph_provider()*.

Parameters **graph_name** (*str*) – The name of the graph provider.

Returns The graph provider class.

Return type *CampaignGraph*

get_graphs()
Get a list of all registered graph providers.

Returns All registered graph providers.

Return type list

Classes

class GraphBase(*application*, *size_request=None*, *style_context=None*)
Bases: object

A basic graph provider for using matplotlib to create graph representations of campaign data. This class is meant to be subclassed by real providers.

__init__(*application*, *size_request=None*, *style_context=None*)

Parameters **size_request** (*tuple*) – The size to set for the canvas.

config = None
A reference to the King Phisher client configuration.

get_color(*color_name*, *default*)
Get a color by its style name such as 'fg' for foreground. If the specified color does not exist, default will be returned. The underlying logic for this function is provided by *gtk_style_context_get_color()*.

Parameters

- **color_name** (*str*) – The style name of the color.
- **default** – The default color to return if the specified one was not found.

Returns The desired color if it was found.

Return type tuple

graph_title = 'Unknown'
The title that will be given to the graph.

make_window()
Create a window from the figure manager.

Returns The graph in a new, dedicated window.

Return type Gtk.Window

minimum_size = None
An absolute minimum size for the canvas.

name = 'Unknown'
The name of the graph provider.

name_human = 'Unknown'
The human readable name of the graph provider used for UI identification.

resize(*width=0*, *height=0*)
Attempt to resize the canvas. Regardless of the parameters the canvas will never be resized to be smaller than *minimum_size*.

Parameters

- **width** (*int*) – The desired width of the canvas.
- **height** (*int*) – The desired height of the canvas.

class CampaignGraph(*application*, *size_request=None*, *style_context=None*)
Bases: *king_phisher.client.graphs.GraphBase*

Graph format used for the graphs generated in the dashboard and in the create graphs tab.

load_graph()
Load the graph information via *refresh()*.

refresh(*info_cache=None*, *stop_event=None*)
Refresh the graph data by retrieving the information from the remote server.

Parameters

- **info_cache** (*dict*) – An optional cache of data tables.
- **stop_event** (threading.Event) – An optional object indicating that the operation should stop.

Returns A dictionary of cached tables from the server.

Return type dict

class CampaignGraphMessageResults(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignPieGraph

Display the percentage of messages which resulted in a visit.

graph_title = 'Campaign Message Results'

name = 'MessageResults'

name_human = 'Pie - Message Results'

class CampaignGraphOverview(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignBarGraph

Display a graph which represents an overview of the campaign.

graph_title = 'Campaign Overview'

name = 'Overview'

name_human = 'Bar - Campaign Overview'

class CampaignGraphPasswordComplexityPie(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignPieGraph

Display a graph which displays the number of passwords which meet standard complexity requirements.

graph_title = 'Campaign Password Complexity'

name = 'PasswordComplexityPie'

name_human = 'Pie - Password Complexity'

class CampaignGraphVisitorInfo(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignBarGraph

Display a graph which shows the different operating systems seen from visitors.

graph_title = 'Campaign Visitor OS Information'

name = 'VisitorInfo'

name_human = 'Bar - Visitor OS Information'

class CampaignGraphVisitorInfoPie(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignPieGraph

Display a graph which compares the different operating systems seen from visitors.

graph_title = 'Campaign Visitor OS Information'

name = 'VisitorInfoPie'

name_human = 'Pie - Visitor OS Information'

class CampaignGraphVisitsMap(*application*, *size_request=None*, *style_context=None*)
Bases: *king_phisher.client.graphs.CampaignGraph*

A base class to display a map which shows the locations of visit origins.

color_with_creds

color_without_creds

draw_states = False

graph_title = 'Campaign Visit Locations'

is_available = False

CampaignGraphVisitsMapUSA

CampaignGraphVisitsMapWorld

class CampaignGraphVisitsTimeline(**args*, ***kwargs*)
Bases: king_phisher.client.graphs.CampaignLineGraph

Display a graph which represents the visits of a campaign over time.

graph_title = 'Campaign Visits Timeline'

name = 'VisitsTimeline'

name_human = 'Line - Visits Timeline'

class CampaignGraphComparison(**args*, ***kwargs*)
Bases: *king_phisher.client.graphs.GraphBase*

Display selected campaigns data by order of campaign start date.

__init__(**args*, ***kwargs*)

Parameters **size_request** (*tuple*) – The size to set for the canvas.

load_graph(*campaigns*)
Load the information to compare the specified and paint it to the canvas. Campaigns are graphed on the X-axis in the order that they are provided. No sorting of campaigns is done by this method.

Parameters **campaigns** (*tuple*) – A tuple containing campaign IDs to compare.

1.1.10 gui_utilities

This module provides various utility functions specific to the graphical nature of the client application.

Data

GOBJECT_PROPERTY_MAP
The dictionary which maps GObjects to either the names of properties to store text or a tuple which contains a set and get function. If a tuple of two functions is specified the set function will be provided two parameters, the object and the value and the get function will just be provided the object.

Functions

delayed_signal (*delay=500*)
A decorator to delay the execution of a signal handler to aggregate emission into a single event. This can for example be used to run a handler when a `Gtk.Entry` widget's `changed` signal is emitted but not after every single key press meaning the handler can perform network operations to validate or otherwise process input.

Note: The decorated function **must** be a method. The wrapper installed by this decorator will automatically add an attribute to the class to track invoked instances to ensure the timeout is respected.

New in version 1.14.0.

Parameters **delay** (*int*) – The delay in milliseconds from the original emission before the handler should be executed.

glib_idle_add_once (*function*, **args*, ***kwargs*)
Execute *function* in the main GTK loop using `GLib.idle_add()` one time. This is useful for threads that need to update GUI data.

Parameters

- **function** (*function*) – The function to call.
- **args** – The positional arguments to *function*.
- **kwargs** – The key word arguments to *function*.

Returns The result of the function call.

glib_idle_add_wait (*function*, **args*, ***kwargs*)
Execute *function* in the main GTK loop using `GLib.idle_add()` and block until it has completed. This is useful for threads that need to update GUI data.

Parameters

- **function** (*function*) – The function to call.
- **args** – The positional arguments to *function*.
- **kwargs** – The key word arguments to *function*.

Returns The result of the function call.

glib_idle_add_store_extend (*store*, *things*, *clear=False*, *wait=False*)
Extend a GTK store object (either `Gtk.ListStore` or `Gtk.TreeStore`) object using `GLib.idle_add()`. This function is suitable for use in non-main GUI threads for synchronizing data.

Parameters

- **store** (`Gtk.ListStore`, `Gtk.TreeStore`) – The GTK storage object to add *things* to.
- **things** (*tuple*) – The array of things to add to *store*.

- **clear** (*bool*) – Whether or not to clear the storage object before adding *things* to it.
- **wait** (*bool*) – Whether or not to wait for the operation to complete before returning.

Returns Regardless of the *wait* parameter, `None` is returned.

Return type None

gobject_get_value(*gobject*, *gtype=None*)

Retrieve the value of a GObject widget. Only objects with corresponding entries present in the *GOBJECT_PROPERTY_MAP* can be processed by this function.

Parameters

- **gobject** (`GObject.Object`) – The object to retrieve the value for.
- **gtype** (*str*) – An explicit type to treat *gobject* as.

Returns The value of *gobject*.

Return type str

gobject_set_value(*gobject*, *value*, *gtype=None*)

Set the value of a GObject widget. Only objects with corresponding entries present in the *GOBJECT_PROPERTY_MAP* can be processed by this function.

Parameters

- **gobject** (`GObject.Object`) – The object to set the value for.
- **value** – The value to set for the object.
- **gtype** (*str*) – An explicit type to treat *gobject* as.

gobject_signal_accumulator(*test=None*)

Create an accumulator function for use with GObject signals. All return values will be collected and returned in a list. If provided, *test* is a callback that will be called with two arguments, the return value from the handler and the list of accumulated return values.

```
stop = test(retval, accumulated)
```

Parameters **test** – A callback to test whether additional handler should be executed.

gobject_signal_blocked(*gobject*, *signal_name*)

This is a context manager that can be used with the 'with' statement to execute a block of code while *signal_name* is blocked.

Parameters

- **gobject** (`GObject.Object`) – The object to block the signal on.
- **signal_name** (*str*) – The name of the signal to block.

gtk_calendar_get_pydate(*gtk_calendar*)

Get the Python date from a `Gtk.Calendar` instance. If the day in *gtk_calendar* is not within the valid range for the specified month, it will be rounded to the closest value (i.e. 0 for unset will become 1 etc.).

Parameters **gtk_calendar** (`Gtk.Calendar`) – The calendar to get the date from.

Returns The date as returned by the calendar's `get_date()` method.

Return type `datetime.date`

gtk_calendar_set_pydate(*gtk_calendar*, *pydate*)

Set the date on a `Gtk.Calendar` instance from a Python `datetime.date` object.

Parameters

- **gtk_calendar** (Gtk.Calendar) – The gtk_calendar to set the date for.
- **pydate** (datetime.date) – The date to set on the gtk_calendar.

gtk_combobox_set_entry_completion (*combobox*)
Add completion for a Gtk.ComboBox widget which contains an entry. They combobox's entry-text-column property it used to determine which column in its model contains the strings to suggest for completion.

New in version 1.14.0.

Parameters **combobox** – The combobox to add completion for.

Type Gtk.ComboBox

gtk_list_store_search (*list_store*, *value*, *column=0*)
Search a Gtk.ListStore for a value and return a Gtk.TreeIter to the first match.

Parameters

- **list_store** (Gtk.ListStore) – The list store to search.
- **value** – The value to search for.
- **column** (*int*) – The column in the row to check.

Returns The row on which the value was found.

Return type Gtk.TreeIter

gtk_listbox_populate_labels (*listbox*, *label_strings*)
Formats and adds labels to a listbox. Each label is styled and added as a separate entry.

New in version 1.13.0.

Parameters

- **listbox** (Gtk.listbox) – Gtk Listbox to put the labels in.
- **label_strings** (*list*) – List of strings to add to the Gtk Listbox as labels.

gtk_listbox_populate_urls (*listbox*, *url_strings*, *signals=None*)
Format and adds URLs to a list box. Each URL is styeled and added as a seperate entry.

New in version 1.14.0.

Parameters

- **listbox** (Gtk.listbox) – Gtk Listbox to put the labels in.
- **url_strings** (*list*) – List of URL strings to add to the Gtk Listbox as labels.
- **signals** (*dict*) – A dictionary, keyed by signal names to signal handler functions for the labels added to the listbox.

gtk_menu_get_item_by_label (*menu*, *label*)
Retrieve a menu item from a menu by it's label. If more than one items share the same label, only the first is returned.

Parameters

- **menu** (Gtk.Menu) – The menu to search for the item in.
- **label** (*str*) – The label to search for in *menu*.

Returns The identified menu item if it could be found, otherwise None is returned.

Return type Gtk.MenuItem

gtk_menu_insert_by_path(*menu*, *menu_path*, *menu_item*)
Add a new menu item into the existing menu at the path specified in *menu_path*.

Parameters

- **menu** (Gtk.Menu Gtk.MenuBar) – The existing menu to add the new item to.
- **menu_path** (*list*) – The labels of submenus to traverse to insert the new item.
- **menu_item** (Gtk.MenuItem) – The new menu item to insert.

gtk_menu_position(*event*, **args*)
Create a menu at the given location for an event. This function is meant to be used as the *func* parameter for the Gtk.Menu.popup() method. The *event* object must be passed in as the first parameter, which can be accomplished using functools.partial().

Parameters **event** – The event to retrieve the coordinates for.

gtk_style_context_get_color(*sc*, *color_name*, *default=None*)
Look up a color by it's name in the Gtk.StyleContext specified in *sc*, and return it as an Gdk.RGBA instance if the color is defined. If the color is not found, *default* will be returned.

Parameters

- **sc** (Gtk.StyleContext) – The style context to use.
- **color_name** (*str*) – The name of the color to lookup.
- **default** (str, Gdk.RGBA) – The default color to return if the specified color was not found.

Returns The color as an RGBA instance.

Return type Gdk.RGBA

gtk_sync()
Wait while all pending GTK events are processed.

gtk_treesortable_sort_func_numeric(*model*, *iter1*, *iter2*, *column_id*)
Sort the model by comparing text numeric values with place holders such as 1,337. This is meant to be set as a sorting function using Gtk.TreeSortable.set_sort_func(). The user_data parameter must be the column id which contains the numeric values to be sorted.

Parameters

- **model** (Gtk.TreeSortable) – The model that is being sorted.
- **iter1** (Gtk.TreeIter) – The iterator of the first item to compare.
- **iter2** (Gtk.TreeIter) – The iterator of the second item to compare.
- **column_id** – The ID of the column containing numeric values.

Returns An integer, -1 if item1 should come before item2, 0 if they are the same and 1 if item1 should come after item2.

Return type int

gtk_treeview_get_column_titles(*treeview*)
Iterate over a GTK TreeView and return a tuple containing the id and title of each of it's columns.

Parameters **treeview** (Gtk.TreeView) – The treeview instance to retrieve columns from.

gtk_treeview_selection_to_clipboard(*treeview*, *columns=0*)
Copy the currently selected values from the specified columns in the treeview to the users clipboard. If no value is selected in the treeview, then the clipboard is left unmodified. If multiple values are selected, they will all be placed in the clipboard on separate lines.

Parameters

- **treeview** (Gtk.TreeView) – The treeview instance to get the selection from.
- **column** (*int, list, tuple*) – The column numbers to retrieve the value for.

gtk_treeview_selection_iterate(*treeview*)
Iterate over the a treeview's selected rows.

Parameters **treeview** (Gtk.TreeView) – The treeview for which to iterate over.

Returns The rows which are selected within the treeview.

Return type Gtk.TreeIter

gtk_treeview_set_column_titles(*treeview*, *column_titles*, *column_offset=0*, *renderers=None*)
Populate the column names of a GTK TreeView and set their sort IDs.

Parameters

- **treeview** (Gtk.TreeView) – The treeview to set column names for.
- **column_titles** (*list*) – The names of the columns.
- **column_offset** (*int*) – The offset to start setting column names at.
- **renderers** (*list*) – A list containing custom renderers to use for each column.

Returns A dict of all the Gtk.TreeViewColumn objects keyed by their column id.

Return type dict

gtk_widget_destroy_children(*widget*)
Destroy all GTK child objects of *widget*.

Parameters **widget** (Gtk.Widget) – The widget to destroy all the children of.

show_dialog(*message_type*, *message*, *parent*, *secondary_text=None*, *message_buttons=<king_phisher.utilities.Mock object>*, *use_markup=False*, *secondary_use_markup=False*)
Display a dialog and return the response. The response is dependent on the value of *message_buttons*.

Parameters

- **message_type** (Gtk.MessageType) – The GTK message type to display.
- **message** (*str*) – The text to display in the dialog.
- **parent** (Gtk.Window) – The parent window that the dialog should belong to.
- **secondary_text** (*str*) – Optional subtext for the dialog.
- **message_buttons** (Gtk.ButtonsType) – The buttons to display in the dialog box.
- **use_markup** (*bool*) – Whether or not to treat the message text as markup.
- **secondary_use_markup** (*bool*) – Whether or not to treat the secondary text as markup.

Returns The response of the dialog.

Return type int

show_dialog_exc_socket_error(*error*, *parent*, *title=None*)
Display an error dialog with details regarding a socket.error exception that has been raised.

Parameters

- **error** (socket.error) – The exception instance that has been raised.
- **parent** (Gtk.Window) – The parent window that the dialog should belong to.
- **title** – The title of the error dialog that is displayed.

show_dialog_error(**args*, ***kwargs*)
Display an error dialog with *show_dialog()*.

show_dialog_info(**args*, ***kwargs*)
Display an informational dialog with *show_dialog()*.

show_dialog_warning(**args*, ***kwargs*)
Display an warning dialog with *show_dialog()*.

show_dialog_yes_no(**args*, ***kwargs*)
Display a dialog which asks a yes or no question with *show_dialog()*.

Returns True if the response is Yes.

Return type bool

which_glade()
Locate the glade data file which stores the UI information in a Gtk Builder format.

Returns The path to the glade data file.

Return type str

Classes

class FileMonitor(*path*, *on_changed*)
Bases: object

Monitor a file for changes.

__init__(*path*, *on_changed*)

Parameters

- **path** (*str*) – The path to monitor for changes.
- **on_changed** (*function*) – The callback function to be called when changes are detected.

class GladeDependencies(*children=None*, *top_level=None*, *name=None*)
Bases: object

A class for defining how objects should be loaded from a GTK Builder data file for use with *GladeGObject*.

__init__(*children=None*, *top_level=None*, *name=None*)
Initialize self. See help(type(self)) for accurate signature.

children
A tuple of string names or *GladeProxy* instances listing the children widgets to load from the parent.

name
The string of the name of the top level parent widget to load.

top_level
A tuple of string names listing additional top level widgets to load such as images.

class GladeGObjectMeta(**args*, ***kwargs*)
Bases: type

A meta class that will update the *GladeDependencies.name* value in the *GladeGObject.dependencies* attribute of instances if no value is defined.

class assigned_name
Bases: str

A type subclassed from str that is used to define names which have been automatically assigned by this class.

class GladeGObject(*application*)
Bases: king_phisher.client.gui_utilities._GladeGObject

A base object to wrap GTK widgets loaded from Glade data files. This provides a number of convenience methods for managing the main widget and child widgets. This class is meant to be subclassed by classes representing objects from the Glade data file.

__init__(*application*)

Parameters **application** (Gtk.Application) – The parent application for this object.

application = None
The parent Gtk.Application instance.

config = None
A reference to the King Phisher client configuration.

config_prefix = ''
A prefix to be used for keys when looking up value in the *config*.

dependencies = <GladeDependencies name='GladeGObject' >
A *GladeDependencies* instance which defines information for loading the widget from the GTK builder data.

destroy()
Call destroy() on the top-level GTK Widget.

get_entry_value(*entry_name*)
Get the value of the specified entry then remove leading and trailing white space and finally determine if the string is empty, in which case return None.

Parameters **entry_name** (*str*) – The name of the entry to retrieve text from.

Returns Either the non-empty string or None.

Return type None, str

gobjects = None
A *FreezableDict* which maps gobjects to their unique GTK Builder id.

gtk_builder = None
A Gtk.Builder instance used to load Glade data with.

gtk_builder_get(*gobject_id*, *parent_name=None*)
Find the child GObject with name *gobject_id* from the GTK builder.

Parameters

- **gobject_id** (*str*) – The object name to look for.

- **parent_name** (*str*) – The name of the parent object in the builder data file.

Returns The GObject as found by the GTK builder.

Return type GObject.Object

hide()
Call hide() on the top-level GTK Widget.

objects_load_from_config()
Iterate through *gobjects* and set the GObject's value from the corresponding value in the *config*.

objects_persist = True
Whether objects should be automatically loaded from and saved to the configuration.

objects_save_to_config()

parent

show()
Call show() on the top-level GTK Widget.

show_all()
Call show_all() on the top-level GTK Widget.

top_gobject = 'gobject'
The name of the attribute to set a reference of the top level GObject to.

class GladeProxy(*destination*)
Bases: object

A class that can be used to load another top level widget from the GTK builder data file in place of a child. This is useful for reusing small widgets as children in larger ones.

__init__(*destination*)
Initialize self. See help(type(self)) for accurate signature.

children = ()
A tuple of string names or *GladeProxy* instances listing the children widgets to load from the top level.

destination
A *GladeProxyDestination* instance describing how this proxied widget should be added to the parent.

name = None
The string of the name of the top level widget to load.

class GladeProxyDestination(*method*, *widget=None*, *args=None*, *kwargs=None*)
Bases: object

A class that is used to define how a *GladeProxy* object shall be loaded into a parent *GladeGObject* instance. This includes the information such as what container widget in the parent the proxied widget should be added to and what method should be used. The proxied widget will be added to the parent by calling *method* with the proxied widget as the first argument.

__init__(*method*, *widget=None*, *args=None*, *kwargs=None*)

Parameters

- **method** (*str*) – The method of the container *widget* to use to add the proxied widget.
- **widget** (*str*) – The widget name to add the proxied widget to. If this value is None, the proxied widget is added to the top level widget.
- **args** (*tuple*) – Position arguments to provide when calling *method*.

- **kwargs** (*dict*) – Key word arguments to provide when calling *method*.

args
Arguments to append after the proxied child instance when calling *method*.

kwargs
Key word arguments to append after the proxied child instance when calling *method*.

method
The method of the parent widget that should be called to add the proxied child.

widget
The name of the parent widget for this proxied child.

1.1.11 mailer

This module provides the functionality used to create and sending messages from the client application.

Data

MIME_TEXT_PLAIN = 'This message requires an HTML aware email agent to be properly viewed.\r
The static string to place in MIME message as a text/plain part. This is shown by email clients that do not support HTML.

Functions

count_targets_file(*target_file*)
Count the number of valid targets that the specified file contains. This skips lines which are missing fields or where the email address is invalid.

Parameters **target_file** (*str*) – The path the the target CSV file on disk.

Returns The number of valid targets.

Return type int

get_invite_start_from_config(*config*)
Get the start time for an invite from the configuration. This takes into account whether the invite is for all day or starts at a specific time.

Parameters **config** (*dict*) – The King Phisher client configuration.

Returns The timestamp of when the invite is to start.

Return type datetime.datetime

guess_smtp_server_address(*host*, *forward_host=None*)
Guess the IP address of the SMTP server that will be connected to given the SMTP host information and an optional SSH forwarding host. If a hostname is in use it will be resolved to an IP address, either IPv4 or IPv6 and in that order. If a hostname resolves to multiple IP addresses, None will be returned. This function is intended to guess the SMTP servers IP address given the client configuration so it can be used for SPF record checks.

Parameters

- **host** (*str*) – The SMTP server that is being connected to.
- **forward_host** (*str*) – An optional host that is being used to tunnel the connection.

Returns The IP address of the SMTP server.

Return type None, ipaddress.IPv4Address, ipaddress.IPv6Address

render_message_template(*template*, *config*, *target=None*, *analyze=False*)

Take a message from a template and format it to be sent by replacing variables and processing other template directives. If the *target* parameter is not set, a placeholder will be created and the message will be formatted to be previewed.

Parameters

- **template** (*str*) – The message template.
- **config** (*dict*) – The King Phisher client configuration.
- **target** (*MessageTarget*) – The messages intended target information.
- **analyze** (*bool*) – Set the template environment to analyze mode.

Returns The formatted message.

Return type str

rfc2282_timestamp(*dt=None*, *utc=False*)

Convert a datetime.datetime instance into an **RFC 2282** compliant timestamp suitable for use in MIME-encoded messages.

Parameters

- **dt** (datetime.datetime) – A time to use for the timestamp otherwise the current time is used.
- **utc** – Whether to return the timestamp as a UTC offset or from the local timezone.

Returns The timestamp.

Return type str

Classes

class MailSenderThread(*application*, *target_file*, *rpc*, *tab=None*)

Bases: threading.Thread

The King Phisher threaded email message sender. This object manages the sending of emails for campaigns and supports pausing the sending of messages which can later be resumed by unpausing. This object reports its information to the GUI through an optional *MailSenderSendTab* instance, these two objects are very interdependent.

__init__(*application*, *target_file*, *rpc*, *tab=None*)

Parameters

- **application** (*KingPhisherClientApplication*) – The GTK application that the thread is associated with.
- **target_file** (*str*) – The CSV formatted file to read message targets from.
- **tab** (*MailSenderSendTab*) – The GUI tab to report information to.
- **rpc** (*KingPhisherRPCClient*) – The client's connected RPC instance.

count_targets()

Count the number of targets that will be sent messages.

Returns The number of targets that will be sent messages.

Return type int

create_message_calendar_invite(*target*, *attachments*)
Create a MIME calendar invite to be sent from a set of parameters.

Parameters

- **target** (*MessageTarget*) – The information for the messages intended recipient.
- **uid** (*str*) – The message's unique identifier.
- **attachments** (Attachments) – The attachments to add to the created message.

Returns The new MIME message.

Return type email.mime.multipart.MIMEMultipart

create_message_email(*target*, *attachments*)
Create a MIME email to be sent from a set of parameters.

Parameters

- **target** (*MessageTarget*) – The information for the messages intended recipient.
- **uid** (*str*) – The message's unique identifier.
- **attachments** (*MessageAttachments*) – The attachments to add to the created message.

Returns The new MIME message.

Return type email.mime.multipart.MIMEMultipart

get_mime_attachments()
Return a *MessageAttachments* object containing both the images and raw files to be included in sent messages.

Returns A namedtuple of both files and images in their MIME containers.

Return type *MessageAttachments*

iterate_targets(*counting=False*)
Iterate over each of the targets as defined within the configuration. If *counting* is False, messages will not be displayed to the end user through the notification tab.

Parameters **counting** (*bool*) – Whether or not to iterate strictly for counting purposes.

Returns Each message target.

Return type *MessageTarget*

missing_files()
Return a list of all missing or unreadable files which are referenced by the message template.

Returns The list of unusable files.

Return type list

pause()
Sets the *running* and *paused* flags correctly to indicate that the object is paused.

paused = None
A threading.Event object indicating if the email sending operation is or should be paused.

process_pause(*set_pause=False*)
Pause sending emails if a pause request has been set.

Parameters **set_pause** (*bool*) – Whether to request a pause before processing it.

Returns Whether or not the sending operation was cancelled during the pause.

Return type bool

run()
The entry point of the thread.

running = None
A threading.Event object indicating if emails are being sent.

send_message(*target_email*, *msg*)
Send an email using the connected SMTP server.

Parameters

- **target_email** (*str*) – The email address to send the message to.
- **msg** (mime.multipart.MIMEMultipart) – The formatted message to be sent.

server_smtp_connect()
Connect and optionally authenticate to the configured SMTP server.

Returns The connection status as one of the *ConnectionErrorReason* constants.

server_smtp_disconnect()
Clean up and close the connection to the remote SMTP server.

server_smtp_reconnect()
Disconnect from the remote SMTP server and then attempt to open a new connection to it.

Returns The reconnection status.

Return type bool

server_ssh_connect()
Connect to the remote SMTP server over SSH and configure port forwarding with *SSHTCPForwarder* for tunneling SMTP traffic.

Returns The connection status as one of the *ConnectionErrorReason* constants.

smtp_connection = None
The smtplib.SMTP connection instance.

stop()
Requests that the email sending operation stop. It can not be resumed from the same position. This function blocks until the stop request has been processed and the thread exits.

tab = None
The optional *MailSenderSendTab* instance for reporting status messages to the GUI.

tab_notify_sent(*emails_done*, *emails_total*)
Notify the tab that messages have been sent.

Parameters

- **emails_done** (*int*) – The number of emails that have been sent.
- **emails_total** (*int*) – The total number of emails that are going to be sent.

tab_notify_status(*message*)
Handle a status message regarding the message sending operation.

Parameters **message** (*str*) – The notification message.

tab_notify_stopped()
Notify the tab that the message sending operation has stopped.

target_file = None
The name of the target file in CSV format.

unpause()
Sets the *running* and *paused* flags correctly to indicate that the object is no longer paused.

class MessageAttachments(*files*, *images*)
A named tuple for holding both image and file attachments for a message.

files
A tuple of `MIMEBase` instances representing the messsages attachments.

images
A tuple of `MIMEImage` instances representing the images in the message.

class MessageTarget(*first_name*, *last_name*, *email_address*, *uid=None*, *department=None*, *line=None*)
Bases: `object`

A simple class for holding information regarding a messages intended recipient.

__init__(*first_name*, *last_name*, *email_address*, *uid=None*, *department=None*, *line=None*)
Initialize self. See help(type(self)) for accurate signature.

department
The target recipient's department name.

email_address
The target recipient's email address.

first_name
The target recipient's first name.

last_name
The target recipient's last name.

line
The line number in the file from which this target was loaded.

uid
The unique identifier that is going to be used for this target.

class MessageTargetPlaceholder(*uid=None*)
Bases: *king_phisher.client.mailer.MessageTarget*

A default *MessageTarget* for use as a placeholder value while rendering, performing tests, etc.

__init__(*uid=None*)
Initialize self. See help(type(self)) for accurate signature.

class TopMIMEMultipart(*mime_type*, *config*, *target*)
Bases: `email.mime.multipart.MIMEMultipart`

A `mime.multipart.MIMEMultipart` subclass for representing the top / outer most part of a MIME multipart message. This adds additional default headers to the message.

__init__(*mime_type*, *config*, *target*)

Parameters

- **mime_type** (*str*) – The type of this part such as related or alternative.
- **config** (*dict*) – The King Phisher client configuration.
- **target** (*MessageTarget*) – The target information for the messages intended recipient.

1.1.12 plugins

Classes

class CatalogCacheManager(*cache_file*)
Bases: object

Manager to handle cache information for catalogs. Cache entries are stored as dictionaries with metadata information and the catalog data under the "value" key.

__init__(*cache_file*)
Initialize self. See help(type(self)) for accurate signature.

get_catalog_by_id(*catalog_id*)
Return the catalog cache data for the specified catalog ID.

Parameters **catalog_id** (*str*) – The ID of the catalog to look up in the cache.

Returns The cache entry for the catalog or None if the catalog was not found.

Return type dict

get_catalog_by_url(*catalog_url*)
Return the catalog cache data for the specified catalog URL.

Parameters **catalog_url** (*str*) – The URL of the catalog to look up in the cache.

Returns The cache entry for the catalog or None if the catalog was not found.

Return type dict

class ClientCatalogManager(*user_data_path*, *manager_type='plugins/client'*, **args*, ***kwargs*)
Bases: *king_phisher.catalog.CatalogManager*

Base manager for handling Catalogs.

__init__(*user_data_path*, *manager_type='plugins/client'*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

add_catalog(*catalog*, *catalog_url*, *cache=False*)
Adds the specified catalog to the manager and stores the associated source URL for caching.

Parameters

- **catalog** (*Catalog*) – The catalog to add to the cache manager.
- **catalog_url** (*str*) – The URL from which the catalog was loaded.
- **cache** (*bool*) – Whether or not the catalog should be saved to the cache.

Returns The catalog.

Return type *Catalog*

catalog_ids()
The key names of the catalogs in the manager.

Returns The catalogs IDs in the manager instance.

Return type tuple

compatibility(*catalog_id*, *repo_id*, *plugin_name*)
Checks the compatibility of a plugin.

Parameters

- **catalog_id** (*str*) – The catalog id associated with the plugin.

- **repo_id** (*str*) – The repository id associated with the plugin.
- **plugin_name** (*str*) – The name of the plugin.

Returns Tuple of packages and if the requirements are met.

Return type tuple

get_cache()
Returns the catalog cache.

Returns The catalog cache.

Return type *CatalogCacheManager*

get_collection(*catalog_id*, *repo_id*)
Returns the manager type of the plugin collection of the requested catalog and repository.

Parameters

- **catalog_id** (*str*) – The name of the catalog the repo belongs to
- **repo_id** (*str*) – The id of the repository requested.

Returns The the collection of manager type from the specified catalog and repository.

install_plugin(*catalog_id*, *repo_id*, *plugin_id*, *install_path*)
Installs the specified plugin to the desired plugin path.

Parameters

- **catalog_id** (*str*) – The id of the catalog of the desired plugin to install.
- **repo_id** (*str*) – The id of the repository of the desired plugin to install.
- **plugin_id** (*str*) – The id of the plugin to install.
- **install_path** (*str*) – The path to install the plugin too.

is_compatible(*catalog_id*, *repo_id*, *plugin_name*)
Checks the compatibility of a plugin.

Parameters

- **catalog_id** – The catalog id associated with the plugin.
- **repo_id** – The repository id associated with the plugin.
- **plugin_name** – The name of the plugin.

Returns Whether or not it is compatible.

Return type bool

save_cache(*catalog*, *catalog_url*)
Saves the catalog or catalogs in the manager to the cache.

Parameters **catalog** – The *Catalog* to save.

class ClientOptionBoolean(*name*, **args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionMixin*, *king_phisher.plugins.OptionBoolean*

__init__(*name*, **args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.

- **description** (*str*) – The description of this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.

get_widget(*_*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*widget*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionEnum(*name*, **args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionMixin*, *king_phisher.plugins.OptionEnum*

__init__(*name*, **args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **choices** (*tuple*) – The supported values for this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option

get_widget(*_*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*widget*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionInteger(*name*, **args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionMixin*, *king_phisher.plugins.OptionInteger*

__init__(*name*, **args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.
- **adjustment** (Gtk.Adjustment) – The adjustment details of the options value.

get_widget(*_*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*_*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionMixin(*name*, **args*, ***kwargs*)
Bases: object

A mixin for options used by plugins for the client application. It provides additional methods for creating GTK widgets for the user to set the option's value as well as retrieve it.

__init__(*name*, **args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.

- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.

get_widget(*application*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*widget*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionPath(*name*, **args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionString*

__init__(*name*, **args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.
- **path_type** (*str*) – The type of the path to select, either 'directory', 'file-open' or 'file-save'.

get_widget(*application*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*widget*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionPort(**args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionInteger*

__init__(**args*, ***kwargs*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.

get_widget(*_*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*_*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientOptionString(*name*, **args*, ***kwargs*)
Bases: *king_phisher.client.plugins.ClientOptionMixin*, *king_phisher.plugins.OptionString*

__init__(*name*, **args*, ***kwargs*)
Changed in version 1.9.0b5: Added the *multiline* parameter.

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.
- **display_name** (*str*) – The name to display in the UI to the user for this option.
- **multiline** (*bool*) – Whether or not this option allows multiple lines of input.

get_widget(*_*, *value*)
Create a widget suitable for configuring this option. This is meant to allow subclasses to specify and create an appropriate widget type.

Parameters

- **application** (Gtk.Application) – The parent application for this object.
- **value** – The initial value to set for this widget.

Returns The widget for the user to set the option with.

