Reference for Pandora FMS Development

Reference for Pandora FMS Development

General architecture of Pandora FMS code

For a separate and detailed explanation about Pandora FMS database structure see the article Pandora FMS engineering.

How to make the links compatible

For all links, it is necessary to use the function `ui_get_full_url`. Before calling the function, it is necessary to include `functions_ui.php`.

- When the URL is needed for refreshing, for example:

```
$url_refresh = ui_get_full_url();
```

- When a URL is needed for a relative path, for example:

```
$url = ui_get_full_url("/relative/path/file_script.php");
```

- JavaScript:

```
<?php
  ...
  $url = ui_get_full_url("/relative/path/file_script.php");
  ...
?>

<script type="text/javascript">
  ...
  jQuery.post ('<?php $url; ?>',
    {
     ...
    });
 ...
</script>
```

- In special cases, it is not necessary to use this function, for example direct links to the `index.php`:

```
echo '<form method="post"
action="index.php?param=111?param=222?param=333?param=444?param=555?param=666">'
;
```

PFMS Web console execution entry points

PFMS Web console only has a few web application execution entry points, unlike other web

applications, for example:

- WordPress only has one frontend entry point and one backend entry point for administration.
- SME web development that normally every PHP file is a possible entry point.

Installation

This entry point is used to perform an installation of PFMS Web console and the database. Once the installation is finished, PFMS Web console requests the file deletion for security reasons.

```
install.php
```

Normal Execution

All user interaction on the console is done through this entry point.

```
index.php
```

AJAX requests

In all AJAX requests, user permissions are checked: To make them consistent and easy to maintain, they are performed through this file by passing by GET or POST method the parameter page parameter, the relative address of the script to execute.

```
ajax.php
```

Mobile console

For mobile terminals that have a significantly smaller screen than a computer monitor, Pandora FMS has a reduced version of the console for these devices, reduced both in visual aspect and simplified in its feature.

```
mobile/index.php
```

API

Pandora FMS has an API with which third party applications can interact through a simple channel through port 80 and with the HTTP/HTTPS protocol.

The strengthening of this script is by means of two parameters, validating both:

- The IP address of the running client is in the list of configured valid IP addresses of PFMS Web console or matches the regular expression also stored in the same place.
- The *token* API configured by PFMS Web console has been passed as a parameter.

```
include/api.php
```

Special cases

Within PFMS Web console there are several special cases for entry points, usually to avoid login or general processing being done at the main entry point (`index.php` of the root).

Cron Tasks Extension

This extension by means of a wget command in the cron can execute the cron tasks assigned to it without asking for login interactively. Of course the set of tasks is bounded and known, preventing malicious code execution without login.

```
enterprise/extensions/cron/cron.php
```

Visual console external view

This generates a page with a full screen visual console view without requiring a login, although for authentication, a hash generated for each view published this way is required.

```
operation/visual_console/public_console.php
```

Console Networkmap Popup detail

A popup form that shows the detail view of an agent that has an item in the Networkmap Console. To authenticate, it uses the session of the authenticated user in PFMS Web console.

```
enterprise/operation/agentes/networkmap_enterprise.popup.php
```

Popup module chart

A pop-up form that displays a detailed graph of a module and also allows configuring multiple display parameters. For authentication, it uses the data of the user logged into PFMS Web console.

```
operation/agentes/stat_win.php
```

Static graphs

Static graphs are image files generated by PHP scripts, the data to be shown, if they are extensive, are saved serialized in a special file that the script processes, this file has a lifetime to avoid malicious accesses and DoS attacks. For the execution of this script, it is not necessary to be authenticated in Pandora FMS.

```
include/graphs/fgraph.php
```

Reports

CSV reports

The script generates a text file containing the CSV data, for authentication it uses the data of the logged in user.

```
enterprise/operation/reporting/reporting_viewer_csv.php
```

PDF report

The script generates a text file containing the PDF data, for authentication it uses the data of the logged in user.

```
enterprise/operation/reporting/reporting_viewer_pdf.php
```

Events

Popup sound events

Popup window that periodically checks whether there is any event to show it in an audible and visual way. Authentication is by means of the user's data that logged into PFMS Web console.

```
operation/events/sound_events.php
```

