.Disco development

.Disco development

Packages ".disco"

Discovery allows you to load both official Pandora FMS plugins and custom ones.

To load custom plugins it is necessary to generate a `.disco` package with everything necessary so that both the console and the Pandora FMS server are capable of:

- Show the configuration interface for new discovery tasks.
- Execute the discovery tasks configured in the environment.

A `.disco` package is a zip file with the extension .disco which contains at least one file called discovery_definition.ini.

Optionally, a `.disco` package may contain a file called logo.png which will be the plugin logo in the Pandora FMS console. If no logo file is added, the console automatically uses a default logo.

Finally, typically, a `.disco` file will contain all the scripts, executables, and libraries needed by the server and the console. Although it is not mandatory, it is common and recommended, to avoid additional installation requirements on systems where you want to configure and execute the tasks that use the plugin.

Therefore, and in short, a `.disco` file will contain:

- `discovery_definition.ini`.
- `logo.png` (optional).
- Scripts, executables and libraries.

File "discovery_definition.ini"

The discovery_definition.ini file is the most important within a `.disco` file, since it is the one that contains the entire plugin definition.

It contains both the parameters that will be displayed by the console to be filled in a task definition form, as well as the executions that the Pandora FMS server will have to perform for the plugin tasks.

In order to facilitate its definition and processing in the console, the format used in a `discovery_definition.ini` file is the INI format of PHP.

A `discovery_definition.ini` file is basically made up of 3 blocks:

- `discovery_extension_definition`.
- `config_steps` (optional).
- `tempfile_confs` (optional).

Common Concepts

During the following explanations it will be common to use some terms. This section explains those terms.

A macro is a unique text key used to store information (values, paths to files, …). When a macro is used (for example during a run) you want to use the information it contains instead.

A valid macro is a text key that starts and ends with underscores _ and can only contain letters (A-Z and a-z) and numbers (0-9) in between.

When we talk about STRING we refer to alphanumeric text strings.

When we talk about NUMBER we will refer exclusively to numbers.

When we talk about BOOL we refer to values true / false, 1 / 0, yes / no.

When we talk about VALUE we refer to any value (STRING, NUMBER or BOOL). It will normally be used in comma-separated lists to represent values of the same type.

When the letters N or M are used in capital letters, we refer to a positive integer. It will normally be used in conjunction with other elements such as VALUE to indicate that it can be repeated more than once (For example, VALUE_N).

Server Macros

Much of the configuration of executions for the server is based on the use of macros. Mainly, these will be defined by the user, but there are some specific ones of the Pandora FMS server that can be useful:

- `__taskMD5__` → Unique MD5 of the task, generated from the task ID and the application short name. It is useful for generating unique elements during executions (such as files).
- `__taskInterval__` → Task execution interval (seconds).
- `__taskGroup__` → Task group (Text).
- `__taskGroupID__` → Task group (ID).
- `__temp__` → Temporary directory of the server configured in the `pandora_server.conf` (temporary).
- `__incomingDir__` → Pandora server incoming directory configured in `pandora_server.conf` (incomingdir).
- `__consoleAPIURL__` → API URL configured in `pandora_server.conf` (console_api_url).
- `__consoleAPIPass__` → API pass configured in `pandora_server.conf` (console_api_pass).

- `__consoleUser__` → Console user configured in `pandora_server.conf` (console_user).
- `__consolePass__` → Console password configured in `pandora_server.conf` (console_pass).

These macros are in no case used by the Pandora FMS console.

discovery_extension_definition block

This block defines the main configuration of the plugin and has the following parameters:

- short_name:
  - Mandatory.
  - Defines the short name of the plugin.
  - The short name must be unique.
  - Short names beginning with the prefix “pandorafms.” They are used by the official Pandora FMS plugins.
  - Short names can only contain letters (`A-Z` and `a-z`), numbers (`0-9`), periods (`.`), hyphens (`-`), and underscores (_).
- section:
  - Mandatory.
  - Defines the section of the console where the plugin will be displayed.
  - Can be app, cloud or custom.
- name:
  - Mandatory.
  - Defines the name of the plugin that will be displayed in the console.
- version:
  - Mandatory.
  - Defines the version of the plugin that will be displayed in the console.
  - It is recommended to change the version if changes are made to a plugin, no matter how minor.
- description:
  - Optional.
  - Defines the description of the plugin that will be displayed in the console.
- execution_file:
  - Optional.
  - Indicates the relative paths (within the `.disco` file) to scripts and executables used by the plugin.
  - All the indicated files will be given execute permission.
  - This parameter is an array whose keys will be valid macros that will be used to indicate the use of the plugin files.
  - Being an array this parameter can be indicated several times using different keys.
  - For example:

```
execution_file[_exec1_] = "script1.py"
execution_file[_exec2_] = "other/script2.py"
```

- exec:
  - Mandatory.