Return type Gtk.Widget

get_widget_value(*widget*)
Get the value of a widget previously created with *get_widget()*.

Parameters **widget** (Gtk.Widget) – The widget from which to retrieve the value from for this option.

Returns The value for this option as set in the widget.

set_widget_value(*widget*, *value*)
Set the value of a widget.

Parameters **widget** (Gtk.Widget) – The widget whose value is to set for this option.

class ClientPlugin(*application*)
Bases: *king_phisher.plugins.PluginBase*

The base object to be inherited by plugins that are loaded into the King Phisher client. This provides a convenient interface for interacting with the runtime.

add_menu_item(*menu_path*, *handler=None*)
Add a new item into the main menu bar of the application. Menu items created through this method are automatically removed when the plugin is disabled. If no *handler* is specified, the menu item will be a separator, otherwise *handler* will automatically be connected to the menu item's `activate` signal.

Parameters

- **menu_path** (*str*) – The path to the menu item, delimited with > characters.
- **handler** – The optional callback function to be connected to the new Gtk.MenuItem instance's activate signal.

Returns The newly created and added menu item.

Return type Gtk.MenuItem

add_submenu(*menu_path*)
Add a submenu into the main menu bar of the application. Submenus created through this method are automatically removed when the plugin is disabled.

Parameters **menu_path** (*str*) – The path to the submenu, delimited with > characters.

Returns The newly created and added menu item.

Return type Gtk.MenuItem

application = None
A reference to the *KingPhisherClientApplication*.

config
A dictionary that can be used by this plugin for persistent storage of it's configuration.

render_template_string(*template_string*, *target=None*, *description='string'*, *log_to_mailer=True*)

Render the specified *template_string* in the message environment. If an error occurs during the rendering process, a message will be logged and `None` will be returned. If *log_to_mailer* is set to `True` then a message will also be displayed in the message send tab of the client.

New in version 1.9.0b5.

Parameters

- **template_string** (*str*) – The string to render as a template.
- **target** (*MessageTarget*) – An optional target to pass to the rendering environment.
- **description** (*str*) – A keyword to use to identify the template string in error messages.
- **log_to_mailer** (*bool*) – Whether or not to log to the message send tab as well.

Returns The rendered string or `None` if an error occurred.

Return type str

signal_connect(*name*, *handler*, *gobject=None*, *after=False*)

Connect *handler* to a signal by *name* to an arbitrary GObject. Signals connected through this method are automatically cleaned up when the plugin is disabled. If no GObject is specified, the *application* instance is used.

> **Warning:** If the signal needs to be disconnected manually by the plugin, this method should not be used. Instead the handler id should be kept as returned by the GObject's native connect method.

Parameters

- **name** (*str*) – The name of the signal.
- **handler** (*function*) – The function to be invoked with the signal is emitted.
- **gobject** – The object to connect the signal to.
- **after** (*bool*) – Whether to call the user specified handler after the default signal handler or before.

signal_connect_server_event(*name*, *handler*, *event_types*, *attributes*)

Connect *handler* to the server signal with *name*. This method is similar to *signal_connect()* but also sets up the necessary event subscriptions to ensure that the handler will be called. These event subscriptions are automatically cleaned up when the plugin is disabled.

> **Warning:** Server events are emitted based on the client applications event subscriptions. This means that while *handler* will be called for the event types specified, it may also be called for additional unspecified event types if other plugins have subscribed to them. This means that it is important to check the event type within the handler itself and react as necessary. To avoid this simply use the *event_type_filter()* decorator for the *handler* function.

Parameters

- **name** (*str*) – The name of the signal.
- **handler** – The function to be invoked with the signal is emitted.

- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

class ClientPluginMailerAttachment(**args*, ***kwargs*)

Bases: *king_phisher.client.plugins.ClientPlugin*

The base object to be inherited by plugins that intend to modify attachment files such as for inserting the tracking URL into them. Plugins which inherit from this base class must override the *process_attachment_file()* method which will automatically be called for each target a user is sending messages to.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

process_attachment_file(*input_path*, *output_path*, *target*)

This function is automatically called for each target that a user is sending messages to. This method is intended to process the specified attachment file. This method removes the need to manually cleanup the *output_path* because it is handled automatically as necessary.

Parameters

- **input_path** (*str*) – The path to the input file to process. This path is guaranteed to be an existing file that is readable.
- **output_path** (*str*) – The path to optionally write the output file to. This path may or may not be the same as *input_path*. If the plugin needs to rename the file, to for example change the extension, then the new output_path must be returned.
- **target** (*MessageTarget*) – The target information for the messages intended recipient.

Returns None or an updated value for *output_path* in the case that the plugin renames it.

class ClientPluginManager(*path*, *application*)

Bases: *king_phisher.plugins.PluginManagerBase*

The manager for plugins loaded into the King Phisher client application.

1.1.13 server_events

This module provides functionality to allow the client application to subscribe to events which are published by the server.

Functions

event_type_filter(*event_types*, *is_method=False*)

A decorator to filter a signal handler by the specified event types. Using this will ensure that the decorated function is only called for the specified event types and not others which may have been subscribed to elsewhere in the application.

Parameters

- **event_types** (*list, str*) – A single event type as a string or a list of event type strings.
- **is_method** (*bool*) – Whether or not the function being decorated is a class method.

Classes

class ServerEventSubscriber(*rpc*)
Bases: GObject.GObject

An object which provides functionality to subscribe to events that are published by the remote King Phisher server instance. This object manages the subscriptions and forwards the events allowing consumers to connect to the available GObject signals.

Note: Both the *ServerEventSubscriber.subscribe()* and *ServerEventSubscriber.unsubscribe()* methods of this object internally implement reference counting for the server events. This makes it possible for multiple subscriptions to be created and deleted without interfering with each other.

The socket is opened automatically when this object is initialized and will automatically attempt to reconnect if the connection is closed if the *reconnect* attribute is true. After initializing this object, check the *is_connected* attribute to ensure that it is properly connected to the server.

__init__(*rpc*)

Parameters **rpc** (*KingPhisherRPCClient*) – The client's connected RPC instance.

is_connected
True if the event socket is connected to the server.

is_subscribed(*event_id*, *event_type*)
Check if the client is currently subscribed to the specified server event.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_type** (*str*) – A sub-type for the corresponding event.

Returns Whether or not the client is subscribed to the event.

Return type bool

reconnect = None
Whether or not the socket should attempt to reconnect itself when it has been closed.

shutdown()
Disconnect the event socket from the remote server. After the object is shutdown, remove events will no longer be published.

Parameters **timeout** (*int*) – An optional timeout for how long to wait on the worker thread.

subscribe(*event_id*, *event_types*, *attributes*)
Subscribe the client to the specified event published by the server. When the event is published the specified *attributes* of it and it's corresponding id and type information will be sent to the client.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

unsubscribe(*event_id*, *event_types*, *attributes*)
Unsubscribe from an event published by the server that the client previously subscribed to.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

1.1.14 web_cloner

This module contains the functionality used by the client to clone web pages.

Classes

class ClonedResourceDetails
A named tuple which contains details regard a resource that has been cloned.

resource
The web resource that has been cloned.

mime_type
The MIME type that was provided by the server for the cloned resource.

size
The size of the original resource that was provided by the server.

file_name
The path to the file which the resource was written to.

class WebPageCloner(*target_url*, *dest_dir*)
Bases: object

This object is used to clone web pages. It will use the WebKit2GTK+ engine and hook signals to detect what remote resources that are loaded from the target URL. These resources are then written to disk. Resources that have a MIME type of text/html have the King Phisher server javascript file patched in..

__init__(*target_url*, *dest_dir*)

Parameters

- **target_url** (*str*) – The URL of the target web page to clone.
- **dest_dir** (*str*) – The path of a directory to write the resources to.

cloned_resources = None
A collections.OrderedDict instance of *ClonedResourceDetails* keyed by the web resource they describe.

copy_resource_data(*resource*, *data*)
Copy the data from a loaded resource to a local file.

Parameters

- **resource** (WebKit2.WebResource) – The resource whose data is being copied.
- **data** (*bytes*, *str*) – The raw data of the represented resource.

patch_html(*data*, *encoding='utf-8'*)
Patch the HTML data to include the King Phisher javascript resource. The script tag is inserted just before the closing head tag. If no head tag is present, the data is left unmodified.

Parameters **data** (*str*) – The HTML data to patch.

Returns The patched HTML data.

Return type str

resource_is_on_target(*resource*)
Test whether the resource is on the target system. This tries to match the hostname, scheme and port number of the resource's URI against the target URI.

Returns Whether the resource is on the target or not.

Return type bool

stop_cloning()
Stop the current cloning operation if it is running.

wait()
Wait for the cloning operation to complete and return whether the operation was successful or not.

Returns True if the operation was successful.

Return type bool

1.2 server

This package contains all packages and modules specific to the server application.

1.2.1 database

manager

This module provides the functionality to manage the server application's database connection.

Functions

clear_database()
Delete all data from all tables in the connected database. The database schema will remain unaffected.

> **Warning:** This action can not be reversed and there is no confirmation before it takes place.

export_database(*target_file*)
Export the contents of the database using SQLAlchemy's serialization. This creates an archive file containing all of the tables and their data. The resulting export can be imported into another supported database so long as the *SCHEMA_VERSION* is the same.

Parameters **target_file** (*str*) – The file to write the export to.

import_database(*target_file*, *clear=True*)
Import the contents of a serialized database from an archive previously created with the *export_database()* function. The current *SCHEMA_VERSION* must be the same as the exported archive.

> **Warning:** This will by default delete the contents of the current database in accordance with the *clear* parameter. If *clear* is not specified and objects in the database and import share an ID, they will be merged.

Parameters

- **target_file** (*str*) – The database archive file to import from.
- **clear** (*bool*) – Whether or not to delete the contents of the existing database before importing the new data.

normalize_connection_url(*connection_url*)

Normalize a connection url by performing any conversions necessary for it to be used with the database API.

Parameters **connection_url** (*str*) – The connection url to normalize.

Returns The normalized connection url.

Return type str

get_metadata(*key*, *session=None*)

Store a piece of metadata regarding the King Phisher database.

Parameters

- **key** (*str*) – The name of the data.
- **value** (*int, str*) – The value to store.
- **session** – The session to use to store the value.

get_row_by_id(*session*, *table*, *row_id*)

Retrieve a database row from the specified table by it's unique id.

Parameters

- **session** (*.Session*) – The database session to use for the query.
- **table** – The table object or the name of the database table where the row resides.
- **row_id** – The id of the row to retrieve.

Returns The object representing the specified row or None if it does not exist.

init_alembic(*engine*, *schema_version*)

Creates the alembic_version table and sets the value of the table according to the specified schema version.

Parameters

- **engine** (sqlalchemy.engine.Engine) – The engine used to connect to the database.
- **schema_version** (*int*) – The MetaData schema_version to set the alembic version to.

init_database(*connection_url*, *extra_init=False*)

Create and initialize the database engine. This must be done before the session object can be used. This will also attempt to perform any updates to the database schema if the backend supports such operations.

Parameters

- **connection_url** (*str*) – The url for the database connection.
- **extra_init** (*bool*) – Run optional extra dbms-specific initialization logic.

Returns The initialized database engine.

init_database_postgresql(*connection_url*)

Perform additional initialization checks and operations for a PostgreSQL database. If the database is hosted locally this will ensure that the service is currently running and start it if it is not. Additionally if the specified database or user do not exist, they will be created.

Parameters **connection_url** (sqlalchemy.engine.url.URL) – The url for the PostgreSQL database connection.

Returns The initialized database engine.

set_metadata(*key*, *value*, *session=None*)
Store a piece of metadata regarding the King Phisher database.

Parameters

- **key** (*str*) – The name of the data.
- **value** (*int, str*) – The value to store.
- **session** – The session to use to store the value.

models

This module provides the models for the data stored in the database as well as functionality for defining and managing the models themselves.

Data

database_tables
A dictionary which contains all the database tables and their *MetaTable* instances.

SCHEMA_VERSION
The schema version of the database, used for compatibility checks.

Functions

current_timestamp(**args*, ***kwargs*)
The function used for creating the timestamp used by database objects.

Returns The current timestamp.

Return type datetime.datetime

get_tables_with_column_id(*column_id*)
Get all tables which contain a column named *column_id*.

Parameters **column_id** (*str*) – The column name to get all the tables of.

Returns The list of matching tables.

Return type set

register_table(*table*)
Register a database table. This will populate the information provided in DATABASE_TABLES dictionary. This also forwards signals to the appropriate listeners within the server.signal module.

Parameters **table** (*cls*) – The table to register.

sql_null()
Return a constant Null construct.

Classes

class BaseRowCls
Bases: object

The base class from which other database table objects inherit from. Provides a standard __repr__ method and default permission checks which are to be overridden as desired by subclasses.

> **Warning:** Subclasses should not directly override the session_has_*_access methods. These contain wrapping logic to do things like checking if the session is an administrator, etc. Instead subclasses looking to control access should override the individual private variants _session_has_*_access. Each of these use the same call signature as their public counterparts.

assert_session_has_permissions(**args*, ***kwargs*)
A convenience function which wraps *session_has_permissions()* and raises a *KingPhisherPermissionError* if the session does not have the specified permissions.

is_private = False
Whether the table is only allowed to be accessed by the server or not.

classmethod metatable()
Generate a *MetaTable* instance for this model class.

Returns The appropriate metadata for the table represented by this model.

Return type *MetaTable*

classmethod session_has_create_access(*session*, *instance=None*)
Check that the authenticated *session* has access to create the specified model *instance*.

Parameters

- **session** – The authenticated session to check access for.
- **instance** – The optional model instance to inspect.

Returns Whether the session has the desired permissions.

Return type bool

classmethod session_has_delete_access(*session*, *instance=None*)
Check that the authenticated *session* has access to delete the specified model *instance*.

Parameters

- **session** – The authenticated session to check access for.
- **instance** – The optional model instance to inspect.

Returns Whether the session has the desired permissions.

Return type bool

session_has_permissions(*access*, *session*)
Check that the authenticated session has the permissions specified in *access*. The permissions in *access* are abbreviated with the first letter of create, read, update, and delete. For example, to check for read and update permissions, *access* would be 'ru'.

Note: This will always return True for sessions which are for administrative users. To maintain this logic, this method **should not** be overridden in subclasses. Instead override the specific

`_session_has_*_access` methods as necessary.

Parameters

- **access** (*str*) – The desired permissions.
- **session** – The authenticated session to check access for.

Returns Whether the session has the desired permissions.

Return type bool

classmethod session_has_read_access(*session*, *instance=None*)
Check that the authenticated *session* has access to read the specified model *instance*.

Parameters

- **session** – The authenticated session to check access for.
- **instance** – The optional model instance to inspect.

Returns Whether the session has the desired permissions.

Return type bool

classmethod session_has_read_prop_access(*session*, *prop*, *instance=None*)
Check that the authenticated *session* has access to read the property of the specified model *instance*. This allows models to only explicitly control which of their attributes can be read by a particular *session*.

Parameters

- **session** – The authenticated session to check access for.
- **instance** – The optional model instance to inspect.

Returns Whether the session has the desired permissions.

Return type bool

classmethod session_has_update_access(*session*, *instance=None*)
Check that the authenticated *session* has access to update the specified model *instance*.

Parameters

- **session** – The authenticated session to check access for.
- **instance** – The optional model instance to inspect.

Returns Whether the session has the desired permissions.

Return type bool

class MetaTable(*column_names*, *model*, *name*, *table*)
Bases: tuple

Metadata describing a table and its various attributes.

column_names
A tuple of strings representing the table's column names.

model
The SQLAlchemy model class associated with this table.

name
The name of this table.

storage

This module provides functionality to utilize the database for persistent storage.

Classes

class KeyValueStorage(*namespace=None*, *order_by='created'*)

This class provides key-value storage of arbitrary data in the database. The *serializers* module is used for converting data into a format suitable for storing in the database. This object, once initialized, provides an interface just like a standard dictionary object. An optional namespace should be specified as a unique identifier, allowing different sources to store data using the same keys. All keys must be strings but data can be anything that is serializable.

__init__(*namespace=None*, *order_by='created'*)

Changed in version 1.14.0: Added the *order_by* parameter.

Parameters

- **namespace** (*str*) – The unique identifier of this namespace.
- **order_by** (*str*) – The attribute to order stored items by. This must be one of "created", "id", "key", or "modified".

order_by

serializer

alias of *king_phisher.serializers.MsgPack*

validation

This module provides the functionality to perform context-sensitive validation of database models.

Functions

validate_credential(*credential*, *campaign*)

Validate a *credential* object with regards to the configuration provided in *campaign*. This uses *validate_credential_fields()* to validate each field individually and then return either `None`, `True` or `False`. If no validation took place on any field, `None` is returned, otherwise if any field was validated then a boolean is returned indicating whether or not all validated (non-`None`) fields passed validation.

Parameters

- **credential** – The credential object to validate.
- **campaign** – The campaign with the validation configuration.

Returns Either a boolean or `None` depending on the results.

validate_credential_fields(*credential*, *campaign*)

Validate a *credential* object with regards to the configuration provided in *campaign*. Each field in the *credential* object is validated and a new *CredentialCollection* is returned with it's fields set to the results of the validation. A fields validation results are either `None`, `True` or `False`. If no validation took place on the field, either because nothing was configured for it in *campaign*,or the validation information was invalid (a malformed regex for example) the result will be `None`. Otherwise, the result is either `True` or `False` for the field depending on the validation.

Parameters

- **credential** – The credential object to validate.
- **campaign** – The campaign with the validation configuration.

Returns A *CredentialCollection* object with the fields set to the results of their respective validation.

Return type *CredentialCollection*

Classes

class CredentialCollection(*username*, *password*, *mfa_token*)

Bases: tuple

A collection describing raw credential information.

mfa_token
Alias for field number 2

password
Alias for field number 1

username
Alias for field number 0

1.2.2 graphql

This package provides the GraphQL interface for querying information from the King Phisher server. This allows flexibility in how the client would like for the returned data to be formatted. This interface can be accessed directly by the server or through the RPC end point at *rpc_graphql()*.

types

database

Functions

sa_get_relationship(*session*, *model*, *name*)

Resolve the relationship on a SQLAlchemy model to either an object (in the case of one-to-one relationships) or a query to all of the objects (in the case of one-to-many relationships).

Parameters

- **session** – The SQLAlchemy session to associate the query with.
- **model** – The SQLAlchemy model of the object associated with the relationship.
- **name** – The name of the relationship as it exists in the *model*.

Returns Either the object or a SQLAlchemy query for the objects.

sa_object_resolver(*attname*, *default_value*, *model*, *info*, ***kwargs*)

Resolve the attribute for the given SQLAlchemy model object. If the attribute is a relationship, use *sa_get_relationship()* to resolve it.

Parameters

- **attname** (*str*) – The name of the attribute to resolve on the object.

- **default_value** – The default value to return if the attribute is unavailable.
- **model** (sqlalchemy.ext.declarative.api.Base) – The SQLAlchemy model to resolve the attribute for.
- **info** (graphql.execution.base.ResolveInfo) – The resolve information for this execution.

middlware

Classes

class AuthorizationMiddleware

Bases: object

An authorization provider to ensure that the permissions on the objects that are queried are respected. If no **rpc_session** key is provided in the **context** dictionary then no authorization checks can be performed and all objects and operations will be accessible. The **rpc_session** key's value must be an instance of *AuthenticatedSession*.

classmethod info_has_read_prop_access(*info*, *model*, *field_name=None*, *instance=None*)

Check that the context provided by *info* has access to read the specified property of the model. This can be used to ensure that sessions which can not read a protected field can also not obtain indirect access such as filtering or sorting by it.

Parameters

- **info** (graphql.execution.base.ResolveInfo) – The resolve information for this execution.
- **model** (sqlalchemy.ext.declarative.api.Base) – The SQLAlchemy model to check read-property access on.
- **field_name** (*str*) – The specific field name to check, otherwise info.field_name.
- **instance** – An optional instance of *model* to use for the access check.

Returns Whether or not the context is authorized to access the property.

Return type bool

schema

Classes

class Query(**args*, ***kwargs*)

Bases: graphene.types.objecttype.ObjectType

This is the root query object used for GraphQL queries.

class Schema(***kwargs*)

Bases: graphene.types.schema.Schema

This is the top level schema object for GraphQL. It automatically sets up sane defaults to be used by the King Phisher server including setting the query to *Query* and adding the *AuthorizationMiddleware* to each execution.

1.2.3 aaa

This module provides the functionality authentication authorization and access to the server application.

Classes

class AuthenticatedSession(*user*)
Bases: object

A container to store information associated with an authenticated session.

__init__(*user*)

Parameters **user** (User) – The user object of the authenticated user.

created

event_socket
An optional *EventSocket* associated with the client. If the client has not opened an event socket, this is None.

classmethod from_db_authenticated_session(*stored_session*)
Load an instance from a record stored in the database.

Parameters **stored_session** – The authenticated session from the database to load.

Returns A new *AuthenticatedSession* instance.

last_seen

user

user_access_level

user_is_admin

class AuthenticatedSessionManager(*timeout='30m'*)
Bases: object

A container for managing authenticated sessions.

__init__(*timeout='30m'*)

Parameters **timeout** (*int, str*) – The length of time in seconds for which sessions are valid.

clean()
Remove sessions which have expired.

get(*session_id*, *update_timestamp=True*)
Look up an *AuthenticatedSession* instance from it's unique identifier and optionally update the last seen timestamp. If the session is not found or has expired, None will be returned.

Parameters

- **session_id** (*str*) – The unique identifier of the session to retrieve.
- **update_timestamp** (*bool*) – Whether or not to update the last seen timestamp for the session.

Returns The session if it exists and is active.

Return type *AuthenticatedSession*

put(*user*)
Create and store a new *AuthenticatedSession* object for the specified user id. Any previously existing sessions for the specified user are removed from the manager.

Parameters **user** (User) – The user object of the authenticated user.

Returns The unique identifier for this session.

Return type str

remove(*session_id*)
Remove the specified session from the manager.

Parameters **session_id** (*str*) – The unique identifier for the session to remove.

stop()

class CachedPassword(*pw_hash*)
Bases: object

A cached in-memory password. Cleartext passwords are salted with data generated at runtime and hashed before being stored for future comparisons.

__init__(*pw_hash*)

Parameters **pw_hash** (*bytes*) – The salted hash of the password to cache.

hash_algorithm = 'sha512'

iterations = 5000

classmethod new_from_password(*password*)
Create a new instance from a plaintext password.

Parameters **password** (*str*) – The password to cache in memory.

pw_hash

salt = '.k^k/QKq'

time

class ForkedAuthenticator(*cache_timeout='10m'*, *required_group=None*, *pam_service='sshd'*)
Bases: object

This provides authentication services to the King Phisher server through PAM. It is initialized while the server is running as root and forks into the background before the privileges are dropped. The child continues to run as root and forwards requests to PAM on behalf of the parent process which is then free to drop privileges. The pipes use JSON to encode the request data as a string before sending it and using a newline character as the terminator. Requests from the parent process to the child process include a sequence number which must be included in the response.

__init__(*cache_timeout='10m'*, *required_group=None*, *pam_service='sshd'*)

Parameters

- **cache_timeout** (*int, str*) – The life time of cached credentials in seconds.
- **required_group** (*str*) – A group that if specified, users must be a member of to be authenticated.
- **pam_service** (*str*) – The service to use for identification to pam when authenticating.

authenticate(*username*, *password*)
Check if a username and password are valid. If they are, the password will be salted, hashed with SHA-512 and stored so the next call with the same values will not require sending a request to the forked child.

Parameters

- **username** (*str*) – The username to check.
- **password** (*str*) – The password to check.

Returns Whether the credentials are valid or not.

Return type bool

cache = None
The credential cache dictionary. Keys are usernames and values are tuples of password hashes and ages.

cache_timeout = None
The timeout of the credential cache in seconds.

child_pid = None
The PID of the forked child.

child_routine()
The main routine that is executed by the child after the object forks. This loop does not exit unless a stop request is made.

response_timeout = None
The timeout for individual requests in seconds.

send(*request*)
Encode and send a request through the pipe to the opposite end. This also sets the 'sequence' member of the request and increments the stored value.

Parameters **request** (*dict*) – A request.

sequence_number = None
A sequence number to use to align requests with responses.

stop()
Send a stop request to the child process and wait for it to exit.

1.2.4 build

This module contains the functionality to build a new server instance from a configuration file. This intends to keep the error checking logic for potential configuration problems contained.

Functions

get_bind_addresses(*config*)
Retrieve the addresses on which the server should bind to. Each of these addresses should be an IP address, port and optionally enable SSL. The returned list will contain tuples for each address found in the configuration. These tuples will be in the (host, port, use_ssl) format that is compatible with AdvancedHTTPServer.

Parameters **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.

Returns The specified addresses to bind to.

Return type list

get_ssl_hostnames(*config*)
Retrieve the SSL hosts that are specified within the configuration. This also ensures that the settings appear to be valid by ensuring that the necessary files are defined and readable.

Parameters **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.

Returns The specified SSH hosts.

Return type list

server_from_config(*config*, *handler_klass=None*, *plugin_manager=None*)

Build a server from a provided configuration instance. If *handler_klass* is specified, then the object must inherit from the corresponding KingPhisherServer base class.

Parameters

- **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.
- **handler_klass** (*KingPhisherRequestHandler*) – Alternative handler class to use.
- **plugin_manager** (*ServerPluginManager*) – The server's plugin manager instance.

Returns A configured server instance.

Return type *KingPhisherServer*

1.2.5 configuration

This module provides the functionality to load the server's configuration data.

Functions

ex_load_config(*config_file*, *validate_schema=True*)

Load the server configuration from the specified file. This function is meant to be called early on during a scripts execution and if any error occurs, details will be printed and the process will exit.

Parameters

- **config_file** (*str*) – The path to the configuration file to load.
- **validate_schema** (*bool*) – Whether or not to validate the schema of the configuration.

Returns The loaded server configuration.

Return type *Configuration*

Classes

class Configuration(*mem_object*, *prefix=''*)

Bases: smoke_zephyr.configuration.MemoryConfiguration

The server configuration object. This can load from files in both the JSON and YAML formats. Files in the YAML format can use the `!include` directive to include data from other files of supported formats.

classmethod from_file(*file_path*)

Load the configuration from the specified file.

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

Returns The loaded server configuration.

Return type *Configuration*

iter_schema_errors(*schema_file*)
Iterate over the ValidationError instances for all errors found within the specified schema.

Parameters **schema_file** (*str*) – The path to the schema file to use for validation.

Returns Each of the validation errors.

Return type ValidationError

schema_errors(*schema_file*)
Get a tuple of ValidationError instances for all errors found within the specified schema.

Parameters **schema_file** (*str*) – The path to the schema file to use for validation.

Returns The validation errors.

Return type tuple

1.2.6 fs_utilities

This module collects various useful file system utility functions that are used throughout the application.

Functions

access(*path*, *mode*, *user=AUTOMATIC*, *group=AUTOMATIC*)
This is a high-level wrapper around os.access() to provide additional functionality. Similar to *os.access* this high-level wrapper will test the given path for a variety of access modes. Additionally *user* or *group* can be specified to test access for a specific user or group.

New in version 1.14.0.

Parameters

- **path** (*str*) – The path to test access for.
- **mode** (*int*) – The mode to test access for. Set to *os.R_OK* to test for readability, *os.W_OK* for writability and *os.X_OK* to determine if path can be executed.
- **user** (int, str, None, *AUTOMATIC*) – The user to test permissions for. If set to *AUTOMATIC*, the process's current uid will be used.
- **group** (int, str, None, *AUTOMATIC*) – The group to test permissions for. If set to *AUTOMATIC*, the group that *user* belongs too will be used.

Returns Returns True only if the user or group has the mode of permission specified else returns False.

Return type bool

chown(*path*, *user=None*, *group=AUTOMATIC*, *recursive=True*)
This is a high-level wrapper around os.chown() to provide additional functionality. None can be specified as the *user* or *group* to leave the value unchanged. At least one of either *user* or *group* must be specified.

New in version 1.14.0.

Parameters

- **path** (*str*) – The path to change the owner information for.
- **user** (int, str, None, *AUTOMATIC*) – The new owner to set for the path. If set to *AUTOMATIC*, the process's current uid will be used.

- **group** (int, str, None, *AUTOMATIC*) – The new group to set for the path. If set to *AUTOMATIC*, the group that *user* belongs too will be used.
- **recursive** (*bool*) – Whether or not to recurse into directories.

1.2.7 letsencrypt

This module provides the functionality related to managing SSL certificates with Let's Encrypt.

Data

LETS_ENCRYPT_DEFAULT_DATA_PATH
The default path at which Let's Encrypt data is stored.

Functions

certbot_issue(*webroot*, *hostname*, *bin_path=None*, *unified_directory=None*)
Issue a certificate using Let's Encrypt's `certbot` utility. This function wraps the `certbot` binary and configures the parameters as appropriate. By default, the resulting certificate will be placed under *LETS_ENCRYPT_DEFAULT_DATA_PATH*, however if *unified_directory* is used then it will be under `$unified_directory/etc`.

Parameters

- **webroot** (*str*) – The webroot to use while requesting the certificate.
- **hostname** (*str*) – The hostname of the certificate to request.
- **bin_path** (*str*) – The optional path to the `certbot` binary. If not specified, then it will be searched for utilizing *which()*.
- **unified_directory** (*str*) – A single directory under which all the Let's Encrypt data should be stored. This is useful when not running the utility as root.

Returns The exit status of the `certbot` utility.

Return type int

get_certbot_bin_path(*config=None*)
Get the path to Let's Encrypt's `certbot` command line utility. If the path is found, it is verified to be both a file and executable. If the path verification fails, `None` is returned.

New in version 1.14.0.

Parameters **config** (`smoke_zephyr.configuration.Configuration`) – Configuration to retrieve settings from.

Returns The path to the certbot binary.

Return type str

get_sni_hostname_config(*hostname*, *config=None*)
Search for and return the SNI configuration for the specified *hostname*. This method will first check to see if the entry exists in the database before searching the Let's Encrypt data directory (if `data_path` is present in the server configuration). If no configuration data is found, or the data file paths appear invalid, `None` is returned.

Parameters

- **hostname** (*str*) – The hostname to retrieve the configuration for.

- **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.

Returns The SNI configuration for the hostname if it was found.

Return type *SNIHostnameConfiguration*

get_sni_hostnames(*config=None*, *check_files=True*)

Retrieve all the hostnames for which a valid SNI configuration can be retrieved. These are the hostnames for which SNI can be enabled. If *check_files* is enabled, the data files will be checked to ensure that they exist and are readable, else the configuration will be omitted.

Parameters

- **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.
- **check_files** (*bool*) – Whether or not to check the referenced data files.

Returns A dictionary, keyed by hostnames with values of *SNIHostnameConfiguration* instances.

Return type dict

Classes

class SNIHostnameConfiguration(*certfile*, *keyfile*, *enabled*)

The information for a certificate used by the server's SSL Server Name Indicator (SNI) extension.

certfile

The path to the SSL certificate file on disk to use for the hostname.

keyfile

The path to the SSL key file on disk to use for the hostname.

enabled

Whether or not this configuration is set to be loaded by the server.

1.2.8 plugins

Classes

class ServerPlugin(*root_config*)

Bases: *king_phisher.plugins.PluginBase*

The base object to be inherited by plugins that are loaded into the King Phisher server. This provides a convenient interface for interacting with the runtime.

config

A dictionary that can be used by this plugin to access it's configuration. Any changes to this configuration will be lost with the server restarts.

register_http(*path*, *method*)

Register a new HTTP request handler at *path* that is handled by *method*. Two parameters are passed to the method. The first parameter is a *KingPhisherRequestHandler* instance and the second is a dictionary of the HTTP query parameters. The specified path is added within the plugins private HTTP handler namespace at `_/plugins/$PLUGIN_NAME/$PATH`

> **Warning:** This resource can be reached by any user whether or not they are authenticated and or associated with a campaign.

New in version 1.7.0.

Parameters

- **path** (*str*) – The path to register the method at.
- **method** – The handler for the HTTP method.

register_rpc(*path*, *method*, *database_access=False*)
Register a new RPC function at *path* that is handled by *method*. This RPC function can only be called by authenticated users. A single parameter of the *KingPhisherRequestHandler* instance is passed to *method* when the RPC function is invoked. The specified path is added within the plugins private RPC handler namespace at `plugins/$PLUGIN_NAME/$PATH`.

New in version 1.7.0.

Changed in version 1.12.0: Added the *database_access* parameter.

Parameters

- **path** (*str*) – The path to register the method at.
- **method** – The handler for the RPC method.

root_config = None
A reference to the main server instance *config*.

server = None
A reference to the *KingPhisherServer* instance. Only available if the instance has been created.

storage = None
An instance of *KeyValueStorage* for this plugin to use for persistent data storage. This attribute is None until the *db_initialized* signal is emitted.

class ServerPluginManager(*config*)
Bases: *king_phisher.plugins.PluginManagerBase*

The manager for plugins loaded into the King Phisher server application.

1.2.9 pylibc

This module provides a wrapped interface for Linux's libc. Most of this functionality is duplicated in Python's own `grp` and `pwd` modules. This implementation however, using `ctypes` to directly interface with libc is necessary to avoid dead-lock issues when authenticating non-local users such as would be found in an environment using an LDAP server.

Functions

getgrnam(*name*, *encoding='utf-8'*)
Get the structure containing the fields from the specified entry in the group database. See getgrnam(3) for more information.

Parameters

- **name** (*str*) – The group name to look up.

- **encoding** (*str*) – The encoding to use for strings.

Returns The entry from the group database or `None` if it was not found.

Return type tuple

getgrouplist(*user*, *group=AUTOMATIC*, *encoding='utf-8'*)
Get the groups that the specified user belongs to. If *group* is not specified, it will be looked up from the password record for *user*. See getgrouplist(3) for more information.

Parameters

- **user** (*str*) – The user name to look up.
- **group** (*int*) – An optional group to add to the returned groups.
- **encoding** (*str*) – The encoding to use for strings.