CSV events

The script generates a text file containing CSV data, for authentication it uses the data of the logged in user.

```
operation/events/export_csv.php
```

RSS events

The script generates a text file containing the CSV data, for authentication it uses the data of the

logged in user with a hash as a parameter.

```
operation/events/events_rss.php
```

Basic functions to obtain agent, module and group statuses

Criteria for states and coding in the database

Agent status despcription:

- Critical (red color): 1 or more modules in critical state.
- Warning (yellow color): 1 or more modules in warning status and none in critical status.
- Unknown (gray color): 1 or more modules in unknown status and none in critical and warning status.
- Normal (green color): all modules are in normal status.

Internal code of the state in the DB:

- Critical: 1
- Warning: 2
- Unknown: 3
- Normal: 0

Agents

State Functions

These functions return the number of modules and alerts triggered from an agent by applying an optional filter.

All functions have the filter option, which was added to make the function more flexible. The filter content is added to the end of the SQL query in all functions. With the filter you may add specific SQL clauses to create filters using the tables: status_tag, tag and module_tag.

```
agents_monitor_critical($id_agent, $filter='')
```

It returns the number of modules in critical status for the agent.

```
agents_monitor_warning ($id_agent, $filter ='')
```

It returns the number of modules in warning status for the agent.

```
agents_monitor_unknown ($id_agent, $filter ='')
```

It returns the number of modules in unknown status for the agent.

```
agents_monitor_ok ($id_agent, $filter ='')
```

It returns the number of modules in normal state for the agent.

```
agents_get_alerts_fired ($id_agent, $filter ='')
```

It returns the number of alerts triggered for the agent.

Auxiliary functions

These functions perform agent-related tasks for some views:

```
agents_tree_view_alert_img ($alert_fired)
```

It returns the path to the alert image used in the Tree View.

```
agetns_tree_view_status_img ($critical, $warning, $unknown)
```

It returns the path of the image for the agent status used in the Tree View.

Groups

These functions return the agent and module statistics for the agent groups defined in Pandora FMS.

The server and console functions must use the same SQL queries to ensure that the result is calculated in the same way.

Server functions

```
pandora_group_statistics
```

This function calculates group statistics if the Use realtime statistics parameter is disabled.

FConsole functions

Groups

The console functions calculate statistics based on a matrix of agent groups.

These functions do not use the disabled agents or modules:

```
groups_agent_unknown ($group_array)
```

It returns the number of agents in unknown state for the given groups.

```
groups_agent_ok ($group_array)
```

It returns the number of agents in normal state for the given groups.

```
groups_agent_critical ($group_array)
```

It returns the number of agents in critical state for the given groups.

```
groups_agent_warning ($group_array)
```

It returns the number of agents in warning status for the given groups.

Modules

These functions calculate statistics for modules. They do not use the disabled modules or agents:

```
groups_monitor_not_init ($group_array)
```

It returns the number of modules with uninitialized status for the given groups.

```
groups_monitor_ok ($group_array)
```

It returns the number of modules with normal status for the given groups.

```
groups_monitor_critical ($group_array)
```

It returns the number of modules with critical status for the given groups.

```
groups_monitor_warning ($group_array)
```

It returns the number of modules with warning status for the given groups.

```
groups_monitor_unknown ($group_array)
```

It returns the number of modules with unknown status for the given groups.

```
groups_monitor_alerts ($group_array)
```

It returns the number of modules with alerts for the given groups.

```
groups_monitor_fired_alerts ($group_array)
```

It returns the number of modules with triggered alerts for the given groups.

Modules

These functions return statistics based on the module name. They do not take into account disabled agents and modules.

```
modules_agents_unknown ($module_name)
```

It returns the number of agents in unknown status that have a module with the given name.

```
modules_agents_ok ($module_name)
```

It returns the number of agents in normal state that have a module with the given name.

```
modules_agents_critical ($module_name)
```

It returns the number of agents in critical status that have a module with the given name.

```
modules_agents_warning ($module_name)
```

It returns the number of agents in warning status that have a module with the given name.