    - Defines the format of the executions that the Pandora FMS server must perform for the plugin tasks.
    - This parameter is an array whose keys will be positional, that is, they can be indicated with numbers or not indicated.
    - Being an array this parameter can be indicated several times using different keys.
    - It will be common to use macros in the defined executions both to indicate the paths to the scripts or executables used and to indicate the parameters of said scripts or executables.
    - The Pandora FMS server will launch each of the defined executions in order and will store the result of all of them to produce the output of the plugin in the task and obtain its status.
    - For example:

```
exec[] = "'_exec1_' -p '_param1_'"
exec[] = "'_exec2_' -p '_param2_'"
```

- passencrypt_script:
    - Optional.
    - Defines the relative path (within the `.disco` file) to a script used to encrypt password-type fields from the console.
    - The indicated file will be given execute permission.
- passencrypt_exec:
    - Optional.
    - Defines the format of the execution of the script to encrypt passwords from the console.
    - The expected result of said execution will be a text that corresponds to the encrypted password.
    - Only supports the use of the _passencrypt_script_ and _password_ macros in its definition.
    - The _passencrypt_script_ macro is replaced by the path to the file defined in passencrypt_script.
    - The _password_ macro is replaced by the value of the password type field that must be encrypted at any time.
    - For example:

```
passencrypt_exec = "'_passencrypt_script_' --encrypt '_password_'"
```

- passdecrypt_script:
    - Optional.
    - Defines the relative path (within the `.disco` file) to a script used to decrypt password-type fields from the console.
    - The indicated file will be given execute permission.
- passdecrypt_exec:
    - Optional.
    - Defines the format of the execution of the script to decrypt passwords from the console.
    - The expected result of said execution will be a text that corresponds to the decrypted password.
    - Only supports the use of the _passdecrypt_script_ and _password_ macros in its definition.
    - The _passdecrypt_script_ macro is replaced by the path to the file defined in passdecrypt_script.
    - The _password_ macro is replaced by the value of the password type field that must be decrypted at any time.
    - For example:

```
passencrypt_exec = "'_passdecrypt_script_' --decrypt '_password_'"
```

- default_value:

- Optional.
- Defines the default values for the various macros used during executions, ie the default values for the data stored for each *Discovery* task created for this plugin.
- This parameter is an array whose keys will be valid macros.
- Being an array this parameter can be indicated several times using different keys.
- Depending on the type of field used in the console forms (see below) the default values may be:
  1. string: *STRING* .
  2. number: *NUMBER* .
  3. password: *STRING* .
  4. textarea: *STRING* .
  5. checkbox: *BOOL* .
  6. select: *VALUE_N* .
  7. multiselect: *[VALUE_1,VALUE_N]* .
  8. tree: *[VALUE_1,VALUE_N]* .
- It is recommended to include a default_value for each of the macros that may be in the task configuration forms.
- For example:

```
default_value[_param1_] = "get_main_info"
default_value[_param2_] = ""
default_value[_param3_] = true
default_value[_param4_] = 0
default_value[_param5_] = "[A,B,C]"
default_value[_param6_] = "[]"
```

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[discovery_extension_definition]

; Mandatory
; Defines discovery application short name
; Short name must be unique
; Short names with "pandorafms." Prefixes are used by Pandora FMS for official
applications

short_name = DISCOVERY_APPLICATION_UNIQUE_NAME

; Mandatory
; Defines the section where application will be shown in console
; Possible values:
; apps
; clouds
; custom

section = app

; Mandatory
; Defines discovery application name, shown in console

name = DISCOVERY_APPLICATION_NAME
```

```
; Mandatory
; Defines discovery application version, shown in console

version = VERSION

; Optional
; Defines discovery application description, shown in console

description = DESCRIPTION

; Optional
; Defines execution files inside .disk
; Several execution files can be defined

execution_file[_EXEC_MACRO_N_] = SCRIPT.pl

; Mandatory
; Defines execution for discovery server
; Several executions can be defined
; At least 1 is required for server execution