Returns The group IDs that *user* belongs to.

Return type tuple

getpwnam(*name*, *encoding='utf-8'*)
Get the structure containing the fields from the specified entry in the passwrd database. See getpwnam(3) for more information.

Parameters

- **name** (*str*) – The user name to look up.
- **encoding** (*str*) – The encoding to use for strings.

Returns The entry from the user database or `None` if it was not found.

Return type tuple

getpwuid(*uid*)
Get the structure containing the fields from the specified entry in the passwrd database. See getpwuid(3) for more information.

Parameters **uid** (*int*) – The user id to look up.

Returns The entry from the user database or `None` if it was not found.

Return type tuple

1.2.10 rest_api

This module provides the functionality exposed by the server application's REST API.

Data

REST_API_BASE
The base URI path for REST API requests.

Functions

generate_token()
Generate the token to be checked when REST API requests are made.

Returns The API token

Return type str

rest_handler(*handle_function*)
A function for decorating REST API handlers. The function checks the API token in the request and encodes the handler response in JSON to be sent to the client.

Parameters **handle_function** – The REST API handler.

1.2.11 server

This module contains the functionality that provides the application's low-level HTTP server logic.

Classes

class KingPhisherRequestHandler(*request*, *client_address*, *server*, ***kwargs*)
Bases: advancedhttpserver.RequestHandler

adjust_path()
Adjust the *path* attribute based on multiple factors.

campaign_id
The campaign id that is associated with the current request's visitor. This is retrieved by looking up the *message_id* value in the database. If no campaign is associated, this value is None.

check_authorization()
Check for the presence of a basic auth Authorization header and if the credentials contained within in are valid.

Returns Whether or not the credentials are valid.

Return type bool

config = None
A reference to the main server instance *KingPhisherServer.config*.

end_headers(**args*, ***kwargs*)
Send the blank line ending the MIME headers.

get_client_ip()
Intelligently get the IP address of the HTTP client, optionally accounting for proxies that may be in use.

Returns The clients IP address.

Return type str

get_query_creds(*check_query=True*)
Get credentials that have been submitted in the request. For credentials to be returned at least a username must have been specified. The returned username will be None or a non-empty string. The returned password will be None if the parameter was not found or a string which maybe empty. This functions checks the query data for credentials first if *check_query* is True, and then checks the contents of an Authorization header.

Parameters **check_query** (*bool*) – Whether or not to check the query data in addition to an Authorization header.

Returns The submitted credentials.

Return type *CredentialCollection*

get_template_vars_client()
Build a dictionary of variables for a client with an associated campaign.

Returns The client specific template variables.

Return type dict

issue_alert(*campaign_id*, *table*, *count*)
Send a campaign alert for the specified table.

Parameters

- **campaign_id** (*int*) – The campaign subscribers to send the alert to.
- **table** (*str*) – The type of event to use as the sender when it is forwarded.
- **count** (*int*) – The number associated with the event alert.

message_id
The message id that is associated with the current request's visitor. This is retrieved by looking at an 'id' parameter in the query and then by checking the *visit_id* value in the database. If no message id is associated, this value is None. The resulting value will be either a confirmed valid value, or the value of the configurations server.secret_id for testing purposes.

on_init()
This method is meant to be over ridden by custom classes. It is called as part of the __init__ method and provides an opportunity for the handler maps to be populated with entries or the config to be customized.

path = None
The resource path of the current HTTP request.

respond_file(*file_path*, *attachment=False*, *query=None*)
Respond to the client by serving a file, either directly or as an attachment.

Parameters

- **file_path** (*str*) – The path to the file to serve, this does not need to be in the web root.
- **attachment** (*bool*) – Whether to serve the file as a download by setting the Content-Disposition header.

respond_not_found()
Respond to the client with a default 404 message.

respond_redirect(*location='/'*)
Respond to the client with a 301 message and redirect them with a Location header.

Parameters **location** (*str*) – The new location to redirect the client to.

send_response(*code*, *message=None*)
Add the response header to the headers buffer and log the response code.

Also send two standard headers with the server software version and the current date.

vhost
The value of the Host HTTP header.

visit_id
The visit id that is associated with the current request's visitor. This is retrieved by looking for the King Phisher cookie. If no cookie is set, this value is None.

class KingPhisherServer(*config*, *plugin_manager*, *handler_klass*, **args*, ***kwargs*)
Bases: advancedhttpserver.AdvancedHTTPServer

The main HTTP and RPC server for King Phisher.

__init__(*config*, *plugin_manager*, *handler_klass*, **args*, ***kwargs*)

Parameters **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.

add_sni_cert (*hostname*, *ssl_certfile=None*, *ssl_keyfile=None*, *ssl_version=None*)
Add an SSL certificate for a specific hostname as supported by SSL's Server Name Indicator (SNI) extension. See **RFC 3546** for more details on SSL extensions. In order to use this method, the server instance must have been initialized with at least one address configured for SSL.

> **Warning:** This method will raise a RuntimeError if either the SNI extension is not available in the ssl module or if SSL was not enabled at initialization time through the use of arguments to *__init__()*.

New in version 2.0.0.

Parameters

- **hostname** (*str*) – The hostname for this configuration.
- **ssl_certfile** (*str*) – An SSL certificate file to use, setting this enables SSL.
- **ssl_keyfile** (*str*) – An SSL certificate file to use.
- **ssl_version** – The SSL protocol version to use.

config = None
A Configuration instance used as the main King Phisher server configuration.

headers = None
A OrderedDict containing additional headers specified from the server configuration to include in responses.

job_manager = None
A JobManager instance for scheduling tasks.

remove_sni_cert (*hostname*)
Remove the SSL Server Name Indicator (SNI) certificate configuration for the specified *hostname*.

> **Warning:** This method will raise a RuntimeError if either the SNI extension is not available in the ssl module or if SSL was not enabled at initialization time through the use of arguments to *__init__()*.

New in version 2.2.0.

Parameters **hostname** (*str*) – The hostname to delete the SNI configuration for.

shutdown (**args*, ***kwargs*)
Request that the server perform any cleanup necessary and then shut down. This will wait for the server to stop before it returns.

1.2.12 server_rpc

This module provides the RPC server functionality that is used by the client to communicate with the server application.

Data

CONFIG_READABLE
Configuration options that can be accessed by the client.

CONFIG_WRITEABLE
Configuration options that can be changed by the client at run time.

RPC_AUTH_HEADER = 'X-RPC-Auth'
The header which contains the RPC authorization / session token.

VIEW_ROW_COUNT = 50
The default number of rows to return when one of the /view methods are called.

Functions

register_rpc(*path*, *database_access=False*, *log_call=False*)
Register an RPC function with the HTTP request handler. This allows the method to be remotely invoked using King Phisher's standard RPC interface. If *database_access* is specified, a SQLAlchemy session will be passed as the second argument, after the standard `RequestHandler` instance.

Parameters

- **path** (*str*) – The path for the RPC function.
- **database_access** (*bool*) – Whether or not the function requires database access.
- **log_call** (*bool*) – Whether or not to log the arguments which the function is called with.

rpc_campaign_alerts_is_subscribed(*handler*, *session*, *campaign_id*)
Check if the user is subscribed to alerts for the specified campaign.

Parameters **campaign_id** (*int*) – The ID of the campaign.

Returns The alert subscription status.

Return type bool

rpc_campaign_alerts_subscribe(*handler*, *session*, *campaign_id*)
Subscribe to alerts for the specified campaign.

Parameters **campaign_id** (*int*) – The ID of the campaign.

rpc_campaign_alerts_unsubscribe(*handler*, *session*, *campaign_id*)
Unsubscribe to alerts for the specified campaign.

Parameters **campaign_id** (*int*) – The ID of the campaign.

rpc_campaign_landing_page_new(*handler*, *session*, *campaign_id*, *hostname*, *page*)
Add a landing page for the specified campaign. Landing pages refer to resources that when visited by a user should cause the visit counter to be incremented.

Parameters

- **campaign_id** (*int*) – The ID of the campaign.
- **hostname** (*str*) – The hostname which will be used to serve the request.
- **page** (*str*) – The request resource.

rpc_campaign_message_new(*handler*, *session*, *campaign_id*, *email_id*, *target_email*, *first_name*, *last_name*, *department_name=None*)
Record a message that has been sent as part of a campaign. These details can be retrieved later for value substitution in template pages.

Parameters

- **campaign_id** (*int*) – The ID of the campaign.
- **email_id** (*str*) – The message id of the sent email.
- **target_email** (*str*) – The email address that the message was sent to.
- **first_name** (*str*) – The first name of the message's recipient.
- **last_name** (*str*) – The last name of the message's recipient.
- **department_name** (*str*) – The name of the company department that the message's recipient belongs to.

rpc_campaign_new(*handler*, *session*, *name*, *description=None*)
Create a new King Phisher campaign and initialize the database information.

Parameters

- **name** (*str*) – The new campaign's name.
- **description** (*str*) – The new campaign's description.

Returns The ID of the new campaign.

Return type int

rpc_campaign_stats(*handler*, *session*, *campaign_id*)
Generate statistics regarding the specified campaign and return them in a dictionary. The dictionary will contain the keys credentials, credentials-unique, messages, messages-trained, visits, visits-unique. Values with unique in the key are counted unique by the message id for which they are associated.

Parameters **campaign_id** – The unique ID of the campaign to generate statistics for.

Returns The statistics for the specified campaign.

Return type dict

rpc_config_get(*handler*, *option_name*)
Retrieve a value from the server's configuration.

Parameters **option_name** (*str*) – The name of the configuration option.

Returns The option's value.

rpc_config_set(*handler*, *options*)
Set options in the server's configuration. Any changes to the server's configuration are not written to disk.

Parameters **options** (*dict*) – A dictionary of option names and values

rpc_events_is_subscribed(*handler*, *event_id*, *event_type*)
Check if the client is currently subscribed to the specified server event.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_type** (*str*) – A sub-type for the corresponding event.

Returns Whether or not the client is subscribed to the event.

Return type bool

rpc_events_subscribe(*handler*, *event_id*, *event_types=None*, *attributes=None*)
Subscribe the client to the specified event published by the server. When the event is published the specified *attributes* of it and it's corresponding id and type information will be sent to the client.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

rpc_events_unsubscribe(*handler*, *event_id*, *event_types=None*, *attributes=None*)
Unsubscribe from an event published by the server that the client previously subscribed to.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

rpc_database_count_rows(*handler*, *session*, *table_name*, *query_filter=None*)
Get a count of the rows in the specified table where the search criteria matches.

Parameters

- **table_name** (*str*) – The name of the database table to query.
- **query_filter** (*dict*) – A dictionary mapping optional search criteria for matching the query.

Returns The number of matching rows.

Return type int

rpc_database_delete_row_by_id(*handler*, *session*, *table_name*, *row_id*)
Delete the row from the table with the specified value in the id column. If the row does not exist, no error is raised.

Parameters

- **table_name** (*str*) – The name of the database table to delete a row from.
- **row_id** – The id value.

rpc_database_delete_rows_by_id(*handler*, *session*, *table_name*, *row_ids*)
Delete multiple rows from a table with the specified values in the id column. If a row id specified in *row_ids* does not exist, then it will be skipped and no error will be thrown.

Parameters

- **table_name** (*str*) – The name of the database table to delete rows from.
- **row_ids** (*list*) – The row ids to delete.

Returns The row ids that were deleted.

Return type list

rpc_database_get_row_by_id(*handler*, *session*, *table_name*, *row_id*)
Retrieve a row from a given table with the specified value in the id column.

Parameters

- **table_name** (*str*) – The name of the database table to retrieve a row from.
- **row_id** – The id value.

Returns The specified row data.

Return type dict

rpc_database_insert_row(*handler*, *session*, *table_name*, *keys*, *values*)
Insert a new row into the specified table.

Parameters

- **table_name** (*str*) – The name of the database table to insert a new row into.
- **keys** (*list*) – The column names of *values*.
- **values** (*list*) – The values to be inserted in the row.

Returns The id of the new row that has been added.

rpc_database_set_row_value(*handler*, *session*, *table_name*, *row_id*, *keys*, *values*)
Set values for a row in the specified table with an id of *row_id*.

Parameters

- **table_name** (*str*) – The name of the database table to set the values of the specified row.
- **keys** (*tuple*) – The column names of *values*.
- **values** (*tuple*) – The values to be updated in the row.

rpc_database_view_rows(*handler*, *session*, *table_name*, *page=0*, *query_filter=None*)
Retrieve the rows from the specified table where the search criteria matches.

Parameters

- **table_name** (*str*) – The name of the database table to query.
- **page** (*int*) – The page number to retrieve results for.
- **query_filter** (*dict*) – A dictionary mapping optional search criteria for matching the query.

Returns A dictionary with columns and rows keys.

Return type dict

rpc_geoip_lookup(*handler*, *ip*, *lang=None*)
Look up an IP address in the servers GeoIP database. If the IP address can not be found in the database, None will be returned.

Parameters

- **ip** (*str*) – The IP address to look up.
- **lang** (*str*) – The language to prefer for regional names.

Returns The geographic information for the specified IP address.

Return type dict

rpc_geoip_lookup_multi(*handler*, *ips*, *lang=None*)
Look up multiple IP addresses in the servers GeoIP database. Each IP address that can not be found in the database will have its result set to None.

Parameters

- **ips** (*list*) – The list of IP addresses to look up.
- **lang** (*str*) – The language to prefer for regional names.

Returns A dictionary containing the results keyed by the specified IP addresses.

Return type dict

rpc_graphql (*handler*, *session*, *query*, *query_vars=None*)
Execute a GraphQL query and return the results. If the query fails to execute the errors returned are populated in the **errors** key of the results dictionary. If the query executes successfully the returned data is available in the **data** key of the results dictionary.

Parameters

- **query** (*str*) – The GraphQL query to execute.
- **query_vars** (*dict*) – Any variables needed by the *query*.

Returns The results of the query as a dictionary.

Return type dict

rpc_hostnames_add (*handler*, *hostname*)
Add a hostname to the list of values that are configured for use with this server. At this time, these changes (like other config changes) are not persisted in the server so they will be lost when the server reboots.

New in version 1.13.0.

Parameters **hostname** (*str*) – The hostname to add.

rpc_hostnames_get (*handler*)
Get the hostnames that are configured for use with this server. This is not related to the `ssl/hostnames` RPC methods which deal with hostnames as they relate to SSL for the purposes of certificate usage.

New in version 1.13.0.

Returns The configured hostnames.

Return type list

rpc_login (*handler*, *session*, *username*, *password*, *otp=None*)

rpc_logout (*handler*, *session*)

rpc_ping (*handler*)
An RPC method that can be used by clients to assert the status and responsiveness of this server.

Returns This method always returns True.

Return type bool

rpc_plugins_list (*handler*)
Return information regarding enabled plugins in the server.

Returns A dictionary representing enabled plugins and their meta-data.

Return type dict

rpc_shutdown (*handler*)
This method can be used to shut down the server. This function will return, however no subsequent requests will be processed.

> **Warning:** This action will stop the server process and there is no confirmation before it takes place.

rpc_ssl_letsencrypt_issue (*handler*, *hostname*, *load=True*)
Issue a certificate with Let's Encrypt. This operation can fail for a wide variety of reasons, check the `message` key of the returned dictionary for a string description of what occurred. Successful operation requires that the certbot utility be installed, and the server's Let's Encrypt data path is configured.

New in version 1.14.0.

Parameters

- **hostname** (*str*) – The hostname of the certificate to issue.
- **load** (*bool*) – Whether or not to load the certificate once it has been issued.

Returns A dictionary containing the results of the operation.

Return type dict

rpc_ssl_letsencrypt_certbot_version(*handler*)

Find the certbot binary and retrieve it's version information. If the certbot binary could not be found, `None` is returned.

New in version 1.14.0.

Returns The version of certbot.

Return type str

rpc_ssl_sni_hostnames_get(*handler*)

Get the hostnames that have available Server Name Indicator (SNI) configurations for use with SSL.

New in version 1.14.0.

Returns A dictionary keyed by hostnames with values of dictionaries containing additional meta-data.

Return type dict

rpc_ssl_sni_hostnames_load(*handler*, *hostname*)

Load the SNI configuration for the specified *hostname*, effectively enabling it. If SSL is not enabled, SNI is not available, or the necessary data files are not available, this function returns `False`.

New in version 1.14.0.

Parameters **hostname** (*str*) – The hostname to configure SSL for.

Returns Returns `True` only if the SNI configuration for *hostname* was either able to be loaded or was already loaded.

Return type bool

rpc_ssl_sni_hostnames_unload(*handler*, *hostname*)

Unload the SNI configuration for the specified *hostname*, effectively disabling it. If SNI is not available, or the specified configuration was not already loaded, this function returns `False`.

New in version 1.14.0.

Parameters **hostname** (*str*) – The hostname to configure SSL for.

Returns Returns `True` only if the SNI configuration for *hostname* was unloaded.

Return type bool

rpc_ssl_status(*handler*)

Get information regarding the status of SSL on the server. This method returns a dictionary with keys describing whether or not SSL is enabled on one or more interfaces, and whether or not the server possess the SNI support. For details regarding which addresses are using SSL, see the *rpc_config_get()* method.

New in version 1.14.0.

Returns A dictionary with SSL status information.

Return type dict

rpc_version(*handler*)
Get the version information of the server. This returns a dictionary with keys of version, version_info and rpc_api_version. These values are provided for the client to determine compatibility.

Returns A dictionary with version information.

Return type dict

1.2.13 signals

This module contains the signals which are used by the server to dispatch events. Additional signal details regarding how these signals are used is available in the *Server Signals* documentation.

Functions

send_safe(*signal*, *logger*, *sender*, ***kwargs*)
Send a signal and catch any exception which may be raised during it's emission. Details regarding the error that occurs (including a stack trace) are logged to the specified *logger*. This is suitable for allowing signals to be emitted in critical code paths without interrupting the emitter.

Parameters

- **signal** (*str*) – The name of the signal to send safely.
- **logger** (logging.Logger) – The logger to use for logging exceptions.
- **sender** – The sender for this signal emission.
- **kwargs** – The key word arguments to be forward to the signal as it is sent.

Signals

campaign_alert
Emitted for each user who is subscribed to alerts for a particular campaign. Users subscribe to campaign alerts through the GUI by enabling the "Subscribe To Event Alerts" setting. Alerts are for either the "credentials" or "visits" table.

Note: This signal is not emitted for every entry into the respective tables but rather at progressively longer intervals to prevent the user from receiving an excessive amount of messages within a short period of time.

Parameters

- **table** (*str*) – The table name that the alert is for.
- **alert_subscription** (king_phisher.server.database.models.AlertSubscription) – The alert subscription.
- **count** (*int*) – The number associated with the alert event per the specified sender.

credentials_received
Sent when a new pair of credentials have been submitted.

Parameters

- **request_handler** – The handler for the received request.
- **username** (*str*) – The username of the credentials that were submitted.

- **password** (*str*) – The password of the credentials that were submitted.

db_initialized
Emitted after a connection has been made and the database has been fully initialized. At this point, it is safe to operate on the database.

Parameters **connection_url** (sqlalchemy.engine.url.URL) – The connection string for the database that has been initialized.

db_session_deleted
Emitted after one or more rows have been deleted on a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been deleted with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_session_inserted
Emitted after one or more rows have been inserted in a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been inserted with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_session_updated
Emitted after one or more rows have been updated in a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been updated with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_table_delete
Emitted before a row inheriting from Base is deleted from the database table. To only subscribe to delete events for a specific table, specify the table's name as the *sender* parameter when calling blinker.base.Signal.connect(). See sqlalchemy.orm.events.MapperEvents.before_delete() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (sqlalchemy.orm.mapper.Mapper) – The Mapper object which is the target of the event.
- **connection** (sqlalchemy.engine.Connection) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.

- **target** – The target object instance.

db_table_insert
Emitted before a row inheriting from `Base` is inserted into the database table. To only subscribe to insert events for a specific table, specify the table's name as the *sender* parameter when calling `blinker.base.Signal.connect()`. See `sqlalchemy.orm.events.MapperEvents.before_insert()` for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (`sqlalchemy.orm.mapper.Mapper`) – The Mapper object which is the target of the event.
- **connection** (`sqlalchemy.engine.Connection`) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.
- **target** – The target object instance.

db_table_update
Emitted before a row inheriting from `Base` is updated in the database table. To only subscribe to update events for a specific table, specify the table's name as the *sender* parameter when calling `blinker.base.Signal.connect()`. See `sqlalchemy.orm.events.MapperEvents.before_update()` for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (`sqlalchemy.orm.mapper.Mapper`) – The Mapper object which is the target of the event.
- **connection** (`sqlalchemy.engine.Connection`) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.
- **target** – The target object instance.

email_opened
Sent when a request for the embedded image is received.

Parameters **request_handler** – The handler for the received request.

request_handle
Sent after a new HTTP request has been received and is about to be handled. This signal is suitable for implementing custom request handlers or aborting requests. This signal is emitted after *request_received* to allow subscribers the opportunity to handle requests themselves.

Note: If a request has been handled by the signal, the signal handler must raise the *KingPhisherAbortRequestError* exception to prevent further processing.

Parameters **request_handler** – The handler for the received request.

request_received
Sent when a new HTTP request has been received and is about to be handled. This signal is *not* suitable for implementing custom request handlers or aborting requests. This signal is emitted before *request_handle* allowing subscribers to be notified before a request may be blocked.

Parameters **request_handler** – The handler for the received request.

response_sent
Sent after a response to an HTTP request has been sent to the client. At this point headers may be added to the response body.

Parameters

- **request_handler** – The handler for the received request.
- **code** (*int*) – The HTTP status code that was sent in the response.
- **message** (*str*) – The HTTP message that was sent in the response.

rpc_method_call
Sent when a new RPC request has been received and it's corresponding method is about to be called.

Parameters

- **method** (*str*) – The RPC method which is about to be executed.
- **request_handler** – The handler for the received request.
- **args** (*tuple*) – The arguments that are to be passed to the method.
- **kwargs** (*dict*) – The key word arguments that are to be passed to the method.

rpc_method_called
Sent after an RPC request has been received and it's corresponding method has been called.

Parameters

- **method** (*str*) – The RPC method which has been executed.
- **request_handler** – The handler for the received request.
- **args** (*tuple*) – The arguments that were passed to the method.
- **kwargs** (*dict*) – The key word arguments that were passed to the method.
- **retval** – The value returned from the RPC method invocation.

rpc_user_logged_in
Sent when a new RPC user has successfully logged in and created a new authenticated session.

Parameters

- **request_handler** – The handler for the received request.
- **session** (*str*) – The session ID of the newly logged in user.
- **name** (*str*) – The username of the newly logged in user.

rpc_user_logged_out
Sent when an authenticated RPC user has successfully logged out and terminated their authenticated session.

Parameters

- **request_handler** – The handler for the received request.
- **session** (*str*) – The session ID of the user who has logged out.
- **name** (*str*) – The username of the user who has logged out.

server_initialized
Sent when a new instance of *KingPhisherServer* is initialized.

Parameters **server** – The newly initialized server instance.

visit_received
Sent when a new visit is received on a landing page. This is only emitted when a new visit entry is added to the database.

Parameters **request_handler** – The handler for the received request.

1.2.14 template_extras

This module provides functionality for Jinja functions used to generate server page content.

Data

functions
A dictionary of the exported page functions.

Functions

embed_youtube_video(*video_id*, *autoplay=True*, *enable_js=False*, *start=0*, *end=None*)
A Jinja function to embed a video into a web page using YouTube's iframe API. In order to enable a training button after the video has ended the youtube.js file needs to be included and *enable_js* just be set to True. If *start* or *end* are specified as strings, the must be in a format suitable to be parsed by `parse_timespan()`.

Parameters

- **video_id** (*str*) – The id of the YouTube video to embed.
- **autoplay** (*bool*) – Start playing the video as soon as the page loads.
- **enable_js** (*bool*) – Enable the Javascript API.
- **start** (*int, str*) – The time offset at which the video should begin playing.
- **end** (*int, str*) – The time offset at which the video should stop playing.

export_function(*function*)
A decorator to "export" a function by placing it in *functions*.

Parameters **function** (*function*) – The function to export.

make_csrf_page(*url*, *params*, *method='POST'*)
A Jinja function which will create an HTML page that will automatically perform a CSRF attack against another page.

Parameters

- **url** (*str*) – The URL to use as the form action.
- **params** (*dict*) – The parameters to send in the forged request.
- **method** (*str*) – The HTTP method to use when submitting the form.

make_redirect_page(*url*, *title='Automatic Redirect'*)
A Jinja function which will create an HTML page that will automatically redirect the viewer to a different url.

Parameters

- **url** (*str*) – The URL to redirect the user to.
- **title** (*str*) – The title to use in the resulting HTML page.

1.2.15 web_sockets

Classes

class Event(*event_id*, *event_type*, *sources*)
Bases: object

An object representing an event which occurred on the server in a way that is ready to be published to client subscribers.

__init__(*event_id*, *event_type*, *sources*)
Initialize self. See help(type(self)) for accurate signature.

event_id
The unique string identifier of this event.

event_type
The unique string identifier of the type of this event.

sources
The source objects which are associated with this event.

class EventSocket(*handler*, *manager*)
Bases: advancedhttpserver.WebSocketHandler

A socket through which server events are published to subscribers. This socket will automatically add and remove itself from the manager that is initialized with.

__init__(*handler*, *manager*)

Parameters

- **handler** (advancedhttpserver.RequestHandler) – The request handler that should be used by this socket.
- **manager** (*WebSocketsManager*) – The manager that this event socket should register with.

is_subscribed(*event_id*, *event_type*)
Check if the client is currently subscribed to the specified server event.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_type** (*str*) – A sub-type for the corresponding event.

Returns Whether or not the client is subscribed to the event.

Return type bool

on_closed()
A method that can be over ridden and is called after the web socket is closed.

publish(*event*)
Publish the event by sending the relevant information to the client. If the client has not requested to receive the information through a subscription, then no data will be sent.

Parameters **event** (*Event*) – The object representing the data to be published.

subscribe(*event_id*, *event_types=None*, *attributes=None*)
Subscribe the client to the specified event published by the server. When the event is published the specified *attributes* of it and it's corresponding id and type information will be sent to the client.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

unsubscribe(*event_id*, *event_types=None*, *attributes=None*)
Unsubscribe from an event published by the server that the client previously subscribed to.

Parameters

- **event_id** (*str*) – The identifier of the event to subscribe to.
- **event_types** (*list*) – A list of sub-types for the corresponding event.
- **attributes** (*list*) – A list of attributes of the event object to be sent to the client.

class WebSocketsManager(*config*, *job_manager*)
Bases: object

An object used to manage connected web sockets.

__init__(*config*, *job_manager*)

Parameters

- **config** (smoke_zephyr.configuration.Configuration) – Configuration to retrieve settings from.
- **job_manager** (smoke_zephyr.job.JobManager) – A job manager instance that can be used to schedule tasks.

add(*web_socket*)
Add a connected web socket to the manager.

Parameters **web_socket** (advancedhttpserver.WebSocketHandler) – The connected web socket.

dispatch(*handler*)
A method that is suitable for use as a web_socket_handler.

Parameters **handler** (*KingPhisherRequestHandler*) – The current request handler instance.

logger = <Logger KingPhisher.Server.WebSocketManager (WARNING)>

ping_all()
Ping all of the connected web sockets to ensure they stay alive. This method is automatically executed periodically through a job added when the manager is initialized.

remove(*web_socket*)
Remove a connected web socket from those that are currently being managed. If the web socket is not currently being managed, no changes are made.

Parameters **web_socket** (advancedhttpserver.WebSocketHandler) – The connected web socket.

stop()
Shutdown the manager and clean up the resources it has allocated.

1.2.16 web_tools

This module contains various functions related to the web-serving configuration of the server.

Functions

get_hostnames(*config*)
List the hostnames that are configured for this server instance. This list is generated by first checking the server's

configuration for the `hostnames` option. Then if `vhost_directories` is enabled, the webroot is checked for additional values.

Note: This function makes no attempt to validate these values, they are strictly what have been configured for use.

New in version 1.13.0.

Parameters **config** (`smoke_zephyr.configuration.Configuration`) – Configuration to retrieve settings from.

Returns A tuple of the enumerated hostnames.

Return type tuple

get_vhost_directories(*config*)

List the hostnames that are configured through the Virtual Host directories. If the server option `vhost_directories` is disabled, this function returns `None`.

New in version 1.13.0.

Parameters **config** (`smoke_zephyr.configuration.Configuration`) – Configuration to retrieve settings from.

Returns A tuple of the enumerated virtual hostname directories.

Return type tuple

1.3 archive

This module provides a generic means to combine data and files into a single archive file.

1.3.1 Functions

is_archive(*file_path*)

Check if the specified file appears to be a valid archive file that can be opened with *ArchiveFile*.

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

Returns Whether or not the file looks like a compatible archive.

Return type bool

patch_zipfile(*input_file*, *patches*, *output_file=None*)

Patch content into the specified input Zip file. The *input_file* must be either an input path string to the file to patch or a `zipfile.ZipFile` instance. Patch data is supplied in the *patch* argument which is a dictionary keyed by the paths to modify, values are then used in place of the specified path. If the value of a path is `None`, then that file is removed from the archive. The *output_file* is either a string to the path of where to place the patched archive, a `ZipFile` instance or `None`. If `None` is specified then *input_file* is modified in place.

Note: If a `ZipFile` instance is specified as *input_file* then *output_file* can not be `None`.

Parameters

- **input_file** (str, `ZipFile`) – The Zip file archive to modify.

- **patches** (*dict*) – The data to modify from the original archive.
- **output_file** (None, str, ZipFile) – The destination of the modified archive

1.3.2 Classes

class ArchiveFile(*file_name*, *mode*, *encoding='utf-8'*)

An object representing a generic archive for storing information. The resulting archive file is a tarfile that can easily be opened and manipulated with external tools. This class also facilitates storing metadata with the archive. This metadata contains basic information such as the version of King Phisher that generated it, and a UTC timestamp of when it was created.

__init__(*file_name*, *mode*, *encoding='utf-8'*)

Parameters

- **file_name** (*str*) – The path to the file to open as an archive.
- **mode** (*str*) – The mode to open the file such as 'r' or 'w'.
- **encoding** (*str*) – The encoding to use for strings.

add_data(*name*, *data*)

Add arbitrary data directly to the archive under the specified name. This allows data to be directly inserted into the archive without first writing it to a file or file like object.

Parameters

- **name** (*str*) – The name of the destination file in the archive.
- **data** (*bytes, str*) – The data to place into the archive.

add_file(*name*, *file_path*, *recursive=True*)

Place a file or directory into the archive. If *file_path* is a directory, it's contents will be added recursively if *recursive* is True.

Parameters

- **name** (*str*) – The name of the destination file in the archive.
- **file_path** (*str*) – The path to the file to add to the archive.
- **recursive** (*bool*) – Whether or not to add directory contents.

close()

Close the handle to the archive.

file_names

This property is a generator which yields the names of all of the contained files. The metadata file is skipped.

Returns A generator which yields all the contained file names.

Return type str

files

This property is a generator which yields tuples of two objects each where the first is the file name and the second is the file object. The metadata file is skipped.

Returns A generator which yields all the contained file name and file objects.

Return type tuple

get_data(*name*)

Return the data contained within the specified archive file.

Parameters **name** (*str*) – The name of the source file in the archive.

Returns The contents of the specified file.

Return type bytes

get_file(*name*)
Return the specified file object from the archive.

Parameters **name** (*str*) – The name of the source file in the archive.

Returns The specified file.

Return type file

get_json(*name*)
Extract the specified file, deserialize it as JSON encoded content and return the result.

New in version 1.14.0.

Parameters **name** (*str*) – The name of the source file in the archive.

Returns The deserialized contents of the specified file.

has_file(*name*)
Check if a file exists within archive.

Parameters **name** (*str*) –

Returns Whether or not the file exists.

Return type bool

metadata_file_name = 'metadata.json'

mode
A read-only attribute representing the mode that the archive file was opened in.

1.4 catalog

This module provides functionality for processing and working with data published on the available add ons for the application.

1.4.1 Overview

The classes within this module are primarily for organizing the large amount of data describing published add ons. This information is broken down into the various objects in a hierarchy where the parent contain zero or more children objects. In this sense the hierarchy is a tree data structure where the nodes are different data types such as catalogs, repositories, collections etc.

The hierarchy of these objects is as follows in order of parent to children:

- *CatalogManager*
- *Catalog*
- *Repository*
- *Collection*
- *CollectionItemFile*

1.4.2 Data

COLLECTION_TYPES
A tuple of the known collection type identity strings. Collection types are logical groupings of published data types. These type identifiers provide some context as to how the data is intended to be used and what parts of the application may be interested in using it.

1.4.3 Functions

sign_item_files(*local_path*, *signing_key*, *repo_path=None*)
This utility function is used to create a *CollectionItemFile* iterator from the specified source to be included in either a catalog file or one of it's included files.

> **Warning:** This function contains a black list of file extensions which will be skipped. This is to avoid signing files originating from the development process such as `.pyc` and `.ui~`.

Parameters

- **local_path** (*str*) – The real location of where the files exist on disk.
- **signing_key** – The key with which to sign the files for verification.
- **repo_path** (*str*) – The path of the repository as it exists on disk.

1.4.4 Classes

class Catalog(*data*, *keys=None*)
Bases: object

An object representing a set of `Repositories` containing add on data for the application. This information can then be loaded from an arbitrary source.

__init__(*data*, *keys=None*)

Parameters

- **data** (*dict*) – The formatted catalog data.
- **keys** (*SecurityKeys*) – The keys to use for verifying remote data.

created
The timestamp of when the remote data was generated.

classmethod from_url(*url*, *keys=None*, *encoding='utf-8'*)
Initialize a new *Catalog* object from a resource at the specified URL. The resulting data is validated against a schema file with *validate_json_schema()* before being passed to *__init__()*.