These functions return statistics using the module group as a filter. It does not take into account disabled agents or modules:

```
modules_group_agent_unknown ($module_group)
```

It returns the number of agents with unknown status that have modules belonging to the given module group.

```
modules_group_agent_ok ($module_group)
```

It returns the number of agents with normal status that have modules belonging to the given module group.

```
modules_group_agent_critical ($module_group)
```

It returns the number of agents with critical status that have modules belonging to the given module group.

```
modules_group_agent_warning ($module_group)
```

It returns the number of agents with warning status that have modules belonging to the given module group.

Policies

These functions return the number of agents for each given state and policy. They do not use the disabled agents and modules to calculate the result.

```
policies_agents_critical ($id_policy)
```

It returns the number of agents with critical status that belong to a given policy.

```
policies_agents_ok ($id_policy)
```

It returns the number of agents with normal status that belong to a given policy.

```
policies_agents_unknown ($id_policy)
```

It returns the number of agents with unknown status that belong to a given policy.

```
policies_agents_warning ($id_policy)
```

It returns the number of agents with warning status that belong to a given policy.

OS

These functions calculate the statistics for the agents based on the Operating Systems they belong to. They do not use disabled agents or modules.

```
os_agents_critical ($id_os)
```

It returns the number of agents in critical state that belong to the given Operating System.

```
os_agents_ok($id_os)
```

It returns the number of agents in normal state that belong to the given Operating System.

```
os_agents_warning ($id_os)
```

It returns the number of agents in warning status that belong to the given Operating System.

```
os_agents_unknown ($id_os)
```

It returns the number of agents in unknown state that belong to the given Operating System.

Development and expansion

Most of the extensions have been described as independent annexes, specialized for server plugin creation, Unix agent plugins, and Web Console extensions. This chapter describes how to collaborate in Pandora FMS and how to compile the Windows agent from the sources.

Collaborate with Pandora FMS project

This project is maintained by volunteer developers with their contributions. New developers, documentation writers, or people who want to help are always welcome. A good way to get started is by subscribing to our mailing lists and/or forum.

Bugs

Inform about possible errors helps us improve Pandora FMS. Please, before forwarding a bug report check our bug database.

Mailing lists

Mailing lists are a great way to keep up to date by email. We have a public mailing list for users and announcements (with low traffic) and a development mailing list for technical discussions and (sometimes daily) development notifications via our GIT (Code Version Control System) automatic notification system.

Compiling the Windows agent from code

Get the latest version of the code

To get the latest version of the code, it is essential to download the sources from the code repository GitHub, where the official Pandora FMS repository is published.

Pandora FMS API

There is a Pandora FMS external API to be able to link third party applications with Pandora FMS, both to obtain information from Pandora FMS and to introduce information inside Pandora FMS. All this documentation is in the appendix of Pandora FMS External API.

Format of XML data files

Knowing Pandora FMS data XML format can help to improve agent plugins, create custom agents or simply send custom XML files to Pandora FMS data server.

Like any XML document, the data file must begin with an XML declaration:

```
<?xml version='1.0' encoding='UTF-8'?>
```

Next comes the agent_data element that defines the agent that sends the data. It supports the following attributes:

- *description*: Agent description.
- *group*: Name of the group to which the agent belongs (it must exist in Pandora FMS database). If it is left empty and there is no default group configured in the server, the agent will not be created.
- *os_name*: Name of the operating system where the agent runs (it must exist in Pandora FMS database).
- *os_version*: Free string describing the operating system version.
- *interval*: Agent interval (in seconds).
- *version*: String with the agent's version.
- *timestamp*: Timestamp that indicates when the XML (`YYYY/MM/DD HH:MM:SS`).
- *agent_name*: Name of agent.
- *timezone_offset*: Offset that is added to the timestamp (in hours). Useful when working with UTC.
- *address*: Agent IP address (or FQDN).
- *parent_agent_name*: Name of agent's parent.
- *agent_alias*: Agent's alias.
- *agent_mode*: Agent's working mode (0: Normal mode, 1: Learning mode, 2: Autodisable mode) .
- *secondary_groups*: Secondary groups added to the agent.
- *custom_id*: Custom agent ID.
- *url_address*: URL of access to the agent.