exec[] = SERVER_EXECUTION

; Optional
; Define password encrypt script

passencrypt_script = PASS_ENCRYPT_SCRIPT.pl

; Optional
; Defines password encrypt script execution format
; _passencrypt_script_ is replaced with the passencrypt_script file path
; _password_ is replaced with the string (password) to encrypt

passencrypt_exec = EXECUTION

; Optional
; Define password decrypt script

passdecrypt_script = PASS_DECRYPT_SCRIPT.pl

; Optional
; Defines password encrypt script execution format
; _passdecrypt_script_ is replaced with the passdecrypt_script file path
; _password_ is replaced with the string (password) to encrypt

passdecrypt_exec = EXECUTION

; Optional
; Defines the default values for the fields when a new task is created
; By default all values are empty or not selected
; Several default values can be defined
; Possible values depending on field type:
```

```
; string - STRING
; number - NUMBER
; password-STRING
; textarea-STRING
; checkbox-BOOL
; select-VALUE_N
; multiselect - [VALUE_1,VALUE_N]
; tree - [VALUE_1,VALUE_N]

default_value[_FIELD_MACRO_N_] = DEFAULT_VALUE
```

config_steps block

This block defines the configuration steps for the plugin tasks and has the following parameters:

These rules apply to all parameters of this block:

- The parameters are arrays whose keys will be positional, that is, they can be indicated with numbers or not indicated. It is recommended to indicate the keys.
- As they are arrays the parameters can be indicated several times using different keys.
- All parameters that share a key refer to the same element. That is, they assign different values (depending on the parameter) to the same element.

- name:
  - Mandatory.
  - Defines the names of the different configuration steps in the tasks.
- script_data_fields:
  - Required if a custom_fields is not defined for the same key.
  - Optional if a custom_fields is defined for the same key.
  - Defines the format of the executions that the Pandora FMS console must carry out for the forms that are generated dynamically. For example, if you are going to monitor a virtualization environment and you want to get a list of virtual machines to select the ones you want to monitor.
  - The keys used must exist in the name array of this block.
  - It will be common to use macros in the executions defined both to indicate the paths to the scripts or executables used and to indicate the parameters of said scripts or executables.
  - The result of the executions indicated here must be a JSON with the following format. The JSON keys correspond to the parameters of the CUSTOM_FIELDS block (see below):

```
        [
            {
                "macro": "_FIELD_MACRO_N_",
                "mandatory_field": "BOOL",
                "name": "FIELD_NAME",
                "tip": "FIELD_TIP",
                "type": "FIELD_TYPE",
```

```
                "placeholder": "PLACEHOLDER",
                "show_on_true": "_FIELD_MACRO_N_",
                "encrypt_on_true": "_FIELD_MACRO_N_",
                "select_data": {
                    "VALUE_1": "TEXT_1",
                    "VALUE_N": "TEXT_N"
                },
                "tree_data": [
                    {
                        "name": "TEXT_1",
                        "selectable": "BOOL_1",
                        "macro": "_FIELD_MACRO_N_",
                        "value": "VALUE_1",
                        "children": [
                            {
                                "name": "TEXT_1_1",
                                "selectable": "BOOL_1_1",
                                "macro": "_FIELD_MACRO_N_",
                                "value": "VALUE_1_1",
                                "children": []
                            },
                            {
                                "name": "TEXT_1_N",
                                "selectable": "BOOL_1_N",
                                "macro": "_FIELD_MACRO_N_",
                                "value": "VALUE_1_N",
                                "children": []
                            }
                        ]
                    },
                    {
                        "name": "TEXT_N",
                        "selectable": "BOOL_N",
                        "macro": "_FIELD_MACRO_N_",
                        "value": "VALUE_N",
                        "children": []
                    }
                ]
            }
        ]
```

- custom_fields:
  - Required if a script_data_fields is not defined for the same key.
  - Optional if a script_data_fields is defined for the same key.
  - Defines the configuration blocks of the same INI file from which the form fields will be obtained when creating tasks in the Pandora FMS console.
  - The keys used must exist in the name array of this block.
- fields_columns:
  - Optional.
  - Defines the number of columns in which the fields of the task forms will be distributed in each configuration step.
  - The keys used must exist in the name array of this block.

- Can be assigned the value 1 or 2.
- Its default value is 2.

For example, this would be a way to define multiple configuration steps:

```
[config_steps]

name[1] = First step
script_data_fields[1] = "'_exec2_' -p '_param1_' --get_fields"

name[2] = Mid step
custom_fields[2] = custom_fields_1

name[3] = Last step
script_data_fields[3] = "'_exec2_' -p '_param2_' --get_fields"
custom_fields[3] = custom_fields_2
```

If script_data_fields and custom_fields are indicated for the same configuration step, the form will first show the fields obtained by script_data_fields and then those defined in custom_fields.

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[config_steps]

; Following parameters can be setup for each configuration step
; Several steps can be defined using a different key (N)

; Mandatory
; Defines configuration step name

name[N] = STEP_NAME

; Mandatory if not custom_fields defined
; Defines configuration fields retrieved to console by a command execution
; execution_file scripts can be used to retrieve data
; Command execution output must be JSON as follows
; [
; {
; "macro": "_FIELD_MACRO_N_",
; "mandatory_field": "BOOL",
; "name": "FIELD_NAME",
; "tip": "FIELD_TIP",
; "type": "FIELD_TYPE",
; "placeholder": "PLACEHOLDER",
; "show_on_true": "_FIELD_MACRO_N_",
; "encrypt_on_true": "_FIELD_MACRO_N_",
; "select_data": {
; "VALUE_1": "TEXT_1",
; "VALUE_N": "TEXT_N"
; },
```

```
; "tree_data": [
; {
; "name": "TEXT_1",
; "selectable": "BOOL_1",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1",
; "children": [
; {
; "name": "TEXT_1_1",
;"selectable": "BOOL_1_1",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1_1",
; "children": []
; },
; {
; "name": "TEXT_1_N",
; "selectable": "BOOL_1_N",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1_N",
; "children": []
; }
; ]
; },
; {
; "name": "TEXT_N",
; "selectable": "BOOL_N",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_N",
; "children": []
; }
; ]
; }
; ]

script_data_fields[N] = CONSOLE_EXECUTION_FIELDS

; Mandatory if not script_data_fields defined
; Defines custom configuration fields

custom_fields[N] = CUSTOM_FIELDS_N

; Optional
; Defines the number of fields columns for the configuration steps (1 or 2)

fields_columns[N] = M
```

tempfile_confs block

This block defines the format for the temporary configuration files used by the plugin and has the following parameters:

- file:
  - Mandatory.
  - Defines the contents for temporary configuration files that can be used during the executions of the Pandora FMS server and console.
  - This parameter is an array whose keys will be valid macros that will be used to indicate the use of temporary files in the plugin.
  - Being an array this parameter can be indicated several times using different keys.
  - When one of the keys of this array is indicated during an execution, a temporary file is generated whose content is the one indicated in this array and the value of the macro is replaced by the location of said file temporary. Once the execution is finished, the file is deleted.
  - Within the content of the temporary configuration files, it is possible to indicate other macros, so that they are replaced by the corresponding value.
  - For example:

```
file[_tempConf_] = "server _param1_
user _param2_
password _param3_
log __temp__/__taskMD5__.log"
```

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[tempfile_confs]

; Mandatory
; Defines the content for the temporary file
; File will be used where temporary file macro is specified during executions
; File content replaces fields macros with their values