Parameters

- **url** (*str*) – The URL to the catalog data to load.
- **keys** (*SecurityKeys*) – The keys to use for verifying remote data.
- **encoding** (*str*) – The encoding of the catalog data.

Returns The new catalog instance.

Return type *Catalog*

id
The unique identifier of this catalog.

maintainers
A tuple containing the maintainers of the catalog and repositories. These are also the key identities that should be present for verifying the remote data.

repositories
A dict of the *Repository* objects included in this catalog keyed by their id.

security_keys
The *SecurityKeys* used for verifying remote data.

to_dict()
Dump the instance to a dictionary suitable for being reloaded with *__init__()*.

Returns The instance represented as a dictionary.

Return type dict

class CatalogManager(*catalog_url=None*)
Bases: object

Base manager for handling multiple *Catalog* instances.

__init__(*catalog_url=None*)
Initialize self. See help(type(self)) for accurate signature.

add_catalog(*catalog*)
Adds the specified catalog to the manager.

Parameters **catalog** (*Catalog*) – Add the specified catalog to the manager.

Returns The catalog.

Return type *Catalog*

catalog_ids()
The key names of the catalogs in the manager.

Returns The catalogs IDs in the manager instance.

Return type tuple

class Collection(*repo*, *type*, *items*)
Bases: collections.abc.Mapping

An object representing a set of *CollectionItemFile* instances, each of which represent a piece of of add on data that are all of the same type (see *COLLECTION_TYPES*). A collection is also a logical domain where the items contained within it must each have a unique identity in the form of its name attribute.

__init__(*repo*, *type*, *items*)

Parameters

- **repo** (*Repository*) – The repository this collection is associated with.
- **type** (*str*) – The collection type of these items.
- **items** (*dict*) – The items that are members of this collection, keyed by their name.

classmethod from_dict(*value*, *repo*)
Load the collection item file from the specified dict object.

Parameters **value** (*dict*) – The dictionary to load the data from.

Returns

get_file(**args*, ***kwargs*)
A simple convenience method which forwards to the associated *Repository*'s *get_file()* method.

get_item(**args*, ***kwargs*)
A simple convenience method which forwards to the associated *Repository*'s *get_item()* method.

get_item_files(**args*, ***kwargs*)
A simple convenience method which forwards to the associated *Repository*'s *get_item_files()* method.

to_dict()
Dump the instance to a dictionary suitable for being reloaded with *from_dict()*.

Returns The instance represented as a dictionary.

Return type dict

class CollectionItemFile(*path_destination*, *path_source*, *signature=None*, *signed_by=None*)
An object representing a single remote file and the necessary data to validate it's integrity. In order to validate the data integrity both the *signature* and *signed_by* attributes must be available. These attributes must either both be present or absent, i.e. one can not be set without the other.

__init__(*path_destination*, *path_source*, *signature=None*, *signed_by=None*)
Initialize self. See help(type(self)) for accurate signature.

classmethod from_dict(*value*)
Load the collection item file from the specified dict object.

Parameters **value** (*dict*) – The dictionary to load the data from.

Returns

path_destination
The relative path of where this file should be placed.

path_source
The relative path of where this file should be retrieved from.

signature
The signature data used for integrity verification of the represented resource.

signed_by
The identity of the *SigningKey* which generated the *signature*

to_dict()
Dump the instance to a dictionary suitable for being reloaded with *from_dict()*.

Returns The instance represented as a dictionary.

Return type dict

class Repository(*data*, *keys=None*)
Bases: object

An object representing a single logical source of add on data.

__init__(*data*, *keys=None*)

Parameters

- **data** (*dict*) – The formatted repository data.
- **keys** (*SecurityKeys*) – The keys to use for verifying remote data.

collections
The dictionary of the different collection types included in this repository.

get_file(*item_file*, *encoding=None*)
Download and return the file data from the repository. If no encoding is specified, the data is return as bytes, otherwise it is decoded to a string using the specified encoding. The file's contents are verified using the signature that must be specified by the *item_file* information.

Parameters

- **item_file** (*CollectionItemFile*) – The information for the file to download.
- **encoding** (*str*) – An optional encoding of the remote data.

Returns The files contents.

Return type bytes, str

get_item(*collection_type*, *name*)
Get the item by it's name from the specified collection type. If the repository does not provide the named item, None is returned.

Parameters

- **collection_type** (*str*) – The type of collection the specified item is in.
- **name** (*str*) – The name of the item to retrieve.

Returns The item if the repository provides it, otherwise None.

get_item_files(*collection_type*, *name*, *destination*)
Download all of the file references from the named item.

Parameters

- **collection_type** (*str*) – The type of collection the specified item is in.
- **name** (*str*) – The name of the item to retrieve.
- **destination** (*str*) – The path of where to save the downloaded files to.

homepage
The URL of the homepage for this repository if it was specified.

id
The unique identifier of this repository.

security_keys
The *SecurityKeys* used for verifying remote data.

title
The title string of this repository.

to_dict()
Dump the instance to a dictionary suitable for being reloaded with *__init__()*.

Returns The instance represented as a dictionary.

Return type dict

url_base
The base URL string of files included in this repository.

1.5 color

This module provides functions for converting and using colors for arbitrary purposes including terminal output.

1.5.1 Functions

convert_hex_to_tuple(*hex_color*, *raw=False*)
Converts an RGB hex triplet such as #ff0000 into an RGB tuple. If *raw* is True then each value is on a scale from 0 to 255 instead of 0.0 to 1.0.

Parameters

- **hex_color** (*str*) – The hex code for the desired color.
- **raw** (*bool*) – Whether the values are raw or percentages.

Returns The color as a red, green, blue tuple.

Return type tuple

convert_tuple_to_hex(*rgb*, *raw=False*)
Converts an RGB color tuple info a hex string such as #ff0000. If *raw* is True then each value is treated as if it were on a scale from 0 to 255 instead of 0.0 to 1.0.

Parameters

- **rgb** (*tuple*) – The RGB tuple to convert into a string.
- **raw** (*bool*) – Whether the values are raw or percentages.

Returns The RGB color as a string.

Return type str

get_scale(*color_low*, *color_high*, *count*, *ascending=True*)
Create a scale of colors gradually moving from the low color to the high color.

Parameters

- **color_low** (*tuple*) – The darker color to start the scale with.
- **color_high** (*tuple*) – The lighter color to end the scale with.
- **count** – The total number of resulting colors.
- **ascending** (*bool*) – Whether the colors should be ascending from lighter to darker or the reverse.

Returns An array of colors starting with the low and gradually transitioning to the high.

Return type tuple

print_error(*message*)
Print an error message to the console.

Parameters **message** (*str*) – The message to print

print_good(*message*)
Print a good message to the console.

Parameters **message** (*str*) – The message to print

print_status(*message*)
Print a status message to the console.

Parameters **message** (*str*) – The message to print

1.5.2 Classes

class ColoredLogFormatter(*fmt=None*, *datefmt=None*, *style='%'*)
A formatting class suitable for use with the `logging` module which colorizes the names of log levels.

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

static formatException(*exc_info*)
Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

1.6 constants

This module keeps collections of related constants organized for use in other modules.

1.6.1 Data

DEFAULT_LOG_LEVEL = 'WARNING'
The default log level to use for filtering messages by importance.

Sentinel Values

Sentinel values are used as place holders where `None` may be valid and have a different meaning.

AUTOMATIC = AUTOMATIC
A sentinel value to indicate that a feature or value is determined automatically.

DISABLED = DISABLED
A sentinel value to indicate that a feature or value is disabled.

1.6.2 Classes

class ConstantGroup
A class for grouping related constants together.

classmethod items()
Iterate over the names and values in a group of constants.

classmethod names()
Iterate over the names in a group of constants.

classmethod values()
Iterate over the values in a group of constants.

class ConnectionErrorReason
Constants which describe possible errors for an arbitrary connection process.

ConnectionErrorReason.**ERROR_AUTHENTICATION_FAILED = 'authentication failed'**

ConnectionErrorReason.**ERROR_CONNECTION = 'connection error'**

ConnectionErrorReason.**ERROR_INCOMPATIBLE_VERSIONS = 'incompatible versions'**

ConnectionErrorReason.**ERROR_INVALID_CREDENTIALS = 'invalid credentials'**

ConnectionErrorReason.**ERROR_INVALID_OTP = 'invalid otp'**

ConnectionErrorReason.**ERROR_PORT_FORWARD = 'port forward error'**

ConnectionErrorReason.**ERROR_UNKNOWN = 'unknown error'**

ConnectionErrorReason.**SUCCESS = 'success'**

class OSArch
Constants for different operating system architectures.

OSArch.**PPC = 'PPC'**

OSArch.**X86 = 'x86'**

OSArch.**X86_64 = 'x86-64'**

class OSFamily
Constants for families of different operating systems.

OSFamily.**ANDROID = 'Android'**

OSFamily.**BLACKBERRY = 'BlackBerry'**

OSFamily.**IOS = 'iOS'**

OSFamily.**LINUX = 'Linux'**

OSFamily.**OSX = 'OS X'**

OSFamily.**WINDOWS = 'Windows NT'**

1.7 errors

This module provides the custom exceptions that are used throughout the package.

1.7.1 Exceptions

exception KingPhisherError(*message=''*)
Bases: Exception

The base exception that is inherited by all custom King Phisher error classes.

exception KingPhisherAbortError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that can be raised to indicate that an arbitrary operation needs to be aborted when no better method can be used.

exception KingPhisherAbortRequestError(*response_sent=False*)
Bases: *king_phisher.errors.KingPhisherAbortError*

An exception that can be raised which when caught will cause the handler to immediately stop processing the current request.

__init__(*response_sent=False*)

Parameters **response_sent** (*bool*) – Whether or not a response has already been sent to the client.

exception KingPhisherDatabaseError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that can be raised by King Phisher when there is any error relating to the database, it's configuration or any action involving it. The underlying database API will raise exceptions of it's own kind.

exception KingPhisherDatabaseAuthenticationError(*message*, *username=None*)
Bases: *king_phisher.errors.KingPhisherDatabaseError*

An exception that is raised when King Phisher can not authenticate to the database. This is usually due to the configured password being incorrect.

exception KingPhisherGraphQLQueryError(*message=''*, *errors=None*, *query=None*, *query_vars=None*)
Bases: *king_phisher.errors.KingPhisherError*

An exception raised when a GraphQL query fails to execute correctly.

exception KingPhisherInputValidationError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is raised when any kind of input into King Phisher fails to be properly validated.

exception KingPhisherPermissionError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is raised by King Phisher when some form of a request can not be satisfied due to the configured level of access.

exception KingPhisherPluginError(*plugin_name*, **args*, ***kwargs*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is raised by King Phisher to indicate an error regarding a particular plugin.

__init__(*plugin_name*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

exception KingPhisherResourceError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is raised by King Phisher when there is a problem relating to a resource such as it is missing, locked, inaccessible or otherwise invalid.

exception KingPhisherTimeoutError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is raised by King Phisher when some form of a request fails to complete within a specified time period.

1.8 find

This module provides a means by which data files distributed with the application can be found at run time by searching a configurable set of directories.

1.8.1 Data

DATA_DIRECTORY_NAME = 'king_phisher'
The name of the directory containing the King Phisher data.

ENV_VAR = 'KING_PHISHER_DATA_PATH'
The name of the environment variable which contains the search path.

1.8.2 Functions

data_path_append(*path*)
Add a directory to the data search path. The directory will be used by the *data_file()* and *data_directory()* functions.

Parameters **path** (*str*) – The path to add for searching.

data_directory(*name*, *access_mode=4*)
Locate a subdirectory in the data search path.

Parameters

- **name** (*str*) – The directory name to locate.
- **access_mode** (*int*) – The access that is required for the directory.

Returns The path to the located directory.

Return type str

data_file(*name*, *access_mode=4*)
Locate a data file by searching the directories specified in *ENV_VAR*. If *access_mode* is specified, it needs to be a value suitable for use with `os.access()`.

Parameters

- **name** (*str*) – The name of the file to locate.
- **access_mode** (*int*) – The access that is required for the file.

Returns The path to the located file.

Return type str

init_data_path(*directory=None*)
Add a directory to the data search path for either client or server data files.

Parameters **directory** (*str*) – The directory to add, either 'client' or 'server'.

1.9 geoip

This module uses GeoLite2 data created by MaxMind, available from http://www.maxmind.com.

1.9.1 Data

DB_RESULT_FIELDS
A tuple listing the fields that are required in database results.

1.9.2 Functions

download_geolite2_city_db(*dest*, *license*, *date=None*)
Download the GeoLite2 database, decompress it, and save it to disk.

Changed in version 1.16.0: Added the *license* and *date* parameters.

Parameters

- **dest** (*str*) – The file path to save the database to.
- **license** (*str*) – The MaxMind license key to use to download the database.
- **date** (datetime.date) – The date for which to download the database.

init_database(*database_file*)
Create and initialize the GeoLite2 database engine. This must be done before classes and functions in this module attempt to look up results. If the specified database file does not exist, a new copy will be downloaded.

Parameters **database_file** (*str*) – The GeoLite2 database file to use.

Returns The initialized GeoLite2 database object.

Return type geoip2.database.Reader

lookup(*ip*, *lang='en'*)
Lookup the geo location information for the specified IP from the configured GeoLite2 City database.

Parameters

- **ip** (*str*) – The IP address to look up the information for.
- **lang** (*str*) – The language to prefer for regional names.

Returns The geo location information as a dict. The keys are the values of *DB_RESULT_FIELDS*.

Return type dict

1.9.3 Classes

class Coordinates(*latitude*, *longitude*)
A named tuple for representing GPS coordinates.

latitude
Alias for field number 0

longitude
Alias for field number 1

class GeoLocation(*ip*, *lang='en'*, *result=None*)
The geographic location information for a given IP address. If *result* is not specified, *lookup()* will be used to obtain the information.

__geo_interface__
A simple implementation of the Python __geo_interface__ specification. This allows this object to be used with modules which also support this interface such as geojson.

Returns A dictionary describing a this location as a GeoJSON Point.

Return type dict

__init__(*ip*, *lang='en'*, *result=None*)

Parameters

- **ip** (*str*) – The IP address to look up geographic location data for.
- **lang** (*str*) – The language to prefer for regional names.
- **result** (*dict*) – A raw query result from a previous call to *lookup()*.

city

continent

coordinates

country

classmethod from_graphql(*ip*, *result*, *lang='en'*)

ip_address
The IPv4Address which this geographic location data describes.

postal_code

time_zone

1.10 ics

This module provides functionality for creating **RFC 5545** compliant iCalendar invite files.

1.10.1 Data

DAY_ABBREVIATIONS
The abbreviations of day names for use in icalendar.vRecur instances.

zoneinfo_path
The path to the directory which holds the IANA timezone data files.

1.10.2 Functions

get_timedelta_for_offset(*offset*)
Take a POSIX environment variable style offset from UTC and convert it into a timedelta instance suitable for use with the icalendar.

Parameters **offset** (*str*) – The offset from UTC such as "-5:00"

Returns The parsed offset.

Return type datetime.timedelta

get_tz_posix_env_var(*tz_name*)
Get the timezone information in the POSIX TZ environment variable format from the IANA timezone data files included in the pytz package.

Parameters **tz_name** (*str*) – The name of the timezone to get the environment variable for such as "America/New_York".

Returns The TZ environment variable string, if it is specified in the timezone data file.

Return type str

parse_tz_posix_env_var(*posix_env_var*)
Get the details regarding a timezone by parsing the POSIX style TZ environment variable.

Parameters **posix_env_var** (*str*) – The POSIX style TZ environment variable.

Returns The parsed TZ environment variable.

Return type *TimezoneOffsetDetails*

1.10.3 Classes

class Calendar(*organizer_email*, *start*, *summary*, *organizer_cn=None*, *description=None*, *duration='1h'*, *location=None*)

Bases: icalendar.cal.Calendar

An icalendar formatted event for converting to an ICS file and then sending in an email.

__init__(*organizer_email*, *start*, *summary*, *organizer_cn=None*, *description=None*, *duration='1h'*, *location=None*)

Parameters

- **organizer_email** (*str*) – The email of the event organizer.
- **start** (datetime.datetime) – The start time for the event.
- **summary** (*str*) – A short summary of the event.
- **organizer_cn** (*str*) – The name of the event organizer.
- **description** (*str*) – A more complete description of the event than what is provided by the *summary* parameter.
- **duration** (int, str, timedelta, *DurationAllDay*) – The events scheduled duration.
- **location** (*str*) – The location for the event.

add_attendee(*email*, *cn=None*, *rsvp=True*)

Add an attendee to the event. If the event is being sent via an email, the recipient should be added as an attendee.

Parameters

- **email** (*str*) – The attendee's email address.
- **cn** (*str*) – The attendee's common name.
- **rsvp** (*bool*) – Whether or not to request an RSVP response from the attendee.

to_ical(*encoding='utf-8'*, ***kwargs*)

Convert the calendar object to a string in the iCalendar format.

Returns The string representation of the data.

Return type str

class DurationAllDay(*days=1*)

Bases: object

A representation of a duration that can be used for an event to indicate that it takes place all day.

__init__(*days=1*)

Initialize self. See help(type(self)) for accurate signature.

class Timezone(*tz_name=None*)

Bases: icalendar.cal.Timezone

An icalendar formatted timezone with all properties populated for the specified zone.

__init__(*tz_name=None*)

Parameters **tz_name** (*str*) – The timezone to represent, if not specified it defaults to the local timezone.

class TimezoneOffsetDetails(*offset*, *offset_dst*, *dst_start*, *dst_end*)

Bases: tuple

A named tuple describing the details of a timezone's UTC offset and DST occurrence.

dst_end
Alias for field number 3

dst_start
Alias for field number 2

offset
Alias for field number 0

offset_dst
Alias for field number 1

1.11 ipaddress

This module provides functionality for dealing with an external "ipaddress" module in a Python 2 backwards compatible way. In Python 2 all string address arguments are converted to unicode which removes the ability to specify addresses as packed binary strings.

1.11.1 Functions

ip_address(*address*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP address. Either IPv4 or IPv6 addresses may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Address or IPv6Address object.

Raises:

ValueError: if the *address* passed isn't either a v4 or a v6 address

ip_network(*address*, *strict=True*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP network. Either IPv4 or IPv6 networks may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Network or IPv6Network object.

Raises:

ValueError: if the string passed isn't either a v4 or a v6 address. Or if the network has host bits set.

ip_interface(*address*)
Take an IP string/int and return an object of the correct type.

Args:

address: A string or integer, the IP address. Either IPv4 or IPv6 addresses may be supplied; integers less than 2**32 will be considered to be IPv4 by default.

Returns: An IPv4Interface or IPv6Interface object.

Raises:

ValueError: if the string passed isn't either a v4 or a v6 address.

Notes: The IPv?Interface classes describe an Address on a particular Network, so they're basically a combination of both the Address and Network classes.

is_loopback(*address*)

Check if an address is a loopback address or a common name for the loopback interface.

Parameters **address** (*str*) – The address to check.

Returns Whether or not the address is a loopback address.

Return type bool

is_valid(*address*)

Check that the string specified appears to be either a valid IPv4 or IPv6 address.

Parameters **address** (*str*) – The IP address to validate.

Returns Whether the IP address appears to be valid or not.

Return type bool

1.11.2 Classes

class IPv4Address(*address*)

Represent and manipulate single IPv4 Addresses.

is_link_local

Test if the address is reserved for link-local.

Returns: A boolean, True if the address is link-local per RFC 3927.

is_loopback

Test if the address is a loopback address.

Returns: A boolean, True if the address is a loopback per RFC 3330.

is_multicast

Test if the address is reserved for multicast use.

Returns: A boolean, True if the address is multicast. See RFC 3171 for details.

is_private

Test if this address is allocated for private networks.

Returns: A boolean, True if the address is reserved per iana-ipv4-special-registry.

is_reserved

Test if the address is otherwise IETF reserved.

Returns: A boolean, True if the address is within the reserved IPv4 Network range.

is_unspecified

Test if the address is unspecified.

Returns: A boolean, True if this is the unspecified address as defined in RFC 5735 3.

packed
The binary representation of this address.

class IPv4Network(*address*, *strict=True*)
This class represents and manipulates 32-bit IPv4 network + addresses..

Attributes: [examples for IPv4Network('192.0.2.0/27')] .network_address: IPv4Address('192.0.2.0') .hostmask: IPv4Address('0.0.0.31') .broadcast_address: IPv4Address('192.0.2.32') .netmask: IPv4Address('255.255.255.224') .prefixlen: 27

is_global
Test if this address is allocated for public networks.

Returns: A boolean, True if the address is not reserved per iana-ipv4-special-registry.

class IPv6Address(*address*)
Represent and manipulate single IPv6 Addresses.

ipv4_mapped
Return the IPv4 mapped address.

Returns: If the IPv6 address is a v4 mapped address, return the IPv4 mapped address. Return None otherwise.

is_global
Test if this address is allocated for public networks.

Returns: A boolean, true if the address is not reserved per iana-ipv6-special-registry.

is_link_local
Test if the address is reserved for link-local.

Returns: A boolean, True if the address is reserved per RFC 4291.

is_loopback
Test if the address is a loopback address.

Returns: A boolean, True if the address is a loopback address as defined in RFC 2373 2.5.3.

is_multicast
Test if the address is reserved for multicast use.

Returns: A boolean, True if the address is a multicast address. See RFC 2373 2.7 for details.

is_private
Test if this address is allocated for private networks.

Returns: A boolean, True if the address is reserved per iana-ipv6-special-registry.

is_reserved
Test if the address is otherwise IETF reserved.

Returns: A boolean, True if the address is within one of the reserved IPv6 Network ranges.

is_site_local
Test if the address is reserved for site-local.

Note that the site-local address space has been deprecated by RFC 3879. Use is_private to test if this address is in the space of unique local addresses as defined by RFC 4193.

Returns: A boolean, True if the address is reserved per RFC 3513 2.5.6.

is_unspecified
Test if the address is unspecified.

Returns: A boolean, True if this is the unspecified address as defined in RFC 2373 2.5.2.

packed
The binary representation of this address.

sixtofour
Return the IPv4 6to4 embedded address.

Returns: The IPv4 6to4-embedded address if present or None if the address doesn't appear to contain a 6to4 embedded address.

teredo
Tuple of embedded teredo IPs.

Returns: Tuple of the (server, client) IPs or None if the address doesn't appear to be a teredo address (doesn't start with 2001::/32)

class IPv6Network(*address*, *strict=True*)
This class represents and manipulates 128-bit IPv6 networks.

Attributes: [examples for IPv6('2001:db8::1000/124')] .network_address: IPv6Address('2001:db8::1000') .hostmask: IPv6Address('::f') .broadcast_address: IPv6Address('2001:db8::100f') .netmask: IPv6Address('ffff:ffff:ffff:ffff:ffff:ffff:ffff:fff0') .prefixlen: 124

hosts()
Generate Iterator over usable hosts in a network.

This is like __iter__ except it doesn't return the Subnet-Router anycast address.

is_site_local
Test if the address is reserved for site-local.

Note that the site-local address space has been deprecated by RFC 3879. Use is_private to test if this address is in the space of unique local addresses as defined by RFC 4193.

Returns: A boolean, True if the address is reserved per RFC 3513 2.5.6.

1.12 its

This module contains variables regarding the runtime environment in a standard location.

Note: This is a *"Clean Room" module* and is suitable for use during initialization.

1.12.1 Data

frozen = False
Whether or not the current environment is a frozen Windows build.

mocked = True
Whether or not certain objects are non-functional mock implementations. These are used for the purpose of generating documentation.

on_linux = True
Whether or not the current platform is Linux.

on_rtd = True
Whether or not the current platform is ReadTheDocs.

on_windows = False
Whether or not the current platform is Windows.

py_v2 = False
Whether or not the current Python version is 2.x.

py_v3 = True
Whether or not the current Python version is 3.x.

1.13 plugins

This module provides the core functionality necessary to support user provided plugins.

1.13.1 Functions

recursive_reload(*module*)
Reload *module* and if it is a package, recursively find and reload it's imported sub-modules.

Parameters **module** (*module*) – The module to reload.

Returns The reloaded module.

1.13.2 Classes

class OptionBase(*name*, *description*, *default=None*)
Bases: object

A base class for options which can be configured for plugins.

__init__(*name*, *description*, *default=None*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.

class OptionBoolean(*name*, *description*, *default=None*)
Bases: king_phisher.plugins.OptionBase

A plugin option which is represented with a boolean value.

__init__(*name*, *description*, *default=None*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.

class OptionEnum(*name*, *description*, *choices*, *default=None*)
Bases: king_phisher.plugins.OptionBase

A plugin option which is represented with an enumerable value.

__init__(*name*, *description*, *choices*, *default=None*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **choices** (*tuple*) – The supported values for this option.
- **default** – The default value of this option.

class OptionInteger(*name*, *description*, *default=None*)
Bases: *king_phisher.plugins.OptionBase*

A plugin option which is represented with an integer value.

__init__(*name*, *description*, *default=None*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.

class OptionString(*name*, *description*, *default=None*)
Bases: *king_phisher.plugins.OptionBase*

A plugin option which is represented with a string value.

__init__(*name*, *description*, *default=None*)

Parameters

- **name** (*str*) – The name of this option.
- **description** (*str*) – The description of this option.
- **default** – The default value of this option.

class PluginBase
Bases: *king_phisher.plugins.PluginBaseMeta*

A base class to be inherited by all plugins. Overriding or extending the standard __init__ method should be avoided to be compatible with future API changes. Instead the *initialize()* and *finalize()* methods should be overridden to provide plugin functionality.

__init__()
Initialize self. See help(type(self)) for accurate signature.

authors = ()
The tuple of authors who have provided this plugin.

classifiers = ()
An array containing optional classifier strings. These are free-formatted strings used to identify functionality.

config = None
The plugins configuration dictionary for storing the values of it's options.

description = None
A description of the plugin and what it does.

finalize()
This method can be overridden to perform any clean up action that the plugin needs such as closing files. It is called automatically by the manager when the plugin is disabled.

homepage = None
An optional homepage for the plugin.

initialize()
This method should be overridden to provide the primary functionality of the plugin. It is called automatically by the manager when the plugin is enabled.

Returns Whether or not the plugin successfully initialized itself.

Return type bool

options = []
A list of configurable option definitions for the plugin.

reference_urls = ()
An array containing optional reference URL strings.

req_min_py_version = None
The required minimum Python version for compatibility.

req_min_version = '1.3.0b0'
The required minimum version for compatibility.

req_packages = {}
A dictionary of required packages, keyed by the package name and a boolean value of it's availability.

req_platforms = ()
A tuple of case-insensitive supported platform names.

title = None
The title of the plugin.

version = '1.0'
The version identifier of this plugin.

class PluginBaseMeta
Bases: type

The meta class for *PluginBase* which provides additional class properties based on defined attributes.

compatibility
A generator which yields tuples of compatibility information based on the classes defined attributes. Each tuple contains three elements, a string describing the requirement, the requirements value, and a boolean indicating whether or not the requirement is met.

Returns Tuples of compatibility information.

is_compatible
Whether or not this plugin is compatible with this version of King Phisher. This can only be checked after the module is imported, so any references to non-existent classes in older versions outside of the class methods will still cause a load error.

Returns Whether or not this plugin class is compatible.

Return type bool

class PluginManagerBase(*path*, *args=None*, *library_path=AUTOMATIC*)
Bases: object

A managing object to control loading and enabling individual plugin objects.

__init__(*path*, *args=None*, *library_path=AUTOMATIC*)

Parameters

- **path** (*tuple*) – A tuple of directories from which to load plugins.
- **args** (*tuple*) – Arguments which should be passed to plugins when their class is initialized.
- **library_path** (*str*) – A path to use for plugins library dependencies. This value will be added to `sys.path` if it is not already included.

available
Return a tuple of all available plugins that can be loaded.

disable(*name*)
Disable a plugin by it's name. This call the plugins *PluginBase.finalize()* method to allow it to perform any clean up operations.

Parameters **name** (*str*) – The name of the plugin to disable.

enable(*name*)
Enable a plugin by it's name. This will create a new instance of the plugin modules "Plugin" class, passing it the arguments defined in `plugin_init_args`. A reference to the plugin instance is kept in *enabled_plugins*. After the instance is created, the plugins *initialize()* method is called.

Parameters **name** (*str*) – The name of the plugin to enable.

Returns The newly created instance.

Return type *PluginBase*

enabled_plugins = None
A dictionary of the enabled plugins and their respective instances.

get_plugin_path(*name*)
Get the path at which the plugin data resides. This is either the path to the single plugin file or a folder in the case that the plugin is a module. In either case, the path is an absolute path.

Parameters **name** (*str*) – The name of the plugin to get the path for.

Returns The path of the plugin data.

Return type str

install_packages(*packages*)
This function will take a list of Python packages and attempt to install them through pip to the *library_path*.

New in version 1.14.0.

Parameters **packages** (*list*) – list of python packages to install using pip.

Returns The process results from the command execution.

Return type *ProcessResults*

library_path = None
The path to a directory which is included for additional libraries. This path must be writable by the current user.

The default value is platform and Python-version (where X.Y is the major and minor versions of Python) dependant:

Linux `~/.local/lib/king-phisher/pythonX.Y/site-packages`

Windows `%LOCALAPPDATA%\king-phisher\lib\pythonX.Y\site-packages`

load(*name*, *reload_module=False*)
Load a plugin into memory, this is effectively the Python equivalent of importing it. A reference to the plugin class is kept in *loaded_plugins*. If the plugin is already loaded, no changes are made.

Parameters

- **name** (*str*) – The name of the plugin to load.
- **reload_module** (*bool*) – Reload the module to allow changes to take affect.

Returns The plugin class.

load_all(*on_error=None*)
Load all available plugins. Exceptions while loading specific plugins are ignored. If *on_error* is specified, it will be called from within the exception handler when a plugin fails to load correctly. It will be called with two parameters, the name of the plugin and the exception instance.

Parameters **on_error** (*function*) – A call back function to call when an error occurs while loading a plugin.

load_module(*name*, *reload_module=False*)
Load the module which contains a plugin into memory and return the entire module object.

Parameters

- **name** (*str*) – The name of the plugin module to load.
- **reload_module** (*bool*) – Reload the module to allow changes to take affect.

Returns The plugin module.

loaded_plugins = None
A dictionary of the loaded plugins and their respective modules.

shutdown()
Unload all plugins and perform additional clean up operations.

uninstall(*name*)
Uninstall a plugin by first unloading it and then delete it's data on disk. The plugin data on disk is found with the *get_plugin_path()* method.

Parameters **name** (*str*) – The name of the plugin to uninstall.

Returns Whether or not the plugin was successfully uninstalled.

Return type bool

unload(*name*)
Unload a plugin from memory. If the specified plugin is currently enabled, it will first be disabled before being unloaded. If the plugin is not already loaded, no changes are made.

Parameters **name** (*str*) – The name of the plugin to unload.

unload_all()
Unload all available plugins. Exceptions while unloading specific plugins are ignored.

class Requirements(*items*)
Bases: collections.abc.Mapping

This object servers to map requirements specified as strings to their respective values. Once the requirements are defined, this class can then be used to evaluate them in an effort to determine which requirements are met and which are not.

__init__(*items*)

Parameters **items** (*dict*) – A dictionary or two-dimensional array mapping requirement names to their respective values.

compatibility_iter()
Iterate over each of the requirements, evaluate them and yield a tuple regarding them.

is_compatible
Whether or not all requirements are met.

to_dict()
Return a dictionary representing the requirements.

1.14 security_keys

This module provides functionality for working with security keys that are used for data integrity checks. Verification is performed using ECDSA keys.

1.14.1 Data

ecdsa_curves
A dictionary of `ecdsa.curves.Curve` objects keyed by their `ecdsa` and OpenSSL compatible names.

1.14.2 Functions

openssl_decrypt_data(*ciphertext*, *password*, *digest='sha256'*, *encoding='utf-8'*)
Decrypt *ciphertext* in the same way as OpenSSL. For the meaning of *digest* see the *openssl_derive_key_and_iv()* function documentation.

Note: This function can be used to decrypt ciphertext created with the `openssl` command line utility.

```
openssl enc -e -aes-256-cbc -in file -out file.enc -md sha256
```

Parameters

- **ciphertext** (*bytes*) – The encrypted data to decrypt.
- **password** (*str*) – The password to use when deriving the decryption key.
- **digest** (*str*) – The name of hashing function to use to generate the key.
- **encoding** (*str*) – The name of the encoding to use for the password.

Returns The decrypted data.

Return type bytes

openssl_derive_key_and_iv(*password*, *salt*, *key_length*, *iv_length*, *digest='sha256'*, *encoding='utf-8'*)
Derive an encryption key and initialization vector (IV) in the same way as OpenSSL.

Note: Different versions of OpenSSL use a different default value for the *digest* function used to derive keys and initialization vectors. A specific one can be used by passing the `-md` option to the `openssl` command which corresponds to the *digest* parameter of this function.

Parameters

- **password** (*str*) – The password to use when deriving the key and IV.
- **salt** (*bytes*) – A value to use as a salt for the operation.
- **key_length** (*int*) – The length in bytes of the key to return.
- **iv_length** (*int*) – The length in bytes of the IV to return.
- **digest** (*str*) – The name of hashing function to use to generate the key.
- **encoding** (*str*) – The name of the encoding to use for the password.

Returns The key and IV as a tuple.

Return type tuple

1.14.3 Classes

class SecurityKeys

Bases: object

The security keys that are installed on the system. These are then used to validate the signatures of downloaded files to ensure they have not been corrupted or tampered with.

Note: Keys are first loaded from the security.json file included with the application source code and then from an optional security.local.json file. Keys loaded from the optional file can not over write keys loaded from the system file.

__init__()

Initialize self. See help(type(self)) for accurate signature.

keys = None

The dictionary of the loaded security keys, keyed by their identity string.

verify(*key_id*, *data*, *signature*)

Verify the data with the specified signature as signed by the specified key. This function will raise an exception if the verification fails for any reason, including if the key can not be found.