A module element is then required for each module, and the following elements can be nested to define the module:

- name: Module name.
- description: Module description.
- tags: Tags associated with the module.
- type: Module type (it must exist in Pandora FMS database).
- data: Module data.
- max: Maximum value of the module.
- min: Minimum module value.
- post_process: Post-processing value.
- module_interval: Module interval (interval in seconds / agent interval).
- min_critical: Minimum value for the critical state.
- max_critical: Maximum value for the critical state.
- min_warning: Minimum value for the alert state.
- max_warning: Maximum value for the alert state.
- disabled: Disable (0) or enable the module. Disabled modules are not processed.
- min_ff_event: Minimum FF threshold.
- status: Module status (NORMAL, WARNING or CRITICAL). The critical and warning status limits are ignored if the status is specified.
- datalist: It sends the module data in datalist format (one database entry for each of the values received). [0/1].
- unit: Module unit. Supports the `_timeticks_` macro to transform a data in *timeticks* format to dd/hh/mm/ss.
- timestamp: It sets a timestamp on the data received from the module.
- module_group: Group of modules to which the module will be added.
- custom_id: Custom ID of the module.
- str_warning: Warning threshold for string modules.
- str_critical: Critical threshold for string modules.
- critical_instructions: Module Critical instructions.
- warning_instructions: Module Warning instructions.
- unknown_instructions: Module Unknown instructions.
- critical_inverse: It activates the Inverse interval at the critical threshold of the module. [0/1].
- warning_inverse: It activates the Inverse interval on the module warning threshold [0/1].
- quiet: It activates the Quiet mode of the module [0/1].
- module_ff_interval: It specifies a value of FF Interval of the module.
- alert_template: It associates an alert template to the module.
- crontab: It specifies a crontab in the module.
- min_ff_event_normal: FF threshold value on change of state to NORMAL.
- min_ff_event_warning: FF threshold value on state change to WARNING.
- min_ff_event_critical: FF threshold value on state change to CRITICAL.
- ff_timeout: FlipFlop timeout value.
- each_ff: It enables the “Change each status” option.
- module_parent: Name of the module in the same agent that will be the parent of this module.
- ff_type: It activates the Keep counters of the FF threshold [0/1].
- min_warning_forced: It forces min_warning to the new specified value even if the module exists, takes precedence over min_warning.
- max_warning_forced: It forces max_warning to the new specified value even if the module exists, takes precedence over max_warning.
- min_critical_forced: It forces min_critical to the new value indicated even if the module exists, takes precedence over min_critical.
- max_critical_forced: It forces max_critical to the new specified value even if the module exists, it

takes precedence over max_critical.
- str_warning_forced: It forces str_warning to the new specified value even if the module exists, takes precedence over str_warning.
- str_critical_forced: It forces str_critical to the new specified value even if the module exists, takes precedence over str_critical.

These tokens will only work for dataserver plugins.

Any other element will be saved as extended information of the module in Pandora FMS database. A module must have at least one element name, type and data, for example:

```
  <module>
    <name>CPU</name>
    <description>CPU usage percentage</description>
    <type>generic_data</type>
    <data>21</data>
  </module>
```

There can be any number of elements in an XML data file before closing the agent_data tag.

There is a special case of multi-item XML, based on a list of data. This applies only to string data. The XML would look something like this:

```
<module>
 <type>async_string</type>
 <datalist>
   <data><value><![CDATA[xxxxx]]></value></data>
   <data><value><![CDATA[yyyyy]]></value></data>
   <data><value><![CDATA[zzzzz]]></value></data>
 </datalist>
</module>
```

A timestamp can be specified for each value:

```
<module>
 <type>async_string</type>
 <datalist>
   <data>
     <value><![CDATA[xxxxx]]></value>
     <timestamp>1970-01-01 00:00:00</timestamp>
   </data>
   <data>
     <value><![CDATA[yyyyy]]></value>
     <timestamp>1970-01-01 00:00:01</timestamp>
   </data>
   <data>
     <value><![CDATA[zzzzz]]></value>
     <timestamp>1970-01-01 00:00:02</timestamp>
```

```
  </data>
</datalist>
</module>
```

Go back to Pandora FMS documentation index