file[_TEMP_FILE_MACRO_N_] = _FIELD_MACRO_N_,_FIELD_MACRO_M_
```

CUSTOM_FIELDS blocks

This type of block can have any name you want, as long as it has been assigned as a value within the config_steps block for its custom_fields parameter. They define the fields for the forms in the plugin configuration steps and have the following parameters:

These rules apply for all parameters in this block:

- The parameters are arrays whose keys will be positional, that is, they can be indicated with numbers or not indicated. It is recommended to indicate the keys.
- Being arrays the parameters can be indicated several times using different keys.
- All parameters that share a key refer to the same element. That is, they assign different values (depending on the parameter) to the same element.

- macro:
  - Mandatory.

    - Defines the macros in which the task configuration values will be stored.
- mandatory_field:
    - Optional.
    - Defines by means of BOOL type values if the configuration field of the form must have a mandatory value or not.
    - The keys used must exist in the macro array of this block.
    - By default all fields are required.
    - Even though they are mandatory, the multiselect and tree type fields admit that elements cannot be selected.
- name:
    - Mandatory.
    - Defines the names of the form configuration fields.
    - The keys used must exist in the macro array of this block.
- tip:
    - Optional.
    - Defines the help of the form configuration fields.
    - The keys used must exist in the macro array of this block.
- type:
    - Mandatory.
    - Defines the types of the form configuration fields.
    - The keys used must exist in the macro array of this block.
    - Its possible values are:
        - string: The field will be a text box. It will admit values of type STRING.
        - number: The field will be a number entry box.Shall accept values of type NUMBER.
        - password: The field will be a text box whose value will be hidden. It will admit values of type STRING.
        - textarea: The field will be a wide text box. It will admit values of type STRING.
        - checkbox: The field will admit values of type BOOL
        - select: The field will be a dropdown in which to select a single value. Shall accept values of type VALUE.
        - multiselect: The field will be a multiple selection box from several options. Shall accept values of type VALUE.
        - tree: The field will be a tree of different levels whose elements can be selected or not to send their values. Shall accept values of type VALUE.
- placeholder:
    - Optional if the associated type is string or textarea.
    - Defines the texts that will be displayed in the text boxes as examples.
    - The keys used must exist in the macro array of this block.
- show_on_true:
    - Optional.
    - Defines macros of other checkbox fields on the form as values, so that when those macros have a value of true on the form, the desired field is displayed.
    - The keys used must exist in the macro array of this block.
- encrypt_on_true:
    - Optional if the associated type is password.
    - Defines macros of other fields of type checkbox in the form as values, so that when these macros have a value true in the form, the desired field is encrypted when saving the configuration.
    - The encryption/decryption will be done based on the configuration in the discovery_extension_definition block.
    - The keys used must exist in the macro array of this block.
- select_data:

    - Required if the associated type is select or multiselect.
    - Defines the source of the values for the dropdowns.
    - The selectors can be generated with data from Pandora FMS or customized. To do this, it admits the following values:
        - The name of other configuration blocks in the same INI file from which the elements of type select or multiselect will be obtained.
            - agent_groups: Use agent groups as data.
            - agents: Use agents as data.
            - module_groups: Use module groups as data.
            - modules: Use modules as data.
            - module_types: Use the module types as data.
            - tags: Use the tags as data.
            - status: Use statuses as data.
            - alert_templates: Use alert templates as data.
            - alert_actions: Use alert actions as data.
            - interval: Use the time selector as data.
            - credentials.custom: Use the custom credentials selector as data.
            - credentials.aws: Uses the AWS credentials selector as data.
            - credentials.azure: Uses the Azure credentials picker as data.
            - credentials.gcp: Uses the Google Cloud credentials selector as data.
            - credentials.sap: Uses the SAP credentials selector as data.
            - credentials.snmp: Uses the SNMP credentials selector as data.
            - credentials.wmi: Uses the WMI credentials selector as data.
            - os: Use the OS as data.
    - The keys used must exist in the macro array of this block.
    - The values of the macro if the associated type is multiselect will be a JSON of type array with the different values selected. For example:

```
[1,5,12,23]
```

- tree_data:
    - Required if the associated type is tree.
    - Defines the configuration blocks of the same INI file from which the tree type elements will be obtained.
    - The keys used must exist in the macro array of this block.
    - The values of the macro will be a JSON of type array with the different values selected. For example:

```
["elementA","elementF","elementK"]
```

For example, this would be a way to define several configuration fields:

```
[custom_fields_1]

macro[1] = _param1_
name[1] = User
type[1] = string

macro[2] = _param2_
name[2] = Password
type[2] = password
encrypt_on_true[2] = _param3_
```

```
macro[3] = _param3_
name[3] = Encrypt password
type[3] = checkbox

macro[4] = _param4_
name[4] = Max threads
type[4] = number
mandatory_field[4] = false

macro[5] = _param5_
name[5] = Agents group
tip[5] = Agents are generated in this group
type[5] = select
select_data[5] = agent_groups

macro[6] = _param6_
name[6] = Mode
type[6] = select
select_data[6] = custom_select_1

macro[7] = _param7_
name[7] = Add extra options
type[7] = checkbox

macro[8] = _param8_
name[8] = Extra elements
type[8] = tree
tree_data[8] = custom_tree_1
show_on_true[8] = _param7_