Parameters

- **key_id** (*str*) – The key's identifier.
- **data** (*bytes*) – The data to verify against the signature.
- **signature** (*bytes*) – The signature of the data to verify.

verify_dict(*data*, *signature_encoding='base64'*)

Verify the signed dictionary, using the key specified within the 'signed-by' key. This function will raise an exception if the verification fails for any reason, including if the key can not be found.

Parameters

- **key_id** (*str*) – The key's identifier.

- **data** (*bytes*) – The data to verify against the signature.
- **signature** (*bytes*) – The signature of the data to verify.

class SigningKey(**args*, ***kwargs*)

Bases: ecdsa.keys.SigningKey, object

classmethod from_dict(*value*, *encoding='base64'*, ***kwargs*)

Load the signing key from the specified dict object.

Parameters

- **value** (*dict*) – The dictionary to load the key data from.
- **encoding** (*str*) – The encoding of the required 'data' key.
- **kwargs** (*dict*) – Additional key word arguments to pass to the class on initialization.

Returns The new signing key.

Return type *SigningKey*

classmethod from_file(*file_path*, *password=None*, *encoding='utf-8'*)

Load the signing key from the specified file. If *password* is specified, the file is assumed to have been encrypted using OpenSSL with aes-256-cbc as the cipher and sha256 as the message digest. This uses *openssl_decrypt_data()* internally for decrypting the data.

Parameters

- **file_path** (*str*) – The path to the file to load.
- **password** (*str*) – An optional password to use for decrypting the file.
- **encoding** (*str*) – The encoding of the data.

Returns A tuple of the key's ID, and the new *SigningKey* instance.

Return type tuple

id = None

An optional string identifier for this key instance.

sign_dict(*data*, *signature_encoding='base64'*)

Sign a dictionary object. The dictionary will have a 'signature' key added is required by the *VerifyingKey.verify_dict()* method. To serialize the dictionary to data suitable for the operation the json.dumps() function is used and the resulting data is then UTF-8 encoded.

Parameters

- **data** (*dict*) – The dictionary of data to sign.
- **signature_encoding** (*str*) – The encoding name of the signature data.

Returns The dictionary object is returned with the 'signature' key added.

class VerifyingKey(**args*, ***kwargs*)

Bases: ecdsa.keys.VerifyingKey, object

classmethod from_dict(*value*, *encoding='base64'*, ***kwargs*)

Load the verifying key from the specified dict object.

Parameters

- **value** (*dict*) – The dictionary to load the key data from.
- **encoding** (*str*) – The encoding of the required 'data' key.
- **kwargs** (*dict*) – Additional key word arguments to pass to the class on initialization.

Returns The new verifying key.

Return type *VerifyingKey*

id = None
An optional string identifier for this key instance.

verify_dict(*data*, *signature_encoding='base64'*)
Verify a signed dictionary object. The dictionary must have a 'signature' key as added by the *SigningKey.sign_dict()* method. To serialize the dictionary to data suitable for the operation the `json.dumps()` function is used and the resulting data is then UTF-8 encoded.

Parameters

- **data** (*dict*) – The dictionary of data to verify.
- **signature_encoding** (*str*) – The encoding name of the signature data.

1.15 serializers

This module provides a standardized interface for serializing objects using different formats. The Serializers provided by this module are organized by their format into different classes. The necessary methods for utilizing them are all `classmethod`'s making it unnecessary to create an instance of any of them.

1.15.1 Functions

from_elementtree_element(*element*, *require_type=True*)
Load a value from an `xml.etree.ElementTree.SubElement` instance. If *require_type* is True, then the element must specify an acceptable value via the "type" attribute. If *require_type* is False and no type attribute is specified, the value is returned as a string.

Parameters

- **element** (`xml.etree.ElementTree.Element`) – The element to load a value from.
- **require_type** (*bool*) – Whether or not to require type information.

Returns The deserialized value from the element.

to_elementtree_subelement(*parent*, *tag*, *value*, *attrib=None*)
Serialize *value* to an `xml.etree.ElementTree.SubElement` with appropriate information describing it's type. If *value* is not of a supported type, a `TypeError` will be raised.

Parameters

- **parent** (`xml.etree.ElementTree.Element`) – The parent element to associate this subelement with.
- **tag** (*str*) – The name of the XML tag.
- **value** – The value to serialize to an XML element.
- **attrib** (*dict*) – Optional attributes to include in the element.

Returns The newly created XML element, representing *value*.

Return type `xml.etree.ElementTree.Element`

1.15.2 Classes

class JSON
Bases: *king_phisher.serializers.Serializer*

classmethod dumps(*data*, *pretty=True*)
Convert a Python object to a JSON encoded string.

Parameters

- **data** – The object to encode.
- **pretty** (*bool*) – Set options to make the resulting JSON data more readable.

Returns The encoded data.

Return type str

classmethod loads(*data*, *strict=True*)
Load JSON encoded data.

Parameters

- **data** (*str*) – The encoded data to load.
- **strict** (*bool*) – Do not try remove trailing commas from the JSON data.

Returns The Python object represented by the encoded data.

class MsgPack
Bases: *king_phisher.serializers.Serializer*

classmethod dumps(*data*)
Convert a Python object to a MsgPack encoded `bytes` instance.

Parameters

- **data** – The object to encode.
- **pretty** (*bool*) – Set options to make the resulting JSON data more readable.

Returns The encoded data.

Return type str

classmethod loads(*data*)
Load MsgPack encoded data.

Parameters **data** (*bytes*) – The encoded data to load.

Returns The Python object represented by the encoded data.

class Serializer
Bases: `king_phisher.serializers._SerializerMeta`

The base class for serializer objects of different formats and protocols. These serializers are extended using a King Phisher-specific protocol for serializing additional types, most notably Python's `datetime.datetime` type.

Note: None of the serializers handle Python 3's `bytes` type. These objects will be treated as strings and silently converted.

---

classmethod dump(*data*, *file_h*, **args*, ***kwargs*)
Write a Python object to a file by encoding it with this serializer.

Parameters

- **data** – The object to encode.
- **file_h** (*file*) – The file to write the encoded string to.

encoding = 'utf-8'
The encoding which this serializer uses for handling strings.

classmethod load(*file_h*, **args*, ***kwargs*)
Load encoded data from the specified file.

Parameters

- **file_h** (*file*) – The file to read and load encoded data from.
- **strict** (*bool*) – Do not try remove trailing commas from the JSON data.

Returns The Python object represented by the encoded data.

1.16 sms

This module provides functionality for sending free SMS messages by emailing a carriers SMS gateway.

1.16.1 Data

CARRIERS
A dictionary for mapping carrier names to SMS via email gateways.

DEFAULT_FROM_ADDRESS
The default email address to use in the from field.

1.16.2 Functions

get_smtp_servers(*domain*)
Get the SMTP servers for the specified domain by querying their MX records.

Parameters **domain** (*str*) – The domain to look up the MX records for.

Returns The smtp servers for the specified domain.

Return type list

lookup_carrier_gateway(*carrier*)
Lookup the SMS gateway for the specified carrier. Normalization on the carrier name does take place and if an invalid or unknown value is specified, None will be returned.

Parameters **carrier** (*str*) – The name of the carrier to lookup.

Returns The SMS gateway for the specified carrier.

Return type str

send_sms(*message_text*, *phone_number*, *carrier*, *from_address=None*)
Send an SMS message by emailing the carriers SMS gateway. This method requires no money however some networks are blocked by the carriers due to being flagged for spam which can cause issues.

Parameters

- **message_text** (*str*) – The message to send.

- **phone_number** (*str*) – The phone number to send the SMS to.
- **carrier** (*str*) – The cellular carrier that the phone number belongs to.
- **from_address** (*str*) – The optional address to display in the 'from' field of the SMS.

Returns This returns the status of the sent messsage.

Return type bool

1.17 smtp_server

This module provides a SMTP server that can be used for debugging purposes.

1.17.1 Classes

class BaseSMTPServer(*localaddr*, *remoteaddr=None*)

Bases: smtpd.SMTPServer, object

An SMTP server useful for debugging. Messages handled by this server are not forwarded anywhere.

__init__(*localaddr*, *remoteaddr=None*)

Parameters

- **localaddr** (*tuple*) – The local address to bind to.
- **remoteaddr** (*tuple*) – The remote address to use as an upstream SMTP relayer.

serve_forever()

Process requests until BaseSMTPServer.shutdown() is called.

1.18 spf

This module provides functionality for checking published Sender Policy Framework (SPF) records. SPF is defined in **RFC 7208**.

1.18.1 Data

DEFAULT_DNS_TIMEOUT = 10

The default number of seconds to wait for a query response from the DNS server.

MACRO_REGEX

A regular expression which matches SPF record macros.

MAX_QUERIES = 10

The maximum number of DNS queries allowed to take place during evaluation as defined within section 4.6.4 of **RFC 7208**.

MAX_QUERIES_VOID = inf

The maximum number of DNS queries allowed to either return with rcode 0 and no answers or rcode 3 (Name Error) as defined within section 4.6.4 of **RFC 7208**.

QUALIFIERS

A dict object keyed with the qualifier symbols to their readable values.

1.18.2 Functions

check_host(*ip*, *domain*, *sender=None*, *timeout=10*)
Analyze the Sender Policy Framework of a domain by creating a *SenderPolicyFramework* instance and returning the result of *SenderPolicyFramework.check_host()*.

Parameters

- **ip** (str, ipaddress.IPv4Address, ipaddress.IPv6Address) – The IP address of the host sending the message.
- **domain** (*str*) – The domain to check the SPF policy of.
- **sender** (*str*) – The "MAIL FROM" identity of the message being sent.
- **timeout** (*int*) – The timeout for DNS queries.

Returns The result of the SPF policy if one can be found or None.

Return type None, str

validate_record(*ip*, *domain*, *sender=None*)
Check if an SPF record exists for the domain and can be parsed by this module.

Returns Whether the record exists and is parsable or not.

Return type bool

1.18.3 Classes

class SenderPolicyFramework(*ip*, *domain*, *sender=None*, *timeout=10*)
Analyze the Sender Policy Framework configuration for a domain to determine if an IP address is authorized to send messages on it's behalf. The exp modifier defined in section 6.2 of the RFC is not supported.

__init__(*ip*, *domain*, *sender=None*, *timeout=10*)

Parameters

- **ip** (str, ipaddress.IPv4Address, ipaddress.IPv6Address) – The IP address of the host sending the message.
- **domain** (*str*) – The domain to check the SPF policy of.
- **sender** (*str*) – The "MAIL FROM" identity of the message being sent.
- **timeout** (*int*) – The timeout for DNS queries.

check_host()
Check the SPF policy described by the object. The string representing the matched policy is returned if an SPF policy exists, otherwise None will be returned if no policy is defined.

Returns The result of the SPF policy described by the object.

Return type None, str

expand_macros(*value*, *ip*, *domain*, *sender*)
Expand a string based on the macros it contains as specified by section 7 of **RFC 7208**.

Parameters

- **value** (*str*) – The string containing macros to expand.
- **ip** (str, ipaddress.IPv4Address, ipaddress.IPv6Address) – The IP address to use when expanding macros.

- **domain** (*str*) – The domain name to use when expanding macros.
- **sender** (*str*) – The email address of the sender to use when expanding macros.

Returns The string with the interpreted macros replaced within it.

Return type str

match

matches = None
A list of *SPFMatch* instances showing the path traversed to identify a matching directive. Multiple entries in this list are present when include directives are used and a match is found within the body of one. The list is ordered from the top level domain to the matching record.

records = None
A `collections.OrderedDict` of all the SPF records that were resolved. This would be any records resolved due to an "include" directive in addition to the top level domain.

timeout = None
The human readable policy result, one of the `SPFResult` constants'.

class SPFDirective(*mechanism*, *qualifier*, *rvalue=None*)
A class representing a single directive within a sender policy framework record.

__init__(*mechanism*, *qualifier*, *rvalue=None*)

Parameters

- **mechanism** (*str*) – The SPF mechanism that this directive uses.
- **qualifier** (*str*) – The qualifier value of the directive in it's single character format.
- **rvalue** (*str*) – The optional rvalue for directives which use them.

class SPFMatch(*record*, *directive*)
A simple container to associate a matched directive with it's record.

__init__
Initialize self. See help(type(self)) for accurate signature.

class SPFRecord(*directives*, *domain=None*)
A class representing a parsed Sender Policy Framework record with all of its directives.

__init__(*directives*, *domain=None*)

Parameters

- **directives** (*list*) – A list of *SPFDirective* instances.
- **domain** (*str*) – The domain with which this record is associated with.

1.18.4 Exceptions

exception SPFError(*message*)
Bases: `Exception`

Base exception for errors raised by this module.

exception SPFTempError(*message*)
Bases: *king_phisher.spf.SPFError*

Exception indicating that the verification process encountered a transient (generally DNS) error while performing the check. Described in section 2.6.6 of **RFC 7208**.

exception SPFTimeOutError(*message*)
Bases: *king_phisher.spf.SPFTempError*

Exception indicating that a timeout occurred while querying the DNS server. This is normally caused when the client can't communicate with the DNS server.

exception SPFParseError(*message*)
Bases: *king_phisher.spf.SPFPermError*

Exception indicating that the domains published records could not be correctly parsed.

exception SPFPermError(*message*)
Bases: *king_phisher.spf.SPFError*

Exception indicating that the domains published records could not be correctly interpreted. Described in section 2.6.7 of **RFC 7208**.

1.19 ssh_forward

This module provides functionality for forwarding network services over SSH.

1.19.1 Classes

class SSHTCPForwarder(*server*, *username*, *password*, *remote_server*, *local_port=0*, *private_key=None*, *missing_host_key_policy=None*)
Bases: threading.Thread

Open an SSH connection and forward TCP traffic through it to a remote host. A private key for authentication can be specified as a string either by it's OpenSSH fingerprint, as a file (prefixed with "file:"), or a raw key string (prefixed with "key:"). If no *missing_host_key_policy* is specified, paramiko.client.AutoAddPolicy will be used to accept all host keys.

Note: This is a threading.Thread object and needs to be started with a call to start() after it is initialized.

__init__(*server*, *username*, *password*, *remote_server*, *local_port=0*, *private_key=None*, *missing_host_key_policy=None*)

Parameters

- **server** (*tuple*) – The SSH server to connect to.
- **username** (*str*) – The username to authenticate with.
- **password** (*str*) – The password to authenticate with.
- **remote_server** (*tuple*) – The remote server to connect to through the specified SSH server.
- **local_port** (*int*) – The local port to forward, if not set a random one will be used.
- **private_key** (*str*) – An RSA key to prefer for authentication.
- **missing_host_key_policy** – The policy to use for missing host keys.

local_server
A tuple representing the local address of the listening service which is forwarding traffic to the specified remote host.

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

start()
Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

1.19.2 Exceptions

class KingPhisherSSHKeyError(*message=''*)
Bases: *king_phisher.errors.KingPhisherError*

An exception that is thrown when there is a problem resolving a users SSH key file. The *message* attribute is formatted to be displayed to the user via a dialog.

1.20 startup

This module provides generic functions for the early initialization of the project's environment. This is primarily used for the management of external dependencies.

Note: This is a *"Clean Room" module* and is suitable for use during initialization.

1.20.1 Functions

argp_add_client(*parser*)
Add client-specific arguments to a new argparse.ArgumentParser instance.

Parameters **parser** (argparse.ArgumentParser) – The parser to add arguments to.

argp_add_default_args(*parser*, *default_root=''*)
Add standard arguments to a new argparse.ArgumentParser instance. Used to add the utilities argparse options to the wrapper for display.

Parameters

- **parser** (argparse.ArgumentParser) – The parser to add arguments to.
- **default_root** (*str*) – The default root logger to specify.

argp_add_server(*parser*)
Add server-specific arguments to a new argparse.ArgumentParser instance.

Parameters **parser** (argparse.ArgumentParser) – The parser to add arguments to.

pipenv_entry(*parser*, *entry_point*)
Run through startup logic for a Pipenv script (see Pipenv: Custom Script Shortcuts for more information). This sets up a basic stream logging configuration, establishes the Pipenv environment and finally calls the actual entry point using os.execve().

Note: Due to the use of `os.execve()`, this function does not return.

Note: Due to the use of `os.execve()` and `os.EX_*` exit codes, this function is not available on Windows.

Parameters

- **parser** – The argument parser to use. Arguments are added to it and extracted before passing the remainder to the entry point.
- **entry_point** (*str*) – The name of the entry point using Pipenv.

run_process(*process_args*, *cwd=None*, *tee=False*, *encoding='utf-8'*)

Run a subprocess, wait for it to complete and return a *ProcessResults* object. This function differs from *start_process()* in the type it returns and the fact that it always waits for the subprocess to finish before returning.

Changed in version 1.15.0: Added the *tee* parameter.

Parameters

- **process_args** (*tuple*) – The arguments for the processes including the binary.
- **cwd** (*bool*) – An optional current working directory to use for the process.
- **tee** (*bool*) – Whether or not to display the console output while the process is running.
- **encoding** (*str*) – The encoding to use for strings.

Returns The results of the process including the status code and any text printed to stdout or stderr.

Return type *ProcessResults*

start_process(*process_args*, *wait=True*, *cwd=None*)

Start a subprocess and optionally wait for it to finish. If not **wait**, a handle to the subprocess is returned instead of `True` when it exits successfully. This function differs from *run_process()* in that it optionally waits for the subprocess to finish, and can return a handle to it.

Parameters

- **process_args** (*tuple*) – The arguments for the processes including the binary.
- **wait** (*bool*) – Whether or not to wait for the subprocess to finish before returning.
- **cwd** (*str*) – The optional current working directory.

Returns If **wait** is set to True, then a boolean indication success is returned, else a handle to the subprocess is returened.

which(*program*)

Examine the `PATH` environment variable to determine the location for the specified program. If it can not be found None is returned. This is fundamentally similar to the Unix utility of the same name.

Parameters **program** (*str*) – The name of the program to search for.

Returns The absolute path to the program if found.

Return type str

1.20.2 Classes

class ProcessResults (*stdout*, *stderr*, *status*)
A named tuple for holding the results of an executed external process.

stdout
A string containing the data the process wrote to stdout.

stderr
A string containing the data the process wrote to stderr.

status
An integer representing the process's exit code.

1.21 templates

This module provides base classes for the Jinja2 environments used throughout the application.

1.21.1 Classes

class FindFileSystemLoader
Bases: `jinja2.loaders.BaseLoader`

A `BaseLoader` which loads templates by name from the file system. Templates are searched for using the *data_file()* function.

get_source (*environment*, *template*)
Get the template source, filename and reload helper for a template. It's passed the environment and template name and has to return a tuple in the form `(source, filename, uptodate)` or raise a *TemplateNotFound* error if it can't locate the template.

The source part of the returned tuple must be the source of the template as unicode string or a ASCII bytestring. The filename should be the name of the file on the filesystem if it was loaded from there, otherwise *None*. The filename is used by python for the tracebacks if no loader extension is used.

The last item in the tuple is the *uptodate* function. If auto reloading is enabled it's always called to check if the template changed. No arguments are passed so the function must store the old state somewhere (for example in a closure). If it returns *False* the template will be reloaded.

class TemplateEnvironmentBase (*loader=None*, *global_vars=None*)
Bases: `jinja2.environment.Environment`

A configured Jinja2 `Environment` with additional filters and default settings.

__init__ (*loader=None*, *global_vars=None*)

Parameters

- **loader** (`jinja2.BaseLoader`) – The loader to supply to the environment.
- **global_vars** (*dict*) – Additional global variables for the environment.

from_file (*path*, ***kwargs*)
A convenience method to load template data from a specified file, passing it to `from_string()`.

> **Warning:** Because this method ultimately passes the template data to the `from_string()` method, the data will not be automatically escaped based on the file extension as it would be when using `get_template()`.

Parameters

- **path** (*str*) – The path from which to load the template data.
- **kwargs** – Additional keyword arguments to pass to `from_string()`.

join_path(*template*, *parent*)
Over ride the default `jinja2.Environment.join_path()` method to explicitly specifying relative paths by prefixing the path with either "./" or "../".

Parameters

- **template** (*str*) – The path of the requested template file.
- **parent** (*str*) – The path of the template file which requested the load.

Returns The new path to the template.

Return type str

standard_variables
Additional standard variables that can optionally be used in templates.

class MessageTemplateEnvironment(**args*, ***kwargs*)
Bases: *king_phisher.templates.TemplateEnvironmentBase*

A configured Jinja2 environment for formatting messages.

MODE_ANALYZE = 1

MODE_PREVIEW = 0

MODE_SEND = 2

attachment_images = None
A dictionary collecting the images that are going to be embedded and sent inline in the message.

set_mode(*mode*)
Set the operation mode for the environment. Valid values are the MODE_* constants.

Parameters **mode** (*int*) – The operation mode.

1.22 testing

This module provides supporting functionality for the included application unit tests.

1.22.1 Data

TEST_MESSAGE_TEMPLATE
A string representing a message template that can be used for testing.

TEST_MESSAGE_TEMPLATE_INLINE_IMAGE
A string with the path to a file used as an inline image in the *TEST_MESSAGE_TEMPLATE*.

1.22.2 Classes

class KingPhisherTestCase(**args*, ***kwargs*)
Bases: smoke_zephyr.utilities.TestCase

This class provides additional functionality over the built in unittest.TestCase object, including better compatibility for methods across Python 2.x and Python 3.x.

assertHasAttribute(*obj*, *attribute*, *msg=None*)
Test that *obj* has the named *attribute*.

assertIsEmpty(*obj*, *msg=None*)
Test that *obj* is empty as determined by len().

assertIsNotEmpty(*obj*, *msg=None*)
Test that *obj* is not empty as determined by len().

assertIsSubclass(*obj*, *cls*, *msg=None*)
Test that *obj* is a subclass of *cls* (which can be a class or a tuple of classes as supported by issubclass()).

class KingPhisherServerTestCase(**args*, ***kwargs*)
Bases: *king_phisher.testing.KingPhisherTestCase*

This class can be inherited to automatically set up a King Phisher server instance configured in a way to be suitable for testing purposes.

assertHTTPStatus(*http_response*, *status*)
Check an HTTP response to ensure that the correct HTTP status code is specified.

Parameters

- **http_response** (httplib.HTTPResponse) – The response object to check.
- **status** (*int*) – The status to check for.

assertRPCPermissionDenied(*method*, **args*, ***kwargs*)
Assert that the specified RPC method fails with a *KingPhisherPermissionError* exception.

Parameters **method** – The RPC method that is to be tested

http_request(*resource*, *method='GET'*, *include_id=True*, *body=None*, *headers=None*)
Make an HTTP request to the specified resource on the test server.

Parameters

- **resource** (*str*) – The resource to send the request to.
- **method** (*str*) – The HTTP method to use for the request.
- **include_id** (*bool*) – Whether to include the the id parameter.
- **body** (*dict, str*) – The data to include in the body of the request.
- **headers** (*dict*) – The headers to include in the request.

Returns The servers HTTP response.

Return type httplib.HTTPResponse

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

web_root_files (*limit=None*, *include_templates=True*)
A generator object that yields valid files which are contained in the web root of the test server instance. This can be used to find resources which the server should process as files. The function will fail if no files can be found in the web root.

Parameters

- **limit** (*int*) – A limit to the number of files to return.
- **include_templates** (*bool*) – Whether or not to include files that might be templates.

1.23 ua_parser

This module provides functionality for parsing browser user agents to extract information from them.

1.23.1 Functions

parse_user_agent (*user_agent*)
Parse a user agent string and return normalized information regarding the operating system.

Parameters **user_agent** (*str*) – The user agent to parse.

Returns A parsed user agent, None is returned if the data can not be processed.

Return type *UserAgent*

1.23.2 Classes

class UserAgent
A parsed representation of the information available from a browsers user agent string. Only the *os_name* attribute is guaranteed to not be None.

os_name
The *OSFamily* constant of the name of the operating system.

os_version
The version of the operating system.

os_arch
The *OSArch* constant of the architecture of the operating system.

1.24 utilities

This module collects various useful utility functions that are used throughout the application.

1.24.1 Functions

argp_add_args (*parser*, *default_root=''*)
Add standard arguments to a new `argparse.ArgumentParser` instance for configuring logging options from the command line and displaying the version information.

Note: This function installs a hook to *parser.parse_args* to automatically handle options which it adds. This includes setting up a stream logger based on the added options.

Parameters

- **parser** (argparse.ArgumentParser) – The parser to add arguments to.
- **default_root** (*str*) – The default root logger to specify.

assert_arg_type(*arg*, *arg_type*, *arg_pos=1*, *func_name=None*)

Check that an argument is an instance of the specified type and if not raise a TypeError exception with a meaningful message. If *func_name* is not specified, it will be determined by examining the stack.

Parameters

- **arg** – The argument to check.
- **arg_type** (*list, tuple, type*) – The type or sequence of types that *arg* can be.
- **arg_pos** (*int*) – The position of the argument in the function.
- **func_name** (*str*) – The name of the function the argument is for.

configure_stream_logger(*logger*, *level=None*)

Configure the default stream handler for logging messages to the console. This also configures the basic logging environment for the application.

Parameters

- **logger** (*str*) – The logger to add the stream handler for.
- **level** (*None, int, str*) – The level to set the logger to, will default to WARNING if no level is specified.

Returns The new configured stream handler.

Return type logging.StreamHandler

datetime_local_to_utc(*dt*)

Convert a datetime.datetime instance from the local time to UTC time.

Parameters **dt** (datetime.datetime) – The time to convert from local to UTC.

Returns The time converted to the UTC timezone.

Return type datetime.datetime

datetime_utc_to_local(*dt*)

Convert a datetime.datetime instance from UTC time to the local time.

Parameters **dt** (datetime.datetime) – The time to convert from UTC to local.

Returns The time converted to the local timezone.

Return type datetime.datetime

format_datetime(*dt*, *encoding='utf-8'*)

Format a date time object into a string. If the object *dt* is not an instance of datetime.datetime then an empty string will be returned.

Parameters

- **dt** (datetime.datetime) – The object to format.

- **encoding** (*str*) – The encoding to use to coerce the return value into a unicode string.

Returns The string representing the formatted time.

Return type str

is_valid_email_address(*email_address*)
Check that the string specified appears to be a valid email address.

Parameters **email_address** (*str*) – The email address to validate.

Returns Whether the email address appears to be valid or not.

Return type bool

make_message_uid(*upper=True*, *lower=True*, *digits=True*)
Creates a random string of specified character set to be used as a message id. At least one of *upper*, *lower*, or *digits* must be `True`.

Parameters

- **upper** (*bool*) – Include upper case characters in the UID.
- **lower** (*bool*) – Include lower case characters in the UID.
- **digits** (*bool*) – Include digits in the UID.

Returns String of characters from the random_string function.

Return type str

make_webrelpath(*path*)
Forcefully make *path* into a web-suitable relative path. This will strip off leading and trailing directory separators.

New in version 1.14.0.

Parameters **path** (*str*) – The path to convert into a web-suitable relative path.

Returns The converted path.

Return type str

make_visit_uid()
Creates a random string of characters and numbers to be used as a visit id.

Returns String of characters from the random_string function.

Return type str

nonempty_string(*value*)
Convert *value* into either a non-empty string or None. This will also strip leading and trailing whitespace.

Parameters **value** (*str*) – The value to convert.

Returns Either the non-empty string or None.

open_uri(*uri*)
Open a URI in a platform intelligent way. On Windows this will use 'cmd.exe /c start' and on Linux this will use gvfs-open or xdg-open depending on which is available. If no suitable application can be found to open the URI, a RuntimeError will be raised.

Parameters **uri** (*str*) – The URI to open.

parse_datetime(*ts*)
Parse a time stamp into a `datetime.datetime` instance. The time stamp must be in a compatible format, as would have been returned from the *format_datetime()* function.

Parameters **ts** (*str*) – The timestamp to parse.

Returns The parsed timestamp.

Return type datetime.datetime

password_is_complex(*password*, *min_len=12*)
Check that the specified string meets standard password complexity requirements. :param str password: The password to validate. :param int min_len: The mininum length the password should be. :return: Whether the strings appears to be complex or not. :rtype: bool

random_string(*size*, *charset=None*)
Generate a random string consisting of uppercase letters, lowercase letters and numbers of the specified size.

Parameters **size** (*int*) – The size of the string to make.

Returns The string containing the random characters.

Return type str

random_string_lower_numeric(*size*)
Generate a random string consisting of lowercase letters and numbers of the specified size.

Parameters **size** (*int*) – The size of the string to make.

Returns The string containing the random characters.

Return type str

switch(*value*, *comp=<built-in function eq>*, *swapped=False*)
A pure Python implementation of a switch case statement. *comp* will be used as a comparison function and passed two arguments of *value* and the provided case respectively.

Switch case example usage:

```
for case in switch(2):
    if case(1):
        print('case 1 matched!')
        break
    if case(2):
        print('case 2 matched!')
        break
else:
    print('no cases were matched')
```

Parameters

- **value** – The value to compare in each of the case statements.
- **comp** – The function to use for comparison in the case statements.
- **swapped** – Whether or not to swap the arguments to the *comp* function.

Returns A function to be called for each case statement.

validate_json_schema(*data*, *schema_file_id*)
Validate the specified data against the specified schema. The schema file will be searched for and loaded based on it's id. If the validation fails a ValidationError will be raised.

Parameters

- **data** – The data to validate against the schema.
- **schema_file_id** – The id of the schema to load.

1.24.2 Classes

class Event
Bases: threading.Event

clear()
Reset the internal flag to false.

Subsequently, threads calling wait() will block until set() is called to set the internal flag to true again.

set()
Set the internal flag to true.

All threads waiting for it to become true are awakened. Threads that call wait() once the flag is true will not block at all.

wait(*timeout=None*)
Block until the internal flag is true.

If the internal flag is true on entry, return immediately. Otherwise, block until another thread calls set() to set the flag to true, or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof).

This method returns the internal flag on exit, so it will always return True except if a timeout is given and the operation times out.

class FreezableDict(**args*, ***kwargs*)
Bases: collections.OrderedDict

A dictionary that can be frozen to prevent further editing. Useful for debugging. If any function tries to edit a frozen dictionary, a RuntimeError will be raised and a traceback will occur.

clear() → None. Remove all items from od.

freeze()
Freeze the dictionary to prevent further editing.

frozen
Whether or not the dictionary is frozen and can not be modified.

Return type bool

pop(*k*[, *d*]) → v, remove specified key and return the corresponding
value. If key is not found, d is returned if given, otherwise KeyError is raised.

popitem(**args*, ***kwargs*)
Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

thaw()
Thaw the dictionary to once again enable editing.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class PrefixLoggerAdapter(*prefix*, **args*, ***kwargs*)
Bases: logging.LoggerAdapter

A log adapter that simply prefixes the specified string to all messages. A single space will be inserted between the prefix and the message.

__init__(*prefix*, **args*, ***kwargs*)

Parameters **prefix** (*str*) – The string to prefix all messages with.

process(*message*, *kwargs*)
Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you'll only need to override this one method in a LoggerAdapter subclass for your specific needs.

class Mock(**args*, ***kwargs*)
Bases: object

A fake object used to replace missing imports when generating documentation.

class Thread(*target=None*, *name=None*, *args=()*, *kwargs=None*, ***_kwargs*)
King Phisher's base threading class with two way events.

1.25 version

This module collects all import version information for the application. This is the authoritative source for the applications version information and should be used anywhere the version is required.

Note: This is a *"Clean Room" module* and is suitable for use during initialization.

1.25.1 Data

distutils_version = '1.16.0b0'
A string suitable for being parsed by distutils.version classes.

revision = '51185b4a2afd7e70bc8436c50550258cf0465f6e'
The git revision identifying the latest commit if available.

rpc_api_version = rpc_api_version(major=6, minor=5)
A tuple representing the local version of the RPC API for use with compatibility checks. The major version is incremented when backwards incompatible changes are made and the minor version is incremented when backwards compatible changes are made.

version = '1.16.0-beta (rev: 51185b4a2afd)'
A string representing the full version information.

version_info = version_info(major=1, minor=16, micro=0)
A tuple representing the version information in the format ('major', 'minor', 'micro')

version_label = 'beta'
A version label such as alpha or beta.

1.25.2 Functions

get_revision(*encoding='utf-8'*)
Retrieve the current git revision identifier. If the git binary can not be found or the repository information is unavailable, None will be returned.

Parameters **encoding** (*str*) – The encoding to use for strings.

Returns The git revision tag if it's available.

Return type str

1.26 xor

This module provides basic support for XOR encoding and decoding operations.

1.26.1 Functions

xor_decode(*data*, *encoding='utf-8'*)

Decode data using the XOR algorithm. This is not suitable for encryption purposes and should only be used for light obfuscation. This function requires the key to be set as the first byte of *data* as done in the *xor_encode()* function.

Parameters **data** (*str*) – The data to decode.

Returns The decoded data.

Return type str

xor_encode(*data*, *seed_key=None*, *encoding='utf-8'*)

Encode data using the XOR algorithm. This is not suitable for encryption purposes and should only be used for light obfuscation. The key is prepended to the data as the first byte which is required to be decoded py the *xor_decode()* function.

Parameters

- **data** (*bytes*) – The data to encode.
- **seed_key** (*int*) – The optional value to use as the for XOR key.

Returns The encoded data.

Return type bytes

CHAPTER 2

The King Phisher Client

2.1 Additional Configuration

The following configuration settings will be honored but can not be set from within the client's user interface. The client configuration file is usually located in the following locations depending on the host operating system:

Linux `~/.config/king-phisher/config.json`

Windows `%LOCALAPPDATA%\king-phisher\config.json`

Note: The King Phisher client will overwrite its configuration file when it exits to store the latest values. This means that the client should not be running when the configuration file is being manually edited so the changes are not overwritten.

Setting Name	Default Value
gui.refresh_frequency	`5m` (5 minutes)
gui.show_deaddrop	`false`
mailer.max_messages_per_connection	`5`
plugins.path	`[]` (No additional plugin paths)
rpc.serializer	`null` (Automatically determined)
ssh_preferred_key	`null` (Automatically determined)
text_font	`"monospace 10"`
text_source.hardtabs	`false`
text_source.highlight_line	`true`
text_source.tab_width	`2`
text_source.theme	`"cobalt"` (One of the GtkSourceView StyleSchemes)
text_source.wrap_mode	`"NONE"` (One of `"CHAR"`, `"NONE"`, `"WORD"`, `"WORD_CHAR"`) [1]

[1] See GtkWrapMode for more details.

2.2 Completion Data

Some classes provided by the `widget.completion_providers` module require large amounts of data to function. This data is stored encoded in JSON to be loaded when these classes are initialized. The formats of the data are specific to each completion provider depending on the needs of their target syntax.

2.2.1 HTML

The HTML data file is a dictionary whose keys are HTML 5 tags such as body, input and script. Each of these keys values is either None if the tag does not have any attributes or a list of the valid attribute names. Each of the defined attributes are assumed to require a value, however ones which do not are suffixed with `!`. This suffix is used by the completion provider to determine if the opening definition for an attribute (`="`) should be appended to the token or not.

Example data containing completion information for the html and input tags:

```
{
  "html": null,
  "input": [
    "disabled!",
    "type"
   ]
}
```

2.2.2 Jinja

The Jinja data file is a dictionary containing two sub keys of `global` and `context` for global, and context specific data respectively. The global key's value is a dictionary containing three subkeys of `filters`, `tests` and `tokens` for the different kinds of Jinja terms which should be auto completed. The filters and tests keys have values of lists including all of the defined Jinja filters and tests respectively.

The tokens key has a value of a dictionary which contains the tokens broken out into a hierarchy of objects and attributes. Attributes which have sub-attributes are represented as dictionaries while attributes which have no attributes and are thus leaves have values of None. In the context of completion, variables and functions are treated as tokens because neither one are dependant on presence of a setup statement which is the case with filters and tests.

Tokens, filters and tests which are callable and require at least one argument to be specified are all suffixed with `(`. This suffix is used by the completion provider to signify that arguments are expected.

The top-level context key contains subkeys that define additional data to be merged with the global filters, tests and tokens based on a defined context. This allows the global Jinja environment data to be added to context specific providers.

Example data containing global filters, tests and tokens along with a truncated "email" context.

```
{
  "context": {
    "email": {
      "tokens": {
        ...
      }
    }
  },
  "global": {
    "filters": [
```

(continues on next page)

(continued from previous page)

```
      "replace(",
      "title"
    ],
    "tests": [
      "defined",
      "equalto("
    ],
    "tokens": {
      "range(": null,
      "time": {
        "local": null,
        "utc": null
      },
      "version": null
    }
  }
}
```

2.3 GObject Signals

These signals can be used by the client API and plugins to subscribe to specific events. To explicitly connect after the default handler for a signal, use the *connect_after* method instead of *connect*. Some signals require a value to be returned by their handlers as noted.

2.3.1 Signal Flags

The "Signal flags" attribute of each of the signals describes certain attributes of their respective signals. See the GObject Signals documentation for more information including a detailed description of the signal emission process.

SIGNAL_ACTION Signals with the `SIGNAL_ACTION` flag are safe to be emitted from arbitrary components within the application. These signals have default handlers which perform their action and are not stateful.

SIGNAL_RUN_FIRST Signals with the `SIGNAL_RUN_FIRST` flag execute the default handler before handlers which are connected with either the `connect()` or `connect_after()` methods. These signals therefor do not provide connected handlers with an opportunity to block the emission of the signal from the default handler.

SIGNAL_RUN_LAST Signals with the `SIGNAL_RUN_LAST` flag execute the default function after handlers which are connected with the `connect()` method but before handlers which are connected with the `connect_after()` method. This provides connected handlers with an opportunity to block the default function by halting emissionof the signal by using the `emit_stop_by_name()` method.

Note: Plugins which connect to signals should use the *`signal_connect()`* method which by defaults uses `connect()` to connect the signal. Alternatively `connect_after()` can be used by setting the **after** keyword argument to `True`.

2.3.2 Application Signals

The following are the signals for the *KingPhisherClientApplication* object.

campaign-changed(campaign_id)
This signal is emitted when campaign attributes are changed. Subscribers to this signal can use it to update and refresh information for the modified campaign.

Signal flags SIGNAL_RUN_FIRST

Parameters **campaign_id** (*str*) – The ID of the campaign whose information was changed.

campaign-created(campaign_id)
This signal is emitted after the user creates a new campaign id. Subscribers to this signal can use it to conduct an action after a new campaign id is created.

Signal flags SIGNAL_RUN_FIRST

Parameters **campaign_id** (*str*) – The ID of the new campaign.

campaign-delete(campaign_id)
This signal is emitted when the user deletes a campaign. Subscribers to this signal can use it to conduct an action after the campaign is deleted.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **campaign_id** (*str*) – The ID of the campaign.

campaign-set(old_campaign_id, new_campaign_id)
This signal is emitted when the user sets the current campaign. Subscribers to this signal can use it to update and refresh information for the current campaign. The `config` "campaign_id" and "campaign_name" keys have already been updated with the new values when this signal is emitted.

Signal flags SIGNAL_RUN_FIRST

Parameters

- **old_campaign_id** (*str*) – The ID of the old campaign or None if the client is selecting one for the first time.
- **new_campaign_id** (*str*) – The ID of the new campaign.

config-load(load_defaults)
This signal is emitted when the client configuration is loaded from disk. This loads all of the clients settings used within the GUI.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **load_defaults** (*bool*) – Load missing options from the template configuration file.

config-save()
This signal is emitted when the client configuration is written to disk. This saves all of the settings used within the GUI so they can be restored at a later point in time.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

credential-delete(row_ids)
This signal is emitted when the user deletes a credential entry. Subscribers to this signal can use it to conduct an action an entry is deleted.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **row_ids** (*[int, ..]*) – The row IDs that are to be deleted.

exit()
This signal is emitted when the client is exiting. Subscribers can use it as a chance to clean up and save any remaining data. It is emitted before the client is disconnected from the server. At this point the exit operation can not be cancelled.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

exit-confirm()
This signal is emitted when the client has requested that the application exit. Subscribers to this signal can use it as a chance to display a warning dialog and cancel the operation.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

message-delete(row_ids)
This signal is emitted when the user deletes a message entry. Subscribers to this signal can use it to conduct an action an entry is deleted.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **row_ids** (*[str, ..]*) – The row IDs that are to be deleted.

message-sent(target_uid, target_email)
This signal is emitted when the user sends a message. Subscribers to this signal can use it to conduct an action after the message is sent, and the information saved to the database.

Signal flags SIGNAL_RUN_FIRST

Parameters

- **target_uid** (*str*) – Message uid that was sent.
- **target_email** (*str*) – Email address associated with the sent message.

reload-css-style()
This signal is emitted to reload the style resources of the King Phisher client.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

rpc-cache-clear()
This signal is emitted to clear the RPC objects cached information. Subsequent invocations of RPC cache enabled methods will return fresh information from the server.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

server-connected()
This signal is emitted when the client has connected to the King Phisher server. The default handler sets the initial campaign optionally prompting the user to select one if one has not already been selected.

Signal flags SIGNAL_RUN_FIRST

server-disconnected()
This signal is emitted when the client has disconnected from the King Phisher server.

Signal flags SIGNAL_RUN_FIRST

sftp-client-start()
This signal is emitted when the client starts sftp client from within King Phisher. Subscribers can conduct an action prior to the default option being ran from the client configuration.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

visit-delete(row_ids)
This signal is emitted when the user deletes a visit entry. Subscribers to this signal can use it to conduct an action an entry is deleted.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **row_ids** (*[str, ..]*) – The row IDs that are to be deleted.

unhandled-exception(exc_info, error_uid)
This signal is emitted when the application encounters an unhandled Python exception.

Signal flags SIGNAL_RUN_FIRST

Parameters

- **exc_info** (*tuple*) – A tuple of three objects corresponding to the return value of the sys.exc_info() function representing the exception that was raised.
- **error_uid** (uuid.UUID) – The unique identifier that has been assigned to this exception for tracking.

2.3.3 Mail Tab Signals

The following are the signals for the *MailSenderTab* object.

message-create(target, message)
This signal is emitted when the message and target have been loaded and constructed. Subscribers to this signal may use it as an opportunity to modify the message object prior to it being sent.

New in version 1.10.0b3.

Signal flags SIGNAL_RUN_FIRST

Parameters

- **target** (*MessageTarget*) – The target for the message.
- **message** (*TopMIMEMultipart*) – The message about to be sent to the target.

message-data-export(target_file)
This signal is emitted when the client is going to export the message configuration to a King Phisher Message (KPM) archive file.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters **target_file** (*str*) – The path to write the archive file to.

Returns Whether or not the message archive was successfully imported.

Return type bool

message-data-import(target_file, dest_dir)
This signal is emitted when the client is going to import the message configuration from a King Phisher Message (KPM) archive file.

Signal flags SIGNAL_ACTION | SIGNAL_RUN_LAST

Parameters

- **target_file** (*str*) – The source archive file to import.
- **dest_dir** (*str*) – The destination directory to unpack the archive into.

Returns Whether or not the message archive was successfully imported.

Return type bool

message-send(target, message)
This signal is emitted after the message has been fully constructed (after *message-create*) and can be used as an opportunity to inspect the message object and prevent it from being sent.

New in version 1.10.0b3.

Signal flags SIGNAL_RUN_LAST

Parameters

- **target** (*MessageTarget*) – The target for the message.
- **message** (*TopMIMEMultipart*) – The message about to be sent to the target.

Returns Whether or not to proceed with sending the message.

Return type bool

send-finished()
This signal is emitted after all messages have been sent.

Signal flags SIGNAL_RUN_FIRST

send-precheck()
This signal is emitted when the user is about to start sending phishing messages. It is used to ensure that all settings are sufficient before proceeding. A handler can return False to indicate that a pre-check condition has failed and the operation should be aborted.

Signal flags SIGNAL_RUN_LAST

Returns Whether or not the handler's pre-check condition has passed.

Return type bool

target-create(target)
This signal is emitted when the target has been loaded and constructed. Subscribers to this signal may use it as an opportunity to modify the target object prior to it being sent.

New in version 1.10.0b3.

Signal flags SIGNAL_RUN_FIRST

Parameters **target** (*MessageTarget*) – The target for the message.

target-send(target)
This signal is emitted after the target has been fully constructed (after *target-create*) and can be used as an opportunity to inspect the target object and prevent it from being sent to.

New in version 1.10.0b3.

Signal flags SIGNAL_RUN_LAST

Parameters **target** (*MessageTarget*) – The target for the message.

Returns Whether or not to proceed with sending to the target.

Return type bool

2.3.4 Server Event Signals

The following are the signals for the *ServerEventSubscriber* object. These events are published by the server forwarded to the client based on the active subscriptions. When an event is forwarded to a client the corresponding GObject signal is emitted for consumption by the client. See the section on *Published Events* for more details.

db-alert-subscriptions(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-campaigns(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-campaign-types(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-companies(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-company-departments(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-credentials(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-deaddrop-connections(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-deaddrop-deployments(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-industries(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-landing-pages(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-messages(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-users(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.
- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

db-visits(event_type, objects)

Signal flags SIGNAL_RUN_FIRST

Parameters

- **event_type** (*str*) – The type of event, one of either deleted, inserted or updated.

- **objects** (*list*) – The objects from the server. The available attributes depend on the subscription.

2.4 Keyboard Shortcuts

The following keyboard shortcuts are available for use within the client GUI.

Key Combination	Action Description
Ctrl + O	Open a campaign
Ctrl + Q	Exit the client
Ctrl + F1	Open an RPC terminal
Ctrl + F2	Open the SFTP client
Ctrl + Shift + F1	Clear the RPC cache
Ctrl + Shift + F2	Write the configuration to disk
Ctrl + Shift + F12	Reload the style css file

CHAPTER 3

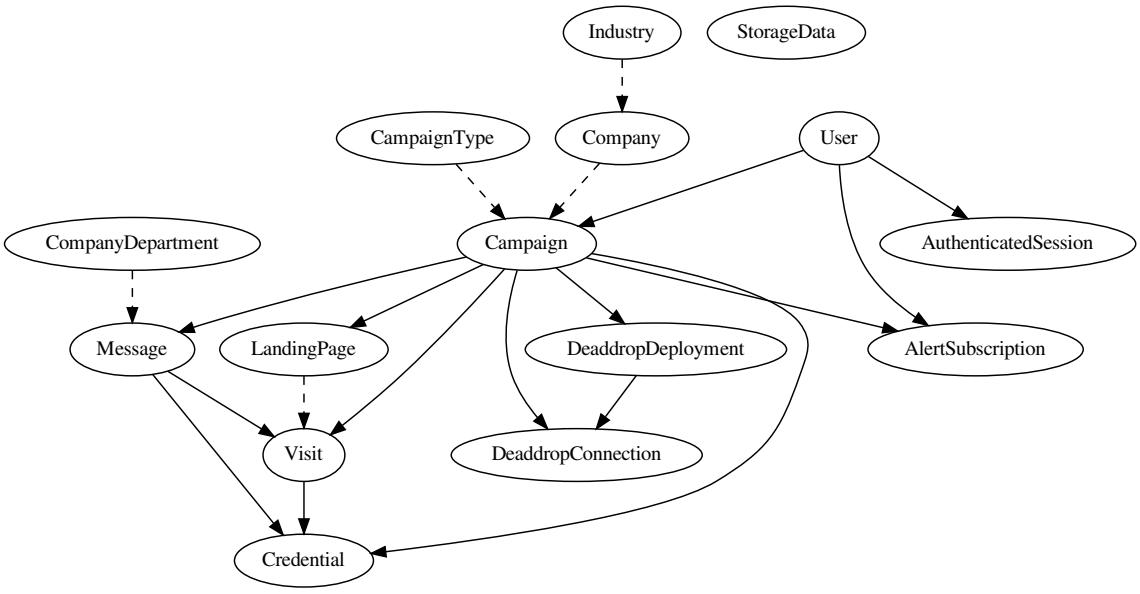
The King Phisher Server

3.1 Database

3.1.1 Database Overview

Table Relationships

The following diagram outlines the relationships of the various tables in the database. Nodes are connected by foreign key constraints. The arrow head references the object which has the constraint.



Schema Versioning

The King Phisher database uses an internal version number defined as *SCHEMA_VERSION* which is used by the initialization code to determine whether or not the stored database schema (the one existing in the database) matches the running schema (the one defined in the source code). When the schemas are not the same, the database is considered to be incompatible. The King Phisher server process will then automatically attempt to upgrade the stored database schema.

If the stored database schema is newer than the running schema, the King Phisher process can not downgrade it. This would happen for example if a developer were to use version control to revert the project code to an older version. In this case the older version would have no knowledge of the newer schema and would therefor be unable to "downgrade" it to a compatible version. In this case the developer must use the included database schema migration utilities to update the stored database schema to a compatible version before switching to the older project revision.

Alembic

King Phisher uses Alembic to manage its database schema versions. This can be used to explicitly upgrade and downgrade the schema version from the command line. The Alembic environment files are stored with the server data files at `data/server/king_phisher/alembic`.

The King Phisher version of the Alembic `env` file is modified to support two ways for the database connection string to be passed from the command line. This removes the need to store the credentials int the `alembic.ini` file. The two supported options are "config" and "database". Both are supplied as settings to the `-x` option in the form `-x SETTING=VALUE` with no spaces between the settings and their values.

config The `config=` option takes a path to the King Phisher server configuration file where the database connection string will be used.

database The `database=` option takes an explicit database connection string on the command line. The syntax is the same as how it would be stored in the server configuration file.

Example running Alembic's `current` subcommand with the database connection string taken from the server's configuration file.

```
# run from data/server/king_phisher
alembic -x config=../../../server_config.yml current
```

Schema Version Identifiers

Alembic and King Phisher must keep separate version identifiers. This is because Alembic uses revision strings in it's internal, linked format while King Phisher uses simple numeric versioning to easily identify newer schemas. When creating a new Alembic migration file, it's important to set the King Phisher schema version as well which must be explicitly done by the developer. The King Phisher stored database schema version exists in the `storage_data` in the `metadata` namespace with the key `schema_version`. See *set_metadata()* for a convenient way to set this value. The Alembic revision identifier is stored as a single record in the `alembic_version` table under the `version_num` column.

Key-Value Storage

The database provides serialized key-value storage to allow semi-arbitrary objects to be stored in the database. This is more convenient than dealing with and managing individual files for the following reasons:

- Server-written data is kept together in a single location (the database)
- The developers do not need to worry about file formats and permissions

The primary interface into this storage is provided by the *storage* module, specifically the *KeyValueStorage* class. Each instance should specify the *namespace* parameter to uniquely identify it's usage.

Key-Value Namespaces

The following namespaces are currently in use by the key-value storage system.

- `metadata` – Storage of metadata values related to the server instance.
- `plugins.$name` – Storage of server plugin specific data. See *king_phisher.server.plugins.ServerPlugin.storage*. `$name` is the name of the plugin using the storage.
- `server.ssl.sni.hostnames` – Storage of SSL-SNI certificate configurations for specific hostnames. Used to permit SNI configuration changes at run time.

3.1.2 Database Schema

This schema defines the various database tables and fields for the objects managed by the King Phisher server. These are exposed over the *GraphQL* interface with the exception of fields which are restricted based on permissions.

Tables

alert_subscriptions
Subscriptions to alerts for campaigns that users are interested in receiving notifications for.

expiration
The expiration for which the user can set to no longer receive notifications.

Nullable True

Type DateTime

id

Primary Key True

Type Integer

user_id
The identifier of the user which created the alert subscription.

Nullable False

Foreignkey *users.id*

campaign_id
The identifier of the campaign the user is interested in receiving notifications for.

Nullable False

Foreignkey *campaigns.id*

authenticated_sessions
An authenticated session associated with a user that has logged into the server over RPC.

id

Primary Key True

Type String

created
The time at which the session was created.

Nullable False

Type DateTime

last_seen
The time at which the last authenticated request associated with this session was seen. Used to support session timeouts.

Nullable False

Type DateTime

user_id
The identifier of the authenticated user who established this session.

Nullable False

Foreignkey *users.id*

campaign_types
The type information for a particular campaign. This information is useful for determining the success metrics. For example, a campaign type can be set as "Credentials" for a campaign intending to collect credentials from users while a campaign which does not can have the type set to "Visits". This will ensure that the campaign of type "Visits" is not considered to be less successful due to it having not collected any credentials.

id

Primary Key True

Type Integer

name
A short name for the campaign type, e.g. "Credentials".

Nullable False

Type String

description
A description of the campaign type, e.g. "Campaigns that intend to collect credentials from target users".

Nullable True

Type String

campaigns
A logical testing unit representing a single campaign.

expiration
The time at which the server should cease collection of testings information.

Nullable True

Type DateTime

id

Primary Key True

Type Integer

name
A short, human-readable name for the campaign.

Nullable False

Type String

description
A field to store any descriptive information regarding the campaign such as why or how it was conducted.

Nullable True

Type String

user_id
The identifier of the user who originally created the campaign.

Nullable False

Foreignkey *users.id*

created
The time at which the campaign was created.

Nullable True

Type DateTime

max_credentials
The maximum number of credentials to collect *per user*. This setting can be used to alter how the server behaves when a target submits multiple credentials during the course of a campaign.

Nullable True

Type Integer

campaign_type_id
The identifier for the campaign's type.

Nullable True

Foreignkey *campaign_types.id*

company_id
The identifier for the company for which this campaign performs testing.

Nullable True

Foreignkey *companies.id*

credential_regex_username
A regular expression that can be used to determine the validity of a credential's username field.

Nullable True

Type String

credential_regex_password
A regular expression that can be used to determine the validity of a credential's password field.

Nullable True

Type String

credential_regex_mfa_token
A regular expression that can be used to determine the validity of a credential's mfa token field.

Nullable True

Type String

companies
An entity for which a campaign's test is conducted for.

id

Primary Key True

Type Integer

name
A short, human-readable name for the entity.

Nullable False

Type String

description
A field to store any descriptive information regarding the entity.

Nullable True

Type String

industry_id
The identifier of the primary industry in which the entity operates.

Nullable True

Foreignkey *industries.id*

url_main
The URL to the entity's main web site, useful for incorporation into site templates.

Nullable True

Type String

url_email
The URL to the entity's email portal, useful for incorporation into site templates.

Nullable True

Type String

url_remote_access
The URL for the entity's remote access solution, useful for incorporation into site templates.

Nullable True

Type String

company_departments
A subdivision of a company used to group targets with similar roles together.

id

Primary Key True

Type Integer

name
A short, human-readable name for the subdivision.

Nullable False

Type String

description
A field to store any descriptive information regarding the subdivision.

Nullable True

Type String

credentials
A table storing authentication information collected from a target during the course of a campaign.

id

Primary Key True

Type Integer

visit_id
The identifier of the visit which submitted the credential information.

Nullable False

Foreignkey *visits.id*

message_id
The identifier of the message which submitted the credential information.

Nullable False

Foreignkey *messages.id*

campaign_id
The identifier campaign the information was collected as a part of.

Nullable False

Foreignkey *campaigns.id*

username
The username submitted by the target.

Nullable True

Type String

password
The password submitted by the target.

Nullable True

Type String

mfa_token
The multi-factor authentication (MFA) token submitted by the target. This may, for example be a Time-Based One-Time Password (TOTP) code.

Nullable True

Type String

submitted
The time at which the credential information was submitted.

Nullable True

Type DateTime

regex_validated
Whether or not the fields passed validation with the regular expressions defined by the campaign at the time the credentials information was submitted. If no validation took place because no regular expressions were defined by the campaign, this field is null. If a regular expression for validation was defined for a field that was not submitted, validation fails and this field is false. See *validate_credential()* for more information.

Nullable True

Type Boolean

deaddrop_connections
A connection instance of an agent which has sent information to the server to prove that the agent was executed.

id

Primary Key True

Type Integer

deployment_id
The deployment identifier of agent which initiated the connection.

Nullable False

Foreignkey *deaddrop_deployments.id*

campaign_id
The identifier campaign the information was collected as a part of.

Nullable False

Foreignkey *campaigns.id*

count
The number of times the agent made the connection with the same information, implying that the agent was executed multiple times.

Nullable True

Type Integer

ip
The external IP address from which this information was submitted and collected from.

Nullable True

Type String

local_username
The username that executed the agent.

Nullable True

Type String

local_hostname
The hostname the agent was executed on.

Nullable True

Type String

local_ip_addresses
The local IP addresses the agent identified on the system from which it was executed.

Nullable True

Type String

first_seen
The first time the information was submitted to the server.

Nullable True

Type DateTime

last_seen
The last time the information was submitted to the server.

Nullable True

Type DateTime

deaddrop_deployments
An instance of a generated agent which can be distributed as part of testing to identify users that are susceptible to executing arbitrary programs.

id

Primary Key True

Type String

campaign_id
The identifier of the campaign the deaddrop agent was generated for.

Nullable False

Foreignkey *campaigns.id*

destination
A descriptive field describing where the agent was deployed to. Used for reporting and tracking purposes.

Nullable True

Type String

industries
An industry in which a company operates in.

id

Primary Key True

Type Integer

name
A short, human-readable name for the industry.

Nullable False

Type String

description
A field to store any descriptive information regarding the industry.

Nullable True

Type String

landing_pages
A page that is intended to be visited during the course of a test to be qualified as a failure. Visits to the landing page will increment the *visits.count* field, while requests to non-landing pages will not. A campaign may

have one or more landing pages, and they are automatically identified from the Target URL when messages are sent.

id

Primary Key True

Type Integer

campaign_id
The identifier of the campaign this landing page is associated with.

Nullable False

Foreignkey *campaigns.id*

hostname
The hostname component of the URL this landing page uses.

Nullable False

Type String

page
The path component of the URL this landing page uses.

Nullable False

Type String

messages
A message that was sent to a target user to test their susceptibility to phishing attempts.

id

Primary Key True

Type String

campaign_id
The identifier of the campaign which this message was sent as a part of.

Nullable False

Foreignkey *campaigns.id*

target_email
The email address of the user who this message was sent to.

Nullable True

Type String

first_name
The first name of the user who this message was sent to.

Nullable True

Type String

last_name
The last name of the user who this message was sent to.

Nullable True

Type String

opened
The time at which the message was confirmed to have been opened. This field is prone to false negatives due to many email clients not automatically loading remote images.

Nullable True

Type DateTime

opener_ip
The IP address which opened the message.

Nullable True

Type String

opener_user_agent
The user agent of the request sent when the message was opened.

Nullable True

Type String

sent
The time at which the message was sent to the target.

Nullable True

Type DateTime

reported
The time at which the message was reported by the target.

Nullable True

Type DateTime

trained
Whether or not the taget agreed to any training provided during the course of the testing.

Nullable True

Type Boolean

delivery_status
A short, human-readable status regarding the state of delivery of the message such as delivered, rejected or deferred.

Nullable True

Type String

delivery_details
Any additional details regarding the state of the message delivery status.

Nullable True

Type String

testing
Whether or not the message was intended for testing and should be omitted from the overall results.

Nullable False

Type Boolean

company_department_id
The identifier of the company subdivision that the target is a member of.

Nullable True

Foreignkey *company_departments.id*

storage_data
Storage for internal server data that is generated at run time.

id

Primary Key True

Type Integer

created
The time at which the data unit was created.

Nullable True

Type DateTime

modified
The time at which the data unit was modified.

Nullable True

Type DateTime

namespace
The namespace in which the data unit exists to allow the same *storage_data.key* to be used multiple times while remaining uniquely identifiable.

Nullable True

Type String

key
The key by which the data unit is retrieved. This value must be unique within the defined *storage_data.namespace*.

Nullable False

Type String

value
The readable and writable data unit itself, serialized as a binary object to be loaded and unloaded from the database.

Nullable True

Type Binary

users
An authorized user as loaded through the server's authentication mechanism.

expiration
The time at which the user should no longer be able to authenticate to the server.

Nullable True

Type DateTime

id

Primary Key True

Type Integer

name
The name of the user.

Nullable False

Type String

description
A field to store any descriptive information regarding the user.

Nullable True

Type String

phone_carrier
The service provider of the user's cell phone. This information is used to send text messages via the providers email to SMS gateway.

Nullable True

Type String

phone_number
The user's cell phone number. This information is used to provide the user with alerts regarding campaigns to which they have subscribed.

Nullable True

Type String

email_address
The user's email address. This information is used to provide the user with alerts regarding campaigns to which they have been subscribed.

Nullable True

Type String

otp_secret
A secret value used when prompting for Multi Factor Authentication (MFA) to the server.

Nullable True

Type String

last_login
The time at which the user last authenticated.

Nullable True

Type DateTime

access_level
The level of access available to a users, where a higher number represents less access than a lower number.

Nullable False

Type Integer

visits
An instance where a targeted user has failed their testing attempt by visiting the link provided to them from a message.

id

Primary Key True

Type String

message_id
The identifier of the message that was sent to the target which initiated the visit.

Nullable False

Foreignkey *messages.id*

campaign_id
The identifier of the campaign that this visit is associated with.

Nullable False

Foreignkey *campaigns.id*

count
The number of times the user visited a landing page associated with the campaign. This would be the case when the user visits the link they were provided multiple times from the same browser.

Nullable True

Type Integer

ip
The IP address from which the user visited the server.

Nullable True

Type String

details
Any applicable details regarding the visist.

Nullable True

Type String

user_agent
The user agent of the visist request.

Nullable True

Type String

first_landing_page_id
The identifier of the first landing page the visit was made. This is used to determine which landing page a user visited if multiple landing pages are associated with the campaign.

Nullable True

Foreignkey *landing_pages.id*

first_seen
The time at which the first visit was made to the server.

Nullable True

Type DateTime

last_seen
The time at which the last visit was made to the server.

Nullable True

Type DateTime

3.2 GraphQL

3.2.1 GraphQL Overview

The RPC API provides a function for executing GraphQL queries against the server. The schema the server supports allows accessing the database models through the `db` type as well as some additional information such as the server plugins.

Note: For consistencies within the GraphQL API and with GraphQL best practices, it is important to note that names are `camelCase` and not `snake_case`.

Interface Extensions

The GraphQL schema supported by King Phisher implements the Relay connection interface allowing easier pagination using a cursor. As an extension to this interface, the King Phisher schema also includes a `total` attribute to the connection object. This attribute allows a query to access the number of nodes available for a specific connection.

Schema

The following table represents the top-level objects available in the GraphQL schema and their various sub-object types as applicable. For more information, see the *GraphQL Schema* documentation.

Object Name	Object Type	Description
`db`	Object	Database models. See *Table Relationships* for information on available sub-objects.
`geoloc`	*`GeoLocation`*	Geolocation information.
`hostnames`	[String]	The hostnames that are configured for use with this server.
`plugin`	*`Plugin`*	Specific information for a loaded plugin.
`plugins`	Connection	Information on all loaded plugins.
`siteTemplate`	*`SiteTemplate`*	Information for an available site template.
`siteTemplates`	Connection	Information on all available site templates.
`ssl`	*`SSL`*	Information regarding the SSL configuration and status.
`version`	String	The *`version`* of the King Phisher server.

Connection A connection sub-object is a special object providing a defined interface used to refer to an array of objects. The connection sub-object has a `total` attribute which is an integer as well as an `edges` attribute. See Connection Types for more information.

Object Objects can in turn have their own attributes which can be a combination of additional sub-objects or scalars.

Additional Database Model Attributes

Database objects which have an IP address string attribute associated with their model have an additional attribute containing the corresponding geo location information. This geo location attribute uses the same naming prefix, for example the geo location information for a `ip` attribute can be accessed from the `ipGeoloc` attribute.

Additional Database Connection Arguments

Database connections can include additional arguments which allow manipulation of the queried data.

The filter Argument

The `filter` argument is a `FilterInput` GraphQL object and can be passed to database connection to filter what data is returned by the query. This argument is an object containing one or more of the following key words.

Keyword	Type	Default	Description
and [1]	List	N/A	A list of additional filter objects, where all must evaluate to true.
or [1]	List	N/A	A list of additional filter objects, where one or more must evaluate to true.
field [1]	String	N/A	The name of a database field to filter by.
operator	FilterOperatorEnum	`EQ`	The operator to use with value, one of `EQ`, `GE`, `GT`, `LE`, `LT`, or `NE`.
value	AnyScalar	`Null` [2]	The value of the field to use with the specified comparison operator.

[1] Exactly one of these keywords must be specified.

[2] `null` can not be passed as a literal for input. To compare a value to `null`, the `value` keyword must be omitted.

The sort Argument

The `sort` argument is a list of `SortInput` GraphQL objects (described below) which can be passed to a database connection to sort the query data by one or more fields.

Keyword	Type	Default	Description
field*	String	N/A	The name of a database field to sort by.
direction	SortDirectionEnum	`AESC`	The direction in which to sort the data, either `AESC` or `DESC`.

* This keyword must be specified.

Executing Raw Queries

Raw GraphQL queries can be executed using the `tools/database_console.py` utility. This console provides a `graphql_query` function which takes a query string parameter and optional query variables. This can be used for easily testing queries. It should be noted however that using this utility directly on the server does not restrict access to data as the RPC interface does.

The client's RPC terminal (only available on Linux due to the dependency on VTE) can also be used to easily execute raw GraphQL queries. The RPC method can be called directly, or when IPython is available, either the `%graphql` or `%graphql_file` commands can be used. The former of which takes a GraphQL query as an argument, while the second takes the path to a file on disk to execute. Both of these are useful for debugging and inspecting GraphQL queries and their resulting data structures.

3.2.2 GraphQL Schema

Top-Level Fields

These are the top-level fields that are accessible from within the default query.

geoloc
ip

Parameters **geoloc** (*String!*) – The IP address to lookup the Geo Location for.

Type *GeoLocation*

Lookup the *GeoLocation* for a specific IP address.

hostnames

Type [String]

A list of strings, one for each hostname that is configured for use on the server.

plugin
name

Parameters **name** (*String!*) – The name of the plugin to retrieve the information for.

Type *Plugin*

Lookup a specific *Plugin* by name.

plugins

Type Connection to *Plugin*

A connection for enumerating available server plugins.

siteTemplate
hostname,, *path*

Parameters

- **hostname** (*String*) – The hostname associated with the template. If the VHOSTs setting is enabled on the server, this option is required.
- **path** (*String!*) – The path of the template to retrieve information for.

Type *SiteTemplate*

Lookup a specific *SiteTemplate* by path and hostname combination.

siteTemplates
hostname,, *max_depth*

Parameters

- **hostname** (*String*) – An optional hostname to use for filtering returned site templates.
- **max_depth** (*Int*) – An optional maximum depth to search for site templates within the web root.

Type Connection to *SiteTemplate*

A connection for enumerating available site templates.

ssl

Type *SSL*

Lookup the server's *SSL* information.

version

Type String

The *version* of the King Phisher server.

Objects

GeoLocation

Location information as retrieved for an arbitrary IP address.

city

Type String

The city in which the location resides.

continent

Type String

The continent in which the location resides.

coordinates

Type [Float]

The coordinates of the location as an array of floating point numbers containing the latitude and longitude.

country

Type String

The country in which the location resides.

postalCode

Type String

The postal code in which the location resides.

timeZone

Type String

The time zone in which the location resides.

Plugin

Information regarding a server plugin.

authors

Type [String]

A list containing each of the author names.

classifiers

Type [String]

A list of string classifiers for describing qualities.

description

Type String

A text description of the plugin including what it does and any other information that may be necessary for users to know.

homepage

Type String

A URL for the homepage where the plugin originated.

name

Type String

The name of the plugin. As opposed to *title*, this value is an internal identifier derived from the plugin's file name and should not change.

reference

Type [String]

An optional list of URLs to use as references.

title

Type String

The plaintext title of the plugin to display in the UI. Unlike *name*, this value is intended for human consumption and may be updated.

version

Type String

The version of the template data.