macro[9] = _param9_
name[9] = Extra options
type[9] = textarea
placeholder[9] = "Add extra options here"
show_on_true[9] = _param7_
```

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[CUSTOM_FIELDS_N]

; Mandatory
; Defines configuration field unique macro
; Macros can be used for script executions, using their value in place

macro[N] = _FIELD_MACRO_N_

; Optional
; Defines if configuration field is mandatory or not
; By default all configuration fields are mandatory
```

```
mandatory_field[N] = BOOL

; Mandatory
; Defines configuration field name to be displayed in console
; Macro will be used as name if this field is empty

name[N] = FIELD_NAME

; Optional
; Define a tip for the field, to be displayed in console

tip[N] = FIELD_TIP

; Mandatory
; Defines the field type to be displayed in console
; Possible values:
; string
; number
; password
; textarea
; check box
; select
; multi select
; tree

type[N] = FIELD_TYPE

; Optional if type is string or textarea
; Defines a placeholder for the field, to be displayed in console

placeholder[N] = PLACEHOLDER

; Optional
; Field is shown in console only if assigned field macro exists, is checkbox and
is true

show_on_true[N] = _FIELD_MACRO_N_

; Optional if type is password
; Field value is encrypted when stored into database if assigned field macro
exists, is checkbox and is true
; Password encrypt and decrypt depends on scripts uploaded to the discovery
application

encrypt_on_true[N] = _FIELD_MACRO_N_

; Mandatory if type is select or multiselect
; Defines select data to be displayed in console
; Possible values:
; agent_groups - Uses agent groups names
; agents - Uses agents names
; module_groups - Uses module groups
```

```
; modules - Uses modules names
; module_types - Uses module types names
; tags - Uses module tags
; status - Uses module status names
; alert_templates - Uses alert templates names
; alert_actions - Uses alert actions names
; interval - Uses time interval selector
; credentials.custom - Uses pandora custom credentials selector
; credentials.aws - Uses pandora AWS credentials selector
; credentials.azure - Uses pandora Microsoft Azure credentials selector
; credentials.gcp - Uses pandora Google Cloud Platform credentials selector
; credentials.sap - Uses pandora SAP credentials selector
; credentials.snmp - Uses pandora SNMP credentials selector
; credentials.wmi - Uses pandora WMI credentials selector
; os - Uses pandora OS names
; CUSTOM_SELECT_N - Uses custom selector values
; multiselect fields value is a comma separated list of selected values

select_data[N] = FIELD_DATA

; Mandatory if type is tree
; Defines tree data to be displayed in console
; tree fields value is a comma separated list of selected values

tree_data[N] = CUSTOM_TREE_DATA_N
```

CUSTOM_SELECT Blocks

This type of block can have any name you like, as long as it has been assigned as a value within a CUSTOM_FIELDS block for its select_data parameter. They define the options of a drop-down of the plugin configuration form and it has the following parameters:

- option:
  - Mandatory.
  - Defines the options of a custom selector.
  - This parameter is an array whose keys are the values of the custom selector options, and its values are the texts displayed for each option in the custom selector.
  - Being an array this parameter can be indicated several times using different keys.

For example:

```
[custom_select_1]

option[v1] = Value 1
option[v2] = Value 2
option[v3] = Value 3
option[v4] = Value 4
option[v5] = Value 5
```

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[CUSTOM_SELECT_N]

; Mandatory
; Defines the value-text pair for the select or multiselect
; Several value-text pairs can be defined

option[VALUE_N] = TEXT_N
```

CUSTOM_TREE_DATA Blocks

This type of block can have any name you like, as long as it has been assigned as a value within a CUSTOM_FIELDS block for its tree_data parameter or within another CUSTOM_TREE_DATA block for its parameter children. They define the elements of a tree of the plugin configuration form and have the following parameters:

These rules apply forto all parameters of this block:

- The parameters are arrays whose keys will be positional, that is, they can be indicated with numbers or not indicated. It is recommended to indicate the keys.
- Being arrays the parameters can be indicated several times using different keys.
- All parameters that share a key refer to the same element. That is, they assign different values (depending on the parameter) to the same element.

- name:
  - Mandatory.
  - Defines the names of the different elements of this level of the tree.
- selectable:
  - Optional.
  - Defines if the elements of this level of the tree are selectable or not.
  - By default all tree elements are selectable.
  - The keys used must exist in the name array of this block.
- macro:
  - Optional if the associated selectable is true
  - Defines the macros for which the values of the elements of this level of the tree that are selected in the form will be stored.
  - Values must be valid macros.
  - The same macro can be defined for different elements of the same tree.
  - Macros cannot be the same as those for other elements outside the tree.
  - If no macro is defined and the associated selectable is true, the value of the tree element will be stored in the macro defined for the tree.
  - The keys used must exist in the name array of this block.

- value:
  - Required if the associated selectable is true
  - Defines the values of the different elements of this level of the tree.
  - The keys used must exist in the name array of this block.

1. The values of the macro will be a JSON of type array with the different values selected in the tree that share the same macro. For example:

```
["element1A","element1F","element1K"]
```

- children:
  - Optional.
  - Defines the names of other configuration blocks in the same INI file from which the child elements of this level of the tree will be obtained.
  - Configuration blocks from the same INI that have been used at a higher level of the tree cannot be referenced to avoid infinite loops.
  - The keys used must exist in the name array of this block.

For example, this would be a way to define several elements of a tree:

```
[custom_tree_1]

name[1] = Performance modules
selectable[1] = false
children[1] = custom_tree_1_A

name[2] = Counter modules
selectable[2] = false
children[2] = custom_tree_1_B

[custom_tree_1_A]

name[1] = Profile1
macro[1] = _param10_
value[1] = p1

name[2] = Perf2
macro[2] = _param10_
value[2] = p2

name[3] = Perf3
macro[3] = _param10_
value[3] = p3

[custom_tree_1_B]

name[1] = Counter1
macro[1] = _param11_
value[1] = c1

name[2] = Counter1
macro[2] = _param11_
```

```
value[2] = c2

name[3] = Counter1
macro[3] = _param11_
value[3] = c3
```

The following example can be used as a base for this block, since it includes all the parameters explained with comments for each case:

```
[CUSTOM_TREE_DATA_N]

; Mandatory
; Defines the name for the tree element
; Several names can be defined

name[N] = TEXT_N

; Optional
; Defines if tree element is selectable or not
; By default all tree elements are selectable
; Several selectables can be defined

selectable[N] = VALUE_N

; Optional if selectable is true
; Defines the macro where value is stored for the tree element
; Several macros can be defined
; Same macro can be defined for several tree elements inside the same tree
; Macro can't be the same as other outside the tree
; If no macro is defined, value is stored for the global tree macro
; Tree values are stored and used the same way as multiselect values do

macro[N] = FIELD_MACRO_N

; Mandatory if selectable is true
; Defines the value for the tree element
; Several values can be defined

value[N] = VALUE_N

; Optional
; Defines the children elements for the tree element
; CUSTOM_TREE_DATA_M can't be the same than in an upper level
; Several children can be defined

children[N] = CUSTOM_TREE_DATA_M
```

Selectors with Pandora FMS data

Within the fields shown in the forms, the simple and multiple selectors have the possibility of using

data from the Pandora FMS application itself.

Depending on the data that is wanted, the values that will be stored and those that will be used in the executions will vary:

- agent_groups:
  - In the selector it will use the names of the agent groups.
  - Will store your IDs as data.
  - On executions it will check if the group ID exists, and if it exists it will use its name as value. Otherwise it will go empty.
- agents:
  - In the selector it will use the aliases of the agents.
  - Will store their names as data.
  - In the runs will check if the agent exists, and if it does it will use its name as a value. Otherwise it will go empty.
- module_groups:
  - In the selector you will use the module groups.
  - Will store your IDs as data.
  - On executions it will check if the group ID exists, and if it exists it will use its name as value. Otherwise it will go empty.
- modules:
  - In the selector it will use the module names.
  - Will store their names as data.
  - In the executions it will check if a module with that name exists, and if it exists it will use its name as a value. Otherwise it will go empty.
- module_types:
  - In the selector you will use the descriptions of module types.
  - Will store their names as data.
  - In executions it will use their names as values.
- tags:
  - In the selector it will use the tags.
  - Will store your IDs as data.
  - In executions it will check if the tag ID exists, and if it exists it will use its name as value. Otherwise it will go empty.
- status:
  - In the selector it will use the names of the module states.
  - It will store your constants as data.
  - In executions it will use the constants as values:
    - 0: Normal
    - 2: Warning
    - 1: Critical
    - 3: Unknown
    - 5: Not init
- alert_templates:
  - In the selector it will use the names of the module templates.
  - Will store your IDs as data.
  - On execution it will check if the template id exists, and if it does it will use its name as the value. Otherwise it will go empty.
- alert_actions:
  - In the selector it will use the names of the alert actions.
  - Will store