SiteTemplate

Information for a site template which is available for use on the server. The template information can be used by the client to build a pretext and determine a landing page URL. As opposed to the *SiteTemplateMetadata* object, this structure contains information regarding where the template is installed versus what the template is.

created

Type DateTime

The timestamp of when this site template was created.

hostname

Type String

An optional hostname associated with this site template. This setting is only applicable when VHOSTs are enabled.

path

Type String

The path at which the site template is installed relative to the web root. This value must be used as the root for the pages defined in the metadata.

metadata

Type *SiteTemplateMetadata*

Metadata describing the site template.

SiteTemplateMetadata

Metadata for a specific site template describing what it is. As opposed to the *SiteTemplate* object, this structure contains information on what the template is versus where it is installed.

authors

Type [String]

A list containing each of the author names.

classifiers

Type [String]

A list of string classifiers for describing qualities.

description

Type String

A text description for the template, containing any notes for the user.

homepage

Type String

A URL for the homepage where the template originated.

pages

Type [String]

A list of relative paths suitable for use as landing pages

referenceUrls

Type [String]

A list of reference URL strings for the template.

title

Type String

The template's title.

version

Type String

The version of the template data.

SSL

Information regarding the use, configuration and capabilities of SSL on the server.

sniHostname

Parameters **hostname** (*String!*) – The hostname to retrieve the SNI configuration for.

Type *SniHostname*

A field for looking up the SNI configuration for a specific hostname.

sniHostnames

Type Connection to *SniHostname*

A connection for enumerating all of the available SNI configurations.

status

Type *SSLStatus*

An object describing the status of the server's SSL configuration.

SniHostname

An object describing the configuration of SSL's Server Name Indicator (SNI) extension for a specific hostname. If this object exists, the necessary data files are available however they may or may not be loaded as indicated by the *enabled* field.

enabled

Type Boolean

Whether or not the hostname is enabled.

hostname

Type String

The hostname for this configuration.

SSLStatus

An object describing the status of SSL as used by the server.

enabled

Type Boolean

Whether or not SSL is enabled for any interface the server is bound with.

hasLetsencrypt

Type Boolean

Whether or not the Let's Encrypt functionality is available. This requires that the `certbot` utility can be found.

hasSni

Type Boolean

Whether or not SSL's Server Name Indicator (SNI) extension is available in the Python implementation.

3.2.3 Example Queries

The following query is an example of retrieving the first 3 users from the users table. The query includes the necessary information to perform subsequent queries to iterate over all entries.

```
# GraphQL queries can have comments like this
query getFirstUser {
   # database objects are accessible under the 'db' type
   db {
      # retrieve the first 3 user objects
      users(first: 3) {
         # 'total' is an extension to the standard GraphQL relay interface
         total
         edges {
            # 'cursor' is a string used for iteration
            cursor
            node {
               # here the desired fields of the user object are specified
               id
               phoneNumber
```

(continues on next page)

(continued from previous page)

```
            }
         }
         # request information regarding the chunk of users returned
         pageInfo {
            endCursor
            hasNextPage
         }
      }
   }
}
```

This query returns a summary of all of the campaigns, including basic information such has when it was created, who by and the number of messages sent and visits received.

```
# Get a summary of all of the campaigns
query getCampaigns {
   db {
      campaigns {
         # get the total number of campaigns
         total
         edges {
            node {
               id
               created
               name
               # get the details about the user that created this campaign
               user {
                  id
                  phoneNumber
               }
               # get the total number of messages in this campaign
               messages {
                  total
               }
               # get the total number of visits in this campaign
               visits {
                  total
               }
            }
         }
      }
   }
}
```

This query demonstrates how whitespace is not necessary in GraphQL and the entire query can be on a single line.

```
# This query does not define the operation type or an operation name
# and is condensed to a single line
{ plugins { total edges { node { name title authors } } } }
```

Queries With Variables

The following two queries show how variables and arguments can be used in GraphQL.

```
# This query is an example of how a single database object can be referenced
# by its ID (which is always a string in GraphQL)
query getSpecificCampaign {
   db {
      # Campaign is specified here (instead of campaigns) as well as the ID
      campaign(id: "1") {
         name
         description
      }
   }
}
```

```
# This query is the same as the previous one, except here the campaign ID
# is defined as a variable
query getSpecificCampaign($id: String) {
   db {
      # The variable, defined above is then used here
      campaign(id: $id) {
         name
         description
      }
   }
}
```

Database Connections

This query uses the `filter` and `sort` arguments to process the queried data. See *Additional Database Connection Arguments* for more details.

```
query getFilteredCampaigns {
   db {
      campaigns(
         # define a filter for the campaigns
         filter: {
            # the following conditions must be met
            and: [
               # created on or after January 1st, 2017 (created GE "2017-01-01")
               {field: "created", operator: GE, value: "2017-01-01"},
               # and with either...
               {
                  or: [
                     # no expiration set (expiration EQ Null)
                     {field: "expiration"},
                     # or expiring before April 1st, 2018 (expiration LT "2018-04-01")
                     {field: "expiration", operator: LT, value: "2018-04-01"}
                  ]
               }
            ]
         },
         # sort the campaigns by the created timestamp
         sort: [{field: "created", direction: AESC}]
      ) {
         total
         edges {
            node {
```

(continues on next page)

(continued from previous page)

```
                id
                name
                # count the number of messages that were opened (opened NE Null)
                messages(filter: {field: "opened", operator: NE}) {
                   total
                }
             }
          }
       }
    }
}
```

Miscellaneous Queries

The following queries are for referencing the various APIs.

Site Templates

```
# get the available site templates
query getSiteTemplates {
   siteTemplates {
      total
      edges {
         node {
            # the top-level node contains information unique to this instance
            # such as the hostname its configured for and the path where it
            # is installed
            created
            hostname
            path
            # the metdata includes information regarding the template itself
            # such as who created it and what pages are intended to be used
            # as landing pages
            metadata {
               authors
               classifiers
               description
               pages
            }
         }
      }
   }
}
```

SSL SNI Hostnames

```
# get the available SSL SNI Hostnames
query getSslSniHostnames {
   ssl {
      sniHostnames {
         total
```

(continues on next page)

(continued from previous page)

```
        edges {
          node {
            # the node contains entries defining the hostname and whether
            # or not the certificate is loaded and enabled on the server
            enabled
            hostname
          }
        }
      }
    }
}
```

3.3 Published Events

3.3.1 Overview

Certain signals used by the server can be forwarded to clients via event subscriptions. In order to take advantage of this functionality the client opens a web socket to the server, and configures it's subscriptions using the available *Event API* functions. When a server signal is emitted the corresponding information is then forwarded to the subscribed clients over their open websocket.

3.3.2 Database Events

Database events can be subscribed to using the *event_id* of `db-TABLE_NAME`. Each of these events have the following sub-event types for each of the database operations.

- `deleted`
- `inserted`
- `updated`

These events are emitted by the respective `db_session_*` *Database Signals*. These signals are converted to events and organized by table (e.g. messages) instead of operation (e.g. inserted) because events are configured to send specific attributes. Not all attributes are available on all tables, however for one table the available attributes will always be available for all operations.

3.4 REST API

3.4.1 Overview

The King Phisher server provides an optional REST API *that is disabled by default*. It can be enabled by setting the server configuration value "rest_api.enabled" to true. An API token is required for all REST methods and must be present in the "token" parameter. If a static token is not specified in the server "rest_api.token" configuration, a new token will be randomly generated every time the server starts. The REST API methods are provided for access to convienence methods only. As such, campaign information can not be accessed via the REST API.

3.4.2 REST Methods

GET /_/api/geoip/lookup
Lookup an IP address in the GeoIP database.

Example request:

```
GET /_/api/geoip/lookup?token=SECRET_TOKEN&ip=4.2.2.2 HTTP/1.1
User-Agent: curl/7.40.0
Host: example.com
Accept: */*
```

Example response:

```
HTTP/1.0 200 OK
Server: Apache/2.4.12 (Unix)
Date: Thu, 04 Jun 2015 14:15:57 GMT
Content-Type: application/json
Content-Length: 204

{
  "result": {
   "city": null,
   "continent": "North America",
   "coordinates": [
     38.0,
     -97.0
   ],
   "country": "United States",
   "postal_code": null,
   "time_zone": null
  }
}
```

Query Parameters

- **token** – The server's REST API token.
- **ip** – The IP address to query geo location information for.

Status Codes

- 200 OK – The operation completed successfully.
- 401 Unauthorized – The REST API service is disabled or the token is invalid.
- 500 Internal Server Error – The operation encountered an exception.

GET /_/api/sms/send
Send an SMS message by emailing the carriers SMS gateway.

Example request:

```
GET /_/api/geoip/lookup?token=SECRET_TOKEN&message=hello+world!&phone_
↪number=1234567890&carrier=Sprint HTTP/1.1
User-Agent: curl/7.40.0
Host: example.com
Accept: */*
```

Example response:

```
HTTP/1.0 200 OK
Server: Apache/2.4.12 (Unix)
Date: Thu, 04 Jun 2015 14:30:40 GMT
Content-Type: application/json
Content-Length: 22

{
  "result": "sent"
}
```

Query Parameters

- **token** – The server's REST API token.
- **message** – The message to send.
- **phone_number** – The phone number to send the SMS to.
- **carrier** – The cellular carrier that the phone number belongs to.
- **from_address** – The optional address to display in the 'from' field of the SMS.

Status Codes

- 200 OK – The operation completed successfully.
- 401 Unauthorized – The REST API service is disabled or the token is invalid.
- 500 Internal Server Error – The operation encountered an exception.

3.5 RPC API

3.5.1 Overview

The RPC API is used by the King Phisher client to communicate with the server. It uses the RPC capabilities provided by the `AdvancedHTTPServer` module for the underlying communications. The RPC API provides a way for the client to retrieve and set information regarding campaigns as well as the server's configuration. RPC requests must be authenticated and are only permitted from the loopback interface. The client is responsible for using SSH to set up a port forward for requests. See the *Login Process* documentation for more information.

3.5.2 RPC API Versioning

It's important for the client and server components to have a compatible RPC version. The version each understands is described in the *rpc_api_version* object. This object contains both a major and minor version identifier. The major version is incremented when backwards-incompatible changes are made such as an argument or method is removed. The minor version is incremented when backwards-compatible changes are made such as when a new method is added or when a keyword argument is added whose default value maintains the original behavior.

In this way, it is possible for the server to support a newer RPC version than the client. This would be the case when the server is newer and provides more functionality than the older client requires. It is not possible for the client to support a newer RPC version than the server. This would imply that the client requires functionality that the server is unable to provide.

Since version v1.10.0, the GraphQL API loosens the interdependency between the RPC API version and the database's *schema version*. Since GraphQL allows the client to specify only the fields it requires, new fields can be added to the database without incrementing the major RPC API version. **It is still important to increment the minor RPC API**

version so the client knows that those fields are available to be requested through the *graphql* endpoint. If database fields are removed, columns are renamed, columns types are changed, or columns have additional restrictions placed on them (such as being nullable), the major RPC API version must be incremented.

The Table Fetch API

The RPC functions responsible for fetching table data through the `db/table/*` API endpoints (*db/table/get* and *db/table/view*) use a hard coded data set located in `data/server/king_phisher/table-api.json` to maintain backwards compatibility. This is required since the RPC client can not specify the columns and order of the columns that it is requesting as it can do with the *graphql* API endpoint. This data set effectively allows the table fetch RPC API endpoints to be artificially pinned to a specific database schema version. The other table API endpoints do not need to be pinned in such a fashion due to them taking the columns to work with as parameters. This means that an older but still compatible client (same major version but a lesser minor version as the server) would not be specifying columns which do not exist since renaming and removing columns require incrementing the major RPC API version.

3.5.3 General API

graphql
query,, *query_vars=None*

Handler *rpc_graphql()*

login

Handler *rpc_login()*

logout

Handler *rpc_logout()*

ping

Handler *rpc_ping()*

plugins/list

Handler *rpc_plugins_list()*

shutdown

Handler *rpc_shutdown()*

version

Handler *rpc_version()*

3.5.4 Campaign API

campaign/alerts/is_subscribed
campaign_id

Handler *rpc_campaign_alerts_is_subscribed()*

campaign/alerts/subscribe
campaign_id

Handler *rpc_campaign_alerts_subscribe()*

campaign/alerts/unsubscribe
campaign_id

Handler *rpc_campaign_alerts_unsubscribe()*

campaign/landing_page/new
campaign_id,, *hostname,*, *page*

Handler *rpc_campaign_landing_page_new()*

campaign/message/new
campaign_id,, *email_id,*, *email_target,*, *company_name,*, *first_name,*, *last_name*

Handler *rpc_campaign_message_new()*

campaign/new
name,, *description=None*

Handler *rpc_campaign_new()*

campaign/stats
campaign_id

Handler *rpc_campaign_stats()*

3.5.5 Configuration API

config/get
option_name

Handler *rpc_config_get()*

config/set
options

Handler *rpc_config_set()*

3.5.6 Event API

events/is_subscribed
event_id,, *event_type*

Handler *rpc_events_is_subscribed()*

events/subscribe
event_id,, *event_types,*, *attributes*

Handler *rpc_events_subscribe()*

events/unsubscribe
event_id,, *event_types,*, *attributes*

Handler *rpc_events_unsubscribe()*

3.5.7 GeoIP API

geoip/lookup
ip,, *lang=None*

Handler *rpc_geoip_lookup()*

geoip/lookup/multi
ips,, *lang=None*

Handler *rpc_geoip_lookup_multi()*

3.5.8 Hostnames API

hostnames/add
hostname

Handler *rpc_hostnames_add()*

New in version 1.13.0.

hostnames/get

Handler *rpc_hostnames_get()*

New in version 1.13.0.

3.5.9 SSL API

/ssl/letsencrypt/certbot_version

Handler *rpc_ssl_letsencrypt_certbot_version()*

/ssl/letsencrypt/issue
hostname,, *load=True*

Handler *rpc_ssl_letsencrypt_issue()*

/ssl/sni_hostnames/get

Handler *rpc_ssl_sni_hostnames_get()*

/ssl/sni_hostnames/load
hostname

Handler *rpc_ssl_sni_hostnames_load()*

/ssl/sni_hostnames/unload
hostname

Handler *rpc_ssl_sni_hostnames_unload()*

/ssl/status

Handler *rpc_ssl_status()*

3.5.10 Table API

db/table/count
table_name,, *query_filter=None*

Handler *rpc_database_count_rows()*

db/table/delete
table_name,, *row_id*

Handler *rpc_database_delete_row_by_id()*

db/table/delete/multi
table_name,, *row_ids*

Handler *rpc_database_delete_rows_by_id()*

db/table/get
table_name,, *row_id*

Handler *rpc_database_get_row_by_id()*

db/table/insert
table_name,, *keys,*, *values*

Handler *rpc_database_insert_row()*

db/table/set
table_name,, *row_id,*, *keys,*, *values*

Handler *rpc_database_set_row_value()*

db/table/view
table_name,, *page=0,*, *query_filter=None*

Handler *rpc_database_view_rows()*

3.6 Server Signals

3.6.1 Overview

Server signals are used by the server to dispatch events to subscribed handlers. This allows plugins to subscribe specific functions to be executed when a particular event occurs. These signals are defined in the `signals` module.

3.6.2 Signal Sender

The first parameter of each signal is the signal sender. It can be used by subscribers to only receive signal when emitted by a particular sender, effectively providing a filter. See the blinker documentation for more information.

3.6.3 Campaign Signals

campaign_alert
Emitted for each user who is subscribed to alerts for a particular campaign. Users subscribe to campaign alerts through the GUI by enabling the "Subscribe To Event Alerts" setting. Alerts are for either the "credentials" or "visits" table.

Note: This signal is not emitted for every entry into the respective tables but rather at progressively longer intervals to prevent the user from receiving an excessive amount of messages within a short period of time.

Parameters

- **table** (*str*) – The table name that the alert is for.
- **alert_subscription** (`king_phisher.server.database.models.AlertSubscription`) – The alert subscription.
- **count** (*int*) – The number associated with the alert event per the specified sender.

campaign_alert_expired
Emitted for each user who is subscribed to alerts for a particular campaign after it has expired.

Parameters

- **campaign** (king_phisher.server.database.models.Campaign) – The campaign which is expiring.
- **alert_subscription** (king_phisher.server.database.models.AlertSubscription) – The alert subscription.

campaign_expired
Emitted after a campaign has expired as determined by the expiration field. The server periodically checks for newly expired campaigns at an arbitrary interval. If a campaign is updated to expire at a time less than the next check minus the interval, then this signal will not be emitted for the campaign.

Parameters **campaign** (king_phisher.server.database.models.Campaign) – The campaign which is expiring.

3.6.4 Database Signals

db_initialized
Emitted after a connection has been made and the database has been fully initialized. At this point, it is safe to operate on the database.

Parameters **connection_url** (sqlalchemy.engine.url.URL) – The connection string for the database that has been initialized.

db_session_deleted
Emitted after one or more rows have been deleted on a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been deleted with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_session_inserted
Emitted after one or more rows have been inserted in a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been inserted with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_session_updated
Emitted after one or more rows have been updated in a SQLAlchemy session. At this point, references are valid but objects can not be modified. See sqlalchemy.orm.events.SessionEvents.after_flush() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target objects belong.
- **targets** (*tuple*) – The objects that have been updated with the session.
- **session** (sqlalchemy.orm.session.Session) – The SQLAlchemy session with which the *targets* are associated.

db_table_delete
Emitted before a row inheriting from Base is deleted from the database table. To only subscribe to delete events for a specific table, specify the table's name as the *sender* parameter when calling blinker.base.Signal.connect(). See sqlalchemy.orm.events.MapperEvents.before_delete() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (sqlalchemy.orm.mapper.Mapper) – The Mapper object which is the target of the event.
- **connection** (sqlalchemy.engine.Connection) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.
- **target** – The target object instance.

db_table_insert
Emitted before a row inheriting from Base is inserted into the database table. To only subscribe to insert events for a specific table, specify the table's name as the *sender* parameter when calling blinker.base.Signal.connect(). See sqlalchemy.orm.events.MapperEvents.before_insert() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (sqlalchemy.orm.mapper.Mapper) – The Mapper object which is the target of the event.
- **connection** (sqlalchemy.engine.Connection) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.
- **target** – The target object instance.

db_table_update
Emitted before a row inheriting from Base is updated in the database table. To only subscribe to update events for a specific table, specify the table's name as the *sender* parameter when calling blinker.base.Signal.connect(). See sqlalchemy.orm.events.MapperEvents.before_update() for more details.

Parameters

- **table** (*str*) – The name of the table for which the target object belongs.
- **mapper** (sqlalchemy.orm.mapper.Mapper) – The Mapper object which is the target of the event.
- **connection** (sqlalchemy.engine.Connection) – The SQLAlchemy connection object which is being used to emit the SQL statements for the instance.
- **target** – The target object instance.

3.6.5 Request Handler Signals

Signals which are emitted for events specific to individual HTTP requests. These signals use the respective instance of *KingPhisherRequestHandler* as the sender.

credentials_received
Sent when a new pair of credentials have been submitted.

Parameters

- **request_handler** – The handler for the received request.
- **username** (*str*) – The username of the credentials that were submitted.
- **password** (*str*) – The password of the credentials that were submitted.

email_opened
Sent when a request for the embedded image is received.

Parameters **request_handler** – The handler for the received request.

request_handle
Sent after a new HTTP request has been received and is about to be handled. This signal is suitable for implementing custom request handlers or aborting requests. This signal is emitted after *request_received* to allow subscribers the opportunity to handle requests themselves.

Note: If a request has been handled by the signal, the signal handler must raise the *KingPhisherAbortRequestError* exception to prevent further processing.

Parameters **request_handler** – The handler for the received request.

request_received
Sent when a new HTTP request has been received and is about to be handled. This signal is *not* suitable for implementing custom request handlers or aborting requests. This signal is emitted before *request_handle* allowing subscribers to be notified before a request may be blocked.

Parameters **request_handler** – The handler for the received request.

response_sent
Sent after a response to an HTTP request has been sent to the client. At this point headers may be added to the response body.

Parameters

- **request_handler** – The handler for the received request.
- **code** (*int*) – The HTTP status code that was sent in the response.
- **message** (*str*) – The HTTP message that was sent in the response.

rpc_method_call
Sent when a new RPC request has been received and it's corresponding method is about to be called.

Parameters

- **method** (*str*) – The RPC method which is about to be executed.
- **request_handler** – The handler for the received request.
- **args** (*tuple*) – The arguments that are to be passed to the method.
- **kwargs** (*dict*) – The key word arguments that are to be passed to the method.

rpc_method_called
Sent after an RPC request has been received and it's corresponding method has been called.

Parameters

- **method** (*str*) – The RPC method which has been executed.

- **request_handler** – The handler for the received request.
- **args** (*tuple*) – The arguments that were passed to the method.
- **kwargs** (*dict*) – The key word arguments that were passed to the method.
- **retval** – The value returned from the RPC method invocation.

rpc_user_logged_in
Sent when a new RPC user has successfully logged in and created a new authenticated session.

Parameters

- **request_handler** – The handler for the received request.
- **session** (*str*) – The session ID of the newly logged in user.
- **name** (*str*) – The username of the newly logged in user.

rpc_user_logged_out
Sent when an authenticated RPC user has successfully logged out and terminated their authenticated session.

Parameters

- **request_handler** – The handler for the received request.
- **session** (*str*) – The session ID of the user who has logged out.
- **name** (*str*) – The username of the user who has logged out.

visit_received
Sent when a new visit is received on a landing page. This is only emitted when a new visit entry is added to the database.

Parameters **request_handler** – The handler for the received request.

3.6.6 Server Signals

Signals which are emitted for a *KingPhisherServer* instance.

server_initialized
Sent when a new instance of *KingPhisherServer* is initialized.

Parameters **server** – The newly initialized server instance.

CHAPTER 4

Plugins

Starting with version v1.3.0 King Phisher includes a plugin system. Both client and server plugins are supported with the common functionality for the two being provided by the `plugins` module and then extended by the irrespective implementations in *king_phisher.client.plugins* and *king_phisher.server.plugins*.

King Phisher supports loading plugins to allow the user to add additional features out side of what is supported by the main-stream application. These plugins are implemented as Python modules which define a `Plugin` class that is the respective plugins entry point as well as the host for various pieces of metadata in the form of class-attributes.

4.1 Plugin Compatibility

Due to the way in which plugins are defined as classes with meta-data provided in their attributes, they need to be able to be imported regardless of compatibility restraints. The base *PluginBase* class offers a number of attributes which can be defined to indicate it's compatibility requirements.

4.1.1 Minimum Python Version

A minimum required version of Python can be specified in the *req_min_py_version* attribute. This allows the plugin to specify that it needs libraries that, for example require version 3.5 at least. The version is defined as a point separated string such as `"3.5.1"`.

The default class value is `None` which causes the plugin to be compatible with all versions of Python supported by King Phisher.

4.1.2 Minimum King Phisher Version

A minimum required version of King Phisher can be specified in the *req_min_version* attribute. This should be used to indicate the version in which API changes were made that the plugin relies upon. The value of this attribute must be a string which can be parsed with Python's `distutils.version.StrictVersion` class for comparison.

The default class value is the first version of King Phisher which introduced the plugin API.

4.1.3 Required Python Packages

Sometimes modules may need additional Python packages and modules to be available in order to function properly. This can be problematic as the modules often need to be imported at the top level which normally would prevent the plugin from loading. In order to avoid this, plugin authors must wrap the import statement using Python's exception handling and define a variable to indicate whether or not the module is available.

This variable then needs to be added to the *req_packages* attribute. This attribute is a dictionary whose keys are the names of packages which are required with values of their availability. Using this method a plugin which requires the externally provided package "foo" can be loaded into King Phisher allowing it to correctly alert the user in the event that the "foo" package can not be loaded. It's highly recommended that the required packages be described in the plugins description.

The default class value is an empty dictionary meaning that now additional packages or modules are required. This is only suitable for use with plugins that do not need any additional packages or modules beyond what Python, King Phisher itself and King Phisher's direct dependencies provide.

4.1.4 Supported Platforms

Plugins that only work on specific platforms such as Windows or Linux can specify which platforms they support using the *req_platforms* attribute. This is a string of tuples that correspond to Python's `platform.system()` function. When defined, the plugin will only be compatible if the current platform is contained within the whitelist.

The default class value is compatible with all platforms.

4.1.5 Example

The following is a commented example of a basic client plugin with compatibility requirements.

```
import king_phisher.client.plugins as plugins
import king_phisher.client.gui_utilities as gui_utilities

try:
    import foobar
except ImportError:
    has_foobar = False  # catch the standard ImportError and set has_foobar to False
else:
    has_foobar = True   # no errors occurred so foobar was able to be imported

class Plugin(plugins.ClientPlugin):
    authors = ['Spencer McIntyre']
    title = 'Compatibility Demo'
    description = """
    A basic plugin which has compatibility requirements. It needs the 'foobar'
    Python package to be installed.
    """
    req_min_py_version = '4.0'  # this is the required minimum version of Python
    req_min_version = '1337.0'  # this is the required minimum version of King Phisher
    req_packages {
        'foobar': has_foobar    # whether or not foobar was able to be imported
    }
    req_platforms = ('Linux',)  # this module is only compatible with Linux
    # plugin method definitions continue
```

4.2 Client Plugins

For information on how client plugins are installed, see the Client Plugins wiki page.

Client plugins need to inherit from the *ClientPlugin* class which provides the basic outline. Client plugins have access to a dictionary for persistent configuration storage through the *config* attribute. In order for the plugin's meta-data to be available to the GUI, class attributes are used. This allows information such as the title, description, etc. to be accessed without initializing the class.

4.2.1 Plugin Manager

When the Plugin Manager window is loaded, all available plugins are loaded in order for their information to be retrieved from the class attributes. This is the effective equivalent of importing the module in Python. When the module is enabled, an instance of the Plugin class created allowing it to fulfill its intended purpose.

Reloading Plugins

Plugin modules and classes can be "reloaded" to allow changes made to the plugin on disk to take effect. This can be accomplished by right clicking the plugin and selecting the "Reload" option from the manager window. If an enabled plugin is reloaded, it will first be disabled before being re-enabled causing it to lose any state information it may have been storing.

Plugin Compatibility

Plugin modules that have requirements can have their compatibility checked by viewing the plugin information pane. This includes a simple yes or no regarding whether all of the plugin's requirements are met and the plugin is thus compatible. Additional information regarding the specific requirements a particular plugin has can be accessed by clicking the `Compatible` link which will show each of the requirements, their values and whether or not they are met. This allows users to easily determine why a particular plugin may not be compatible.

4.2.2 Plugin Options

Client plugins have special *ClientOption* classes available to them for specifying options that the user can set. The *king_phisher.client.plugins.ClientOptionMixin.__init__()* method offers additional parameters such as *display_name* to configure the information shown to the end user in the configuration widget.

The following are the different option classes available for client plugins:

- *ClientOptionBoolean*
- *ClientOptionEnum*
- *ClientOptionInteger*
- *ClientOptionPath*
- *ClientOptionPort*
- *ClientOptionString*

4.2.3 Example

The following is a commented example of a basic "Hello World" plugin.

```
import king_phisher.client.plugins as plugins
import king_phisher.client.gui_utilities as gui_utilities

try:
    # imports that may not be available need to be an exception handler so
    # the plugin module will still load
    import advancedhttpserver
except ImportError:
    # set a variable to whether this package is available or not for later use
    has_ahs = False
else:
    has_ahs = True

# this is the main plugin class, it is necessary to inherit from plugins.ClientPlugin
class Plugin(plugins.ClientPlugin):
    authors = ['Spencer McIntyre']  # the plugins author
    title = 'Hello World!'          # the title of the plugin to be shown to users
    description = """
    A 'hello world' plugin to serve as a basic template and demonstration. This
    plugin will display a message box when King Phisher exits.
    """                             # a description of the plugin to be shown to users
    homepage = 'https://github.com/securestate/king-phisher-plugins'  # an optional␣
↪home page
    options = [                     # specify options which can be configured through␣
↪the GUI
        plugins.ClientOptionString(
            'name',                                 # the name of the option as it will␣
↪appear in the configuration
            'The name to which to say goodbye.',  # the description of the option as␣
↪shown to users
            default='Alice Liddle',                 # a default value for the option
            display_name='Your Name'                # a name of the option as shown to␣
↪users
        ),
        plugins.ClientOptionBoolean(
            'validiction',
            'Whether or not this plugin say good bye.',
            default=True,
            display_name='Say Good Bye'
        ),
        plugins.ClientOptionInteger(
            'some_number',
            'An example number option.',
            default=1337,
            display_name='A Number'
        ),
        plugins.ClientOptionPort(
            'tcp_port',
            'The TCP port to connect to.',
            default=80,
            display_name='Connection Port'
        )
    ]
    req_min_py_version = '3.3.0'         # (optional) specify the required minimum␣
↪version of python
```

(continues on next page)

(continued from previous page)

```
    req_min_version = '1.4.0'             # (optional) specify the required minimum␣
↪version of king phisher
    req_packages = {                      # (optional) specify a dictionary of␣
↪required package names
        'advancedhttpserver': has_ahs     # set from within the exception handler when␣
↪importing
    }
    req_platforms = ('Linux', 'Windows') # (optional) specify the supported platforms
    version = '1.0'                       # (optional) specify this plugin's version
    # this is the primary plugin entry point which is executed when the plugin is␣
↪enabled
    def initialize(self):
        print('Hello World!')
        self.signal_connect('exit', self.signal_exit)
        # it is necessary to return True to indicate that the initialization was␣
↪successful
        # this allows the plugin to check its options and return false to indicate a␣
↪failure
        return True

    # this is a cleanup method to allow the plugin to close any open resources
    def finalize(self):
        print('Good Bye World!')

    # the plugin connects this handler to the applications 'exit' signal
    def signal_exit(self, app):
        # check the 'validiction' option in the configuration
        if not self.config['validiction']:
            return
        gui_utilities.show_dialog_info(
            "Good bye {0}!".format(self.config['name']),
            app.get_active_window()
        )
```

4.3 Server Plugins

For information on how client plugins are installed, see the Server Plugins wiki page.

Server plugins need to inherit from the *ServerPlugin* class which provides the basic outline. Server plugins have access to their respective configurations from the *config* attribute. This data is loaded from the server's configuration file and while it can be changed at runtime, the changes will not be kept after the server has stopped.

A plugin that needs to store data persistently can use the *storage* attribute which acts as a simple key value store and is backed by the database. Values stored in this must be able to be serialized making it impossible to directly store custom objects.

Server plugins can hook functionality by utilizing the `signals` module. This allows plugins to provide functionality for specific events.

4.3.1 Adding RPC Methods

Server plugins can provide new RPC methods that are available to client plugins and the client's RPC terminal. This allows server plugins to provide extended functionality for use by these other components.

Registering new RPC methods is as simple as calling the *register_rpc()* method. This function, like signal handlers, takes a method as an argument for use as a call back. This method is then called when the RPC function is invoked. The return value of this method is then returned to the caller of the RPC function. The method will automatically be passed the current *KingPhisherRequestHandler* instance as the first argument (after the standard "self" argument for class methods as applicable). Additional arguments after that accepted from the RPC invocation.

The following is an example of two custom RPC methods.

```
# ... other initialization code
class Plugin(plugins.ServerPlugin):
    # ... other initialization code
    def initialize(self):
         self.register_rpc('add', self.rpc_add)
         self.register_rpc('greet', self.rpc_greet)
         return True

    # this example takes two arguments to be invoked and returns their sum
    # >>> rpc('plugins/example/add', 1, 2)
    # 3
    def rpc_add(self, handler, number_1, number_2):
        return number_1 + number_2

    # this example takes no arguments but accesses the rpc_session to
    # retrieve the current user name
    # >>> rpc('plugins/example/greet')
    # 'Hello steiner'
    def rpc_greet(self, handler):
        rpc_session = handler.rpc_session
        return 'Hello ' + rpc_session.user
```

4.3.2 Example

The following is a commented example of a basic "Hello World" plugin.

```
import king_phisher.plugins as plugin_opts
import king_phisher.server.plugins as plugins
import king_phisher.server.signals as signals

# this is the main plugin class, it is necessary to inherit from plugins.ServerPlugin
class Plugin(plugins.ServerPlugin):
    authors = ['Spencer McIntyre']  # the plugins author
    title = 'Hello World!'
    description = """
    A 'hello world' plugin to serve as a basic template and demonstration.
    """
    homepage = 'https://github.com/securestate/king-phisher-plugins'
    options = [  # specify options which need to be set through the configuration file
        plugin_opts.OptionString(
            'name',               # the options name
            'the name to greet',  # a basic description of the option
            default=None          # a default value can be specified to
        )
    ]
    req_min_version = '1.4.0'     # (optional) specify the required minimum version␣
↪of king phisher
```

(continues on next page)

(continued from previous page)

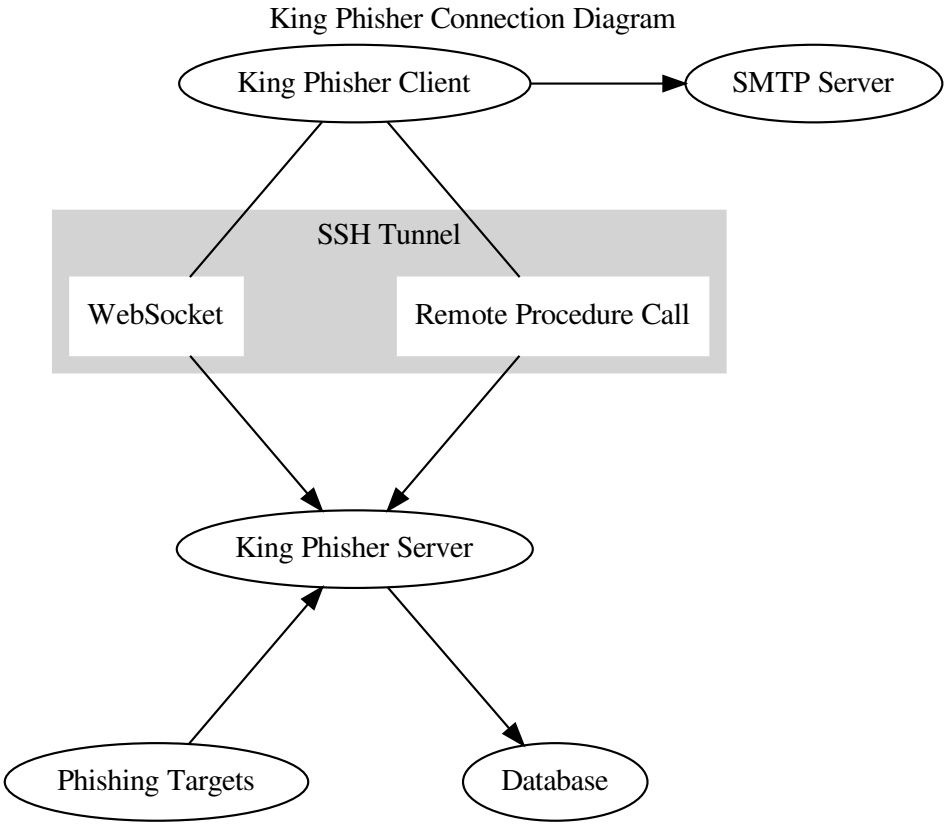
```
    version = '1.0'               # (optional) specify this plugin's version
    def initialize(self):
        self.logger.warning('hello ' + self.config['name'] + '!')
        # connect to a signal via it's object in the signals module
        signals.server_initialized.connect(self.on_server_initialized)
        return True

    def on_server_initialized(self, server):
        self.logger.warning('the server has been initialized')
```

CHAPTER 5

Development References

5.1 Architecture Overview

The following diagram outlines the generic architecture of how the major components both contained in and used by King Phisher interact.



In the diagram above, all major components (shown in oval shapes) can technically coexist on the same host system. In this case the term "host" refers to a single OS installation whether that be a Virtual Machine or not. It is however recommended to at a minimum install the King Phisher client and server components on separate hosts for production deployments.

The King Phisher project consists of the client and server components. The major responsibilities of each are noted as follows:

5.1.1 Client Responsibilities

- **Creating Campaigns** – The client facilitates creating new campaigns through its user interface. Once the campaign user is done adjusting the settings for the new campaign, the client uses RPC to transfer the information to the King Phisher server.
- **Sending Email** – The client is responsible for editing, rendering and ultimately sending phishing messages through an external SMTP server. Once a message is sent, the client notifies the King Phisher server via an RPC call with the applicable details such as who the message was sent to.

- **Processing Campaign Data** – Once data has been collected on the King Phisher server for a particular campaign, the client retrieves it using GraphQL queries over RPC. Once the data has been transferred it is displayed through the user interface.

5.1.2 Server Responsibilities

- **Handling HTTP(S) Requests** – The server handles all HTTP and HTTPS requests either from phishing targets or the King Phisher client (which uses a form of RPC over HTTP).
- **Tracking Campaigns** – The server tracks the status of campaigns through the configured database backend. This allows the King Phisher client to disconnect and shutdown once it is done making changes.
- **Dispatching Campaign Notifications** – While tracking campaign data, the server publishes event notifications to various pieces of subscriber code. Plugins utilizes this model to subscribe to events and execute arbitrary routines (such as sending alerts to end users) when they are received. The King Phisher client can also subscribe to a subset of events which are forwarded over websockets.

5.1.3 Login Process

The following steps outline the procedure taken by the client to open a a connection to, and authenticate with the server for communication.

1. The client communicates to the server through an SSH tunnel which it establishes first. This requires the client to authenticate to the host on which the server is running.
2. The client issues an RPC request through the established SSH tunnel to the `version` endpoint to determine compatibility.
3. The client issues an additional RPC request through the established SSH tunnel, this time to the `login` endpoint to authenticate and create a new session.
4. The client opens a websocket connection through the SSH tunnel to subscribe to and receive events published by the server in real time.

At this point the client is fully connected to the server.

5.1.4 Signal Architecture

Both the client and server utilize and provide functionality for signal-driven callbacks. The two use different backends, but in both cases, there is a core interface through which King Phisher signals are published to registered callback functions as events. The signal subsystem is particularly useful for plugins to modify system behavior.

Client Signals

Due to the nature of the client application using GTK, the GObject Signal functionality is used to provide the core of client events. These events are defined in the *GObject Signals* documentation. These signals are published by particular object instances, with the most notable being the `KingPhisherClientApplication`.

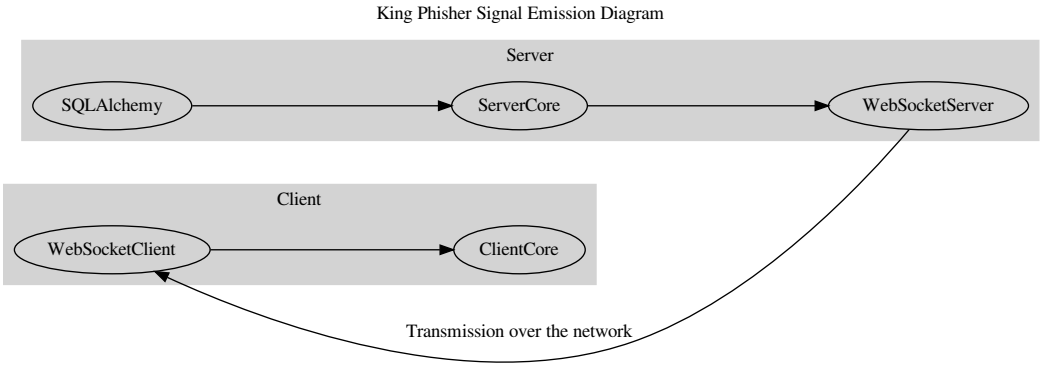
Server Signals

The server utilizes the `blinker` module to support application events. This interface is defined and documented in *Server Signals* documentation. Server signals are centrally located within the `signals` module from which that can be both connected to and emitted.

Signal Forwarders

Due to the both the client and server having a centralized signal mechanism, there are notable components which both forward signals to and from other components to make the interface consistent.

Name	Direction	Description
SQLAlchemy	**From:** SQLAlchemy **To:** Server Core	Forwards events from SQLAlchemy into the server's core signal dispatcher. This allows server components to connect to SQLAlchemy signals for database events through the central interface.
WebSocket Server	**From:** Server Core **To:** Web-Socket Clients	Forwards events from the server's core signal dispatcher to connected and subscribed client web sockets. This effectively enables subscribers to receive a subset of server signals.
WebSocket Client	**From:** WebSocket Client **To:** Client Core	Forwards events received from the web sockets to the client's core signal dispatcher. This effectively enables client components to receive a subset of server signals.



5.2 Modules

The project's code base is split among multiple Python modules under the primary *king_phisher* package. Code which is not specific to either the client or server code bases is directly in the root of the *king_phisher* package with code that is specific to either the client or server being under either the *king_phisher.client* sub-package or *king_phisher.server* sub-package respectively.

5.2.1 Special Modules

Some modules have special designations to identify them as having particular qualities.

Clean Room Modules

Modules that qualify for the "Clean Room" classification are suitable for use during the early phases of the application's initialization. They may also be used for general purposes.

- Modules must not import any code which is not either included in the Python standard library or packaged with King Phisher. For example, `os`, `sys`, and *`king_phisher.startup`* may be imported while `advancedhttpserver`, `jinja2`, and `smoke_zephyr` may not.
- Modules may only import other King Phisher modules which also have the "Clean Room" classification.

Modules with this designation have the following comment banner included in their source file just below the standard splat.

```
##############################################################################
#
# CLEAN ROOM MODULE
#
# This module is classified as a "Clean Room" module and is subject to
# restrictions on what it may import.
#
# See: https://king-phisher.readthedocs.io/en/latest/development/modules.html#clean-
↪room-modules
#
##############################################################################
```

5.3 Environment Variables

The following environment variables can be set to change normal operation. None of them are required to be set under normal circumstances.

Variable Name	Variable Description
General Purpose	
`KING_PHISHER_DATA_PATH`	Paths to search for data files
`KING_PHISHER_DEV_KEY`	Path to a development key
`KING_PHISHER_GLADE_FILE`	Name of the client Glade UI data file
Testing Specific	
`KING_PHISHER_TEST_GEOIP_DB`	The GeoIP database used for unit tests
`KING_PHISHER_TEST_OFFLINE`	Skip unit tests which require a network connection

5.4 Style Guide

It's important for a project to have a standardized style for it's code. The King Phisher project, being a Python project follows the PEP-8 style guide with the following notable exceptions:

- Do use hard tabs instead of spaces.
- Do not use more than one consecutive blank line, ever.
- Do limit lines of code to 120 characters long instead of 79.
  - Do limit documentation lines to 80 characters long.

- Do use single quotes for strings with the exception of template strings (such as those used by `str.format`) and documentation strings which should use triple double-quotes.
- Optionally use additional spaces within a line for visual grouping. For example, when defining a long list of constants use additional spaces after the name and before the value to align all the values on the right.

5.4.1 Multi Line Indentation

Use hanging indentation for parenthesized statements. This is to say, the last non-whitespace character of the line should be the opening parenthesis with each subsequent line being indented until the closing parenthesis. Furthermore, in the case that this style is used, each expression should be on a separate line.

Example:

```
# good (standard one-line invocation)
this_function(takes_two, different_arguments)

# good (multi-line invocation)
this_other_function_has_a_longer_name(
    and_also_takes_two,
    different_arguments
)

# bad
this_other_function_has_a_longer_name(and_one_argument_up_here,
    and_another_down_here
)
```

This same style is applied to multi-line list, tuple and dictionary definitions with the bracket, or curly-brace taking the place of the opening and closing parenthesis as appropriate.

5.4.2 Special Method Names

Some functions (and methods) have special prefixes or suffixes to denote specific compatibility. Prefixes are permitted to be either preceded by a single or double underscore to denote internal or private use respectively.

Name	Type	Details
`_tsafe`	Suffix	Non-Main GUI thread safe
`async_rpc_cb_`	Prefix	An asynchronous client RPC callback
`signal_`	Prefix	GTK signal handler

5.4.3 English Verbiage

Use full, complete and grammatically correct sentences for all documentation purposes. This includes class, function, attribute, and parameter documentation. Additionally, proper sentences should be used for any messages that are displayed to end users with the notable exception of log messages. Log messages are to be entirely lowercase with the exception of acronyms which are currently inconsistently cased. Either all capital letters or all lower case letters are acceptable for acronyms within log messages.

5.4.4 Documentation

When documenting a function, use the grammar provided by Sphinx. Documentation strings should be surrounded by triple double quotes (`"""`). There should be a single blank line between the body of the description and the parameter and return definitions.

```
def add_two_numbers(x, y):
    """
    Add two values specified as *x* and *y* together returning their sum.

    :param int x: The value for the first number to return.
    :param int y: The value for the second number to return.
    :return: The sum of the two values.
    :rtype: int
    """
    return x + y
```

Native Python types are able to be specified on the `:param` line. More complex types, such instances of classes defined by modules in the project must be defined on a separate line using a dedicated `:type` annotation. See the Sphinx documentation for the Python Domain.

```
:param foo: The widget this function uses.
:type foo: :py:class:`~the_full.module_path_to.Foo`
```

5.4.5 CLI Arguments

For utilities which take arguments on the command line, the following default values should be supported.

Argument Flag	Meaning
`-h / --help`	Display help information
`-v / --version`	Display version information
`-V / --verbose` *	Enable verbose output
`-L / --log` *	Set the log level to use

* These values are optional, but should not be overridden.

5.4.6 Log Levels

When logging messages, the following levels should be used as described.

CRITICAL Reserved for when an unrecoverable error has occurred that stops the application from running.

Examples:

- A required library, module or resource file is missing.
- An unknown exception occurs which is raised to the main method of an application.

ERROR A recoverable error has occurred that stops the process from functioning as intended.

Examples:

- On a client, the user fails to authenticate successfully.
- A network socket failed to connect to a server.

WARNING A recoverable error has occurred that does not stop the process from functioning as intended.

Examples:

- On a server, a user fails to authenticate successfully.
- When information provided by the user is invalid and the user can be prompted for new information.

INFO High level information regarding what is happening in an application, should be use sparingly within loops.

Examples:

- Listing resources that are being loaded and processed.
- The child pid when `fork()` is used.

DEBUG Low level information regarding what is happening in an application including the values of variables, this may be used more frequently within loops.

Examples:

- Printing identifying information for threads that are spawned.
- Printing the value of arguments that are passed into functions.

5.5 Classifiers

Classifier strings can be applied to complex objects to describe arbitrary qualities that are desirable to determine pragmatically. For example, a classifier can be used to describe that a client plugin is intended to be used for Spam evasion purposes or that a server template is intended to be used for gathering credentials. Data structures which use classifiers expose them as a list of strings to allow for multiple classifiers to be defined. When defining a classifier, it is important that the classifier is not unique as the purpose of the data is to identify objects by arbitrary traits.

5.5.1 Classifier Format

Classifiers take a simple format of one or more words separated by two colons (`::`). The words should be capitalized for consistency and are arranged in a hierarchical format. For example, `Foo :: Bar :: Baz` overlaps with `Foo :: Bar` and thus an object with the `Foo :: Bar :: Baz` classifier implicitly contains `Foo :: Bar` and does not require it to be explicitly defined. As such, while searching classifiers, a query term of `Foo :: Bar` must match `Foo :: Bar :: Baz`.

5.5.2 Common Classifiers

The following is a reference of common classifiers, mostly used by external components such as plugins and templates.

Plugin :: Client – An executable plugin to be loaded by the King Phisher client that will typically provide new or modify existing functionality.

Plugin :: Client :: Email :: Attachment – A client plugin which creates or modifies email attachments.

Plugin :: Client :: Email :: Spam Evasion – A client plugin which can be used for the purpose of Spam filter evasion.

Plugin :: Client :: Tool - A client plugin which provides generic utility functionality typically for the purposes of convenience.

Plugin :: Client :: Tool :: Data Management – A client plugin which manages data in some fashion such as for organization or archival purposes.

Plugin :: Server – An executable plugin to be loaded by the King Phisher server that will typically provide new or modify existing functionality.

Plugin :: Server :: Notifications – A server plugin which dispatches notifications through an arbitrary, plugin-provided means.

Plugin :: Server :: Notifications :: Alerts – A server plugin which dispatches notifications through the alerts interface. Notifications through the alerts interface can be self-managed by individual users as opposed to being server-wide.

Script :: CLI – An object, typically a plugin which provides an interface to be executed as a standalone script from the command line.

Template :: Site – A template for use by the King Phisher server to be presented to targeted users when they visit. When used as parting of a phishing campaign, a site template provides the content to be viewed by users which have fallen for the pretext.

Template :: Site :: Credentials – A site template which incorporates functionality for prompt the visitor for and recording submitted credentials.

Template :: Site :: Payload – A site template which will provide the visitor with a payload of some kind (typically an executable file) with the intention of having them run it.

Template :: Site :: Training – A site template that informs the user that they were involved in a phishing exercise, failed and attempts provide information for training purposes.

5.6 Release Steps

This document contains the steps that are followed for each point version release of King Phisher.

5.6.1 Pre Release Steps

1. Test and fix any issues with the Windows MSI build
2. Ensure unit tests pass with Python 3.4+
3. Remove the version label
4. Create the final Windows MSI build
5. Update the change log

5.6.2 Release Steps

1. Create a final signed commit on the dev branch and push it to GitHub
2. Merge dev into master and push master to GitHub
3. Create and push a signed tag of the release commit
4. Create a new release on GitHub
    1. Upload the final Windows build
    2. Insert the changes from the change log
    3. Insert the MD5, SHA1 and SHA512 hashes of the Windows build
5. Publicize the release

5.6.3 Post Release Steps

1. Open a new issue with the Kali bug tracker notifying them of the release
2. Increment the version number on the dev branch and reset the version label
3. Update the Python packages list in Pipfile
    1. List the outdated packages with: `pipenv update --outdated`
    2. Update each one with: `pipenv install PACKAGE==VERSION`
    3. Manually synchronize `docs/requirements.txt`

5.7 Software Versions

King Phisher development needs to track the support of critical libraries it uses for compatibility purposes. This information is used to make decisions regarding dropping support for legacy systems.

5.7.1 Python Packages Reference Table

Last Updated: December 3rd, 2019 by Spencer McIntyre

Package	Reason	Pinned Version
graphene		2.1.8
graphene-sqlalchemy	Holds graphql-relay graphql-core <3	2.2.0
graphql-relay	Highest version for graphene-sqlalchemy	2.1.1
numpy	Required by Basemap in the setup.py file before installation	1.16.4
matplotlib	Windows build limitation	2.2.4
cryptography	Required by Paramiko 2.60	2.2.4
pyproj	Required by Basemap	
pygobject	Required for gi/gtk	
psycopg2	Required by SQLAlchmey	
markdown	Required by py-gfm (must be <3.0)	2.6.11
jsonschema	Minor unit tests failures	2.6.0

5.7.2 Operating System Reference Table

Last Updated: December 3rd, 2019 by Spencer McIntyre

Flavor	Software	Version
Backbox 5.1	GTK3	3.18.9
	Python3	3.5.2
Debian 7 (Wheezy) May 4th 2013	GTK3	3.4.2
	Python3	3.2.3
Debian 8 (Jessie) April 25th 2015	GTK3	3.14.5
	Python3	3.4.2
Debian 9 (Stretch) June 17th 2017	GTK3	3.22.11
	Python3	3.5.3

Continued on next page

Table 1 – continued from previous page

Debian 10 (Buster)	GTK3	3.22.29
	Python3	3.6.4
Fedora 24 June 21st 2016	GTK3	3.20.6
	Python3	3.5.1
Fedora 25 November 15th 2016	GTK3	3.22.2
	Python3	3.5.4
Fedora 26 July 11th 2017	GTK3	3.22.16
	Python3	3.6.4
Fedora 27 November 14th 2017	GTK3	3.22.24
	Python3	3.6.4
Fedora 28 May 1st 2018	GTK3	3.22.30
	Python3	3.6.5
Fedora 29 October 30th 2018	GTK3	3.24.1
	Python3	3.7.0
Fedora 30 April 30th 2019	GTK3	3.24.8
	Python3	3.7.3
Fedora 31 October 22nd 2019	GTK3	3.24.12
	Python3	3.7.4
Kali Rolling	GTK3	3.22.29
	Python3	3.7.2 [1]
Ubuntu 14.04 (Trusty) April 17th 2014	GTK3	3.10.8
	Python3	3.4.3
Ubuntu 16.04 (Xenial) April 21st 2016	GTK3	3.18.9
	Python3	3.5.2
Windows	GTK3	3.18.9
	Python3	3.4.4

[1] Kali Rolling is continuously updated. The version number noted was accurate as of the last time this document was updated.

5.7.3 Information Sources

Debian

Search using packages.debian.com.

Fedora

```
# use koji
sudo dnf install koji
# check the version of GTK3 for Fedora 24
koji latest-pkg --all f24 | grep -i gtk3
```

Windows

```
# run KingPhisher in debug mode
cd king-phisher
python KingPhisher -L DEBUG
# The first 7 lines of out put will contain gi.repository version information.
```

(continues on next page)

(continued from previous page)

```
# Get python version
python --version
```

5.8 Windows Build

Each release of King Phisher includes an MSI build of the client for easy use on Windows systems. Creating this build is one of the last steps prior to creating a new version release. The build is created using the Python cx_Freeze package.

Before the build can be created the PyGObject for Windows package must be installed. While installing this package, it prompts for which GNOME libraries are to be included. When the prompt appears the following packages should be selected.

- Base packages
- ATK
- GConf
- GDK-Pixbuf
- GTK+
- GTKSourceView
- GXML
- Pango
- Soup
- WebkitGTK

Once all packages have been installed and the King Phisher client is running with Python, the "tools/development/build_msi.bat" script can be executed to create the build. The build process will take a few minutes, but once completed an MSI installer file will be created in a new "dist" directory in the projects root folder.

5.8.1 Version Information

After building the MSI file, the custom properties will need to be added. These are added by right clicking on the MSI file, selecting properties, and then the custom tab where custom fields can be created. These need to include the Python version, and PyGI-AIO version utilized in making the build as text entries. Below is the name fields and example values.

Name	Example Value
Python Version	3.4.4rc1
PyGI-AIO Version	3.24.1 rev1
gi.repository GLib	2.52.1
gi.repository GObject	3.24.1
gi.repository GTK	3.18.9

5.8.2 Python 3.4 Build

As of King Phisher v1.8.0, the Windows client is built with Python 3.4. To install basemaps for Python 3.4 geos will need to be compiled for Windows. In addition to the packages in the "requirements.txt" file, `pypiwin32api`, and `numpy` will need to be installed manually.

For information on how to build geos on Windows with CMake visit: https://trac.osgeo.org/geos/wiki/BuildingOnWindowsWithCMake.

It is important that the same version of geos be built that is used with basemaps.

Once geos is complied the two generated DLLs `geos.dll` and `geos_c.dll` need to be copied to "[python34]libssite-packages".

Note: C++ 2010 Express and older will need to have the `floor` and `ceil` functions defined. These two functions are required by the geos library but are unavailable in older versions of the standard library.

5.8.3 CX Freeze version 5.0.1

After building and installing the MSI file, if the short cut link fails because it cannot `from . import xxx`, it is because the working directory for the shortcut is not set. To change this so builds have the working directory set automatically, the last line of "[python34]Libsite-packagescx_Freezewindist.py" needs to be updated from `None` to `"TARGETDIR"`.

The ouput example of lines 52-62 of cx_freeze's "windist.py" file, with change applied.

```
for index, executable in enumerate(self.distribution.executables):
    if executable.shortcutName is not None \
            and executable.shortcutDir is not None:
        baseName = os.path.basename(executable.targetName)
        msilib.add_data(self.db, "Shortcut",
                [("S_APP_%s" % index, executable.shortcutDir,
                        executable.shortcutName, "TARGETDIR",
                        "[TARGETDIR]%s" % baseName, None, None, None,
                        None, None, None, "TARGETDIR")])
```

CHAPTER 6

Change Log

This document contains notes on the major changes for each version of King Phisher. In comparison to the git log, this list is curated by the development team for note worthy changes.

6.1 Version 1.x.x

6.1.1 Version 1.16.0

In Progress

6.1.2 Version 1.15.0

Released v1.15.0 on September 24th, 2019

- Add support to select visible columns for tables in the Campaign tab
- Add support for printing pipenv running output in real time
- Windows build will now install PyPI requirements for plugins during installation
- Multiple bug fixes

6.1.3 Version 1.14.1

- Fixed the return value when loading already loaded SNI certificates

6.1.4 Version 1.14.0

Released v1.14.0 on August 1st, 2019

- Added the `Message-ID` MIME header to outgoing messages

- Attempt SSH authentication with all agent-provided SSH keys
- Deleted `Pipfile.lock` from repository to prevent hash issues between python interpreter versions
- Add `--three` to `pipenv install` and `pipenv --update` startup procedures to force use of Python 3
- Added server support for installing missing plugin requirements during initialization
- Added asynchronous RPC methods to the client
- Added GraphQL and database schema documentation
- Changed Target URL to Web Server URL in Campaign Editor
- Added the ability issue SSL Certificates through certbot

6.1.5 Version 1.13.1

Released v1.13.1 on April 19th, 2019

- Fixed broken references to *`start_process()`*
- Fixed a `KeyError` when creating a campaign for the first time (see: #365)
- Updated SQLAlchemy and Jinja2 libraries for security patches

6.1.6 Version 1.13.0

Released v1.13.0 on April 4th, 2019

- Added support for logging MFA tokens with credentials
- Added support for using regular expressions to validate credentials
- Automatically try to install plugin dependencies with pip from PyPi
- Added advanced, rule-based filtering support to the Campaign tabs
- Added site template metadata
  - Site templates can now include a metadata file for describing their content
  - The Campaign Assistant will help select a target URL based on available templates

6.1.7 Version 1.12.0

Released v1.12.0 on November 7th, 2018

- Added support for users to set their email address for campaign alerts via email
- Added additional plugin metadata fields for reference URLs and category classifiers
- Added additional documentation including an architecture overview for reference
- Multiple improvements to the client plugin manager
  - There is now an option to update plugins in the menu
  - Plugins can ship with dedicated documentation in markdown files that will be displayed
  - The GUI no longer locks up while tasks like downloading plugins are taking place
- Added the new `fetch` Jinja function and `fromjson` Jinja filter

- Added `campaign-alert-expired` and `campaign-expired` server signals
- Switched to using Pipenv to manage the environment and dependencies

6.1.8 Version 1.11.0

Released v1.11.0 on April 12th, 2018

- Updated to support matplotlib version 2.2.0
- Removed docker server support
- Multiple improvements to the installation script
  - Users can now specify a supported Linux distro when it is not automatically detected
  - The database connection string is kept to avoid PostgreSQL password resets
- Added support for setting message UID character set options
- Bumped the required minimum version of Python to 3.4 and GTK to 3.14
- Update Windows build to use pygi-aio-3.24.1_rev1 PyGObjects
- Multiple bug fixes

6.1.9 Version 1.10.0

Released v1.10.0 on March 16th, 2018

- Added a `campaign-alert` server signal for custom alert delivery mechanisms
- Use GraphQL for loading data instead of the legacy table-based API
- Support fault-tolerance when dispatching server signals
- Allow a country code to be set in users' phone numbers
- Visits will now be tracked if the landing page is any existing type
- Multiple RPC Terminal improvements
  - Fix a bug regarding line wrapping due to the `TERM` environment variable
  - Use ipython when it's installed
  - Added `%graphql` and `%graphql_file` magic commands
- Tweaks to the default MIME-encoded HTML message to reduce it's SpamAssassin score
- Modified client signals to allow better API control
  - Added `message-create` and `target-create` for modifying the respective objects
  - Added `message-send` and `target-send` to allow skipping the message and target
  - Removed the `send-message` and `send-target` signals in favor of the new ones

6.1.10 Version 1.9.0

Released v1.9.0 on November 22nd, 2017

- Support resetting plugins options to their respective defaults
- Moved Office 2007+ metadata removal to a new plugin
- Added support for installing plugins from remote sources through the UI
- Added timeout support for SPF DNS queries
- Support for installing on Arch Linux
- Multiple server improvements
  - Upgrade AdvancedHTTPServer to v2.0.11 to support async SSL handshakes
  - Support using an include directive in the server configuration file
  - Added a `request-handle` signal for custom HTTP request handlers
  - Removed `address` support from the server config in favor of `addresses`
  - Support `login` as an alias of the `username` parameter for credentials

6.1.11 Version 1.8.0

Released v1.8.0 on June 6th, 2017

- Install script now supports Red Hat Server 7
- Support the client on OS X by using Docker
- Support for issuing certificates with acme while the server is running
- Add a wrapping tool for certbot to make the process easier
- Updated `tools/cx_freeze.py` to build the King Phisher client in Python 3.4
- Updated documentation for the Windows build

6.1.12 Version 1.7.1

Released v1.7.1 on April 14th, 2017

- Bug fix in the Windows build for HTTPS connections from the requests package

6.1.13 Version 1.7.0

Released v1.7.0 on April 4th, 2017

- Better error messages for malformed server configuration files
- Support for sending to targets via To / CC / BCC fields
- New features for client and server plugins
- Add comparison of "trained" statistics to the campaign comparison
- Support for including and importing Jinja templates from relative paths
- Support for including custom HTTP headers in server responses

- New feature to import Campaigns from XML files
- Support for emails address with longer top level domain names

6.1.14 Version 1.6.0

Released v1.6.0 on January 31st, 2017

- Support negotiating STARTTLS with SMTP servers that support it
- Support for real time event publishing to the client
- Support for a new GraphQL API for more efficient data queries
- More flexibility in configuring server logging
- Add persistent storage for server plugin data
- Add a Jinja function to check if a password is complex
- Add client `message-data-export` and `message-data-import` signals
- King Phisher now starts with Python3 by default
- `tools/install.sh` now creates a backup of server_config.yml when present
- Minor bug fixes
  - Minor CSS fixes
  - Special characters now display in the UI correctly

6.1.15 Version 1.5.2

Released v1.5.2 on December 23rd, 2016

- Minor bug fixes
  - Use Default SMS sender to fix SMS subscription with T-Mobile
  - Upgrade AdvancedHTTPServer to v2.0.6 to fix select polling
  - Corrected issue when attachment file is inaccessible
  - Fixed issue when message file directory is gone
  - Fixed server side encoding error with basic auth
  - Fixed TypeError handling while rendering templates
  - Fixed a unicode bug when processing targets csv
  - Fixed install.sh script for CentOS7 and python3
  - Fixed show exception dialog with Glib idle_add
  - Fixed a logic bug causing premature SMTP reconnects
  - Fixed Webkit-1 load_string Null error

6.1.16 Version 1.5.1

Released v1.5.1 on October 3rd, 2016

- Automated installation script improvements
  - Backup an existing server configuration file
  - Log warnings when the PostgreSQL user exists
- Improve the Metasploit plugin for session notifications via SMS
- Support exporting credentials for use with Metasploit's `USERPASS_FILE` option

6.1.17 Version 1.5.0

Released v1.5.0 on September 22nd, 2016

- Added an SPF button to the client for on demand SPF record checking
- Fixed missing packages in the Windows build for timezone data
- Transitioned to the dnspython package for Python 2.x and 3.x

6.1.18 Version 1.4.0

Released v1.4.0 on August 5th, 2016

- Added additional Jinja variables for server pages
- Upgraded to AdvancedHTTPServer version 2
  - Added support for binding to multiple interfaces
  - Added support for multiple SSL hostnames via SNI
- Support for plugins in the server application
- Added server signals for event subscriptions in plugins
- Updated the style for GTK 3.20
- Start to warn users about the impending Python 2.7 deprecation
- Change to installing for Python 3
- Added an uninstallation script

6.1.19 Version 1.3.0

Released v1.3.0 on May 17th, 2016

- Added automatic setup of PostgreSQL database for the server
- Server bug fixes when running on non-standard HTTP ports
- Added completion to the messaged editor
- Support for plugins in the client application
- Added a client plugin to automatically check for updates
- Added a client plugin to generate anonmous statistics

- Added debug logging of parameters for key RPC methods
- Lots of Python 3.x compatiblity fixes

6.1.20 Version 1.2.0

Released v1.2.0 on March 18th, 2016

- SSH host key validation
- Install script command line flags
- Support for authenticating to SMTP servers
- Style and compatibility changes for Kali

6.1.21 Version 1.1.0

Released v1.1.0 on December 30th, 2015

- Added an option to send a message to a single target
- Support for sending calendar invite messages
- Added PostgreSQL setup to the installer
- Support for exporting to Excel
- Added a Jupyter notebook for interactive data analysis
- Added additional campaign filtering options
- Support for removal of metadata from Microsoft Office 2007+ documents

6.1.22 Version 1.0.0

Released v1.0.0 on October 15th, 2015

- Moved templates to a dedicated separate repository
- Added a custom theme for the client
- Added support for two factor authentication with TOTP
- Support for specifying an img style attribute for inline images in messages

6.2 Version 0.x.x

6.2.1 Version 0.3.0

Released v0.3.0 on August 21st, 2015

- Added a new campaign creation assistant
- Support for expiring campaigns at a specified time
- Track more details when messages are opened such as the IP address and User Agent
- Support for tagging campaign types

- Support for organizing campaigns by companies
- Support for storing email recipients department name
- Support for collecting credentials via Basic Auth

6.2.2 Version 0.2.1

Released v0.2.1 on July 14th, 2015

- Added syntax highlighting to the message edit tab
- Technical documentation improvements, including documenting the REST API
- Support reloading message templates when they change from an external editor
- Support for pulling the client IP from a cookie set by an upstream proxy
- Support for embedding training videos from YouTube
- Added a Metasploit plugin for using the REST API to send SMS messages
- Support for exporting visit information to GeoJSON

6.2.3 Version 0.2.0

Released v0.2.0 on April 28th, 2015

- Added additional graphs including maps when basemap is available
- Added geolocation support
- Made dashboard layout configurable
- Support for cloning web pages
- Support for installing on Fedora
- Support for running the server with Docker

6.2.4 Version 0.1.7

Released v0.1.7 on February 19th, 2015

- Added make_csrf_page function
- Added server support for SSL
- Support verifying the server configuration file
- Added a desktop file and icon for the client GUI
- Added support for operating on multiple rows in the client's campaign tables
- Support starting an external SFTP application from the client
- Tweaked miscellaneous features to scale for larger campaigns (35k+ messages)
- Updated AdvancedHTTPServer to version 0.4.2 which supports Python 3
- Added integration for checking Sender Policy Framework (SPF) records

6.2.5 Version 0.1.6

Released v0.1.6 on November 3rd, 2014

- Migrated to SQLAlchemy backend (SQLite will no longer be supported for database upgrades)
- Added additional documentation to the wiki
- Enhanced error handling and UI documentation for a better user experience
- Support for quickly adding common dates and times in the message editor

6.2.6 Version 0.1.5

Released v0.1.5 on September 29th, 2014

- Added support for inline images in emails
- Import and export support for message configurations
- Highlight the current campaign in the selection dialog

6.2.7 Version 0.1.4

Released v0.1.4 on September 4th, 2014

- Full API documentation
- Install script for Kali & Ubuntu
- Lots of bug fixes

6.2.8 Version 0.1.3

Released v0.1.3 on June 4th, 2014

- Jinja2 templates for both the client and server
- API version checking to warn when the client and server versions are incompatible

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

King Phisher REST API

/_

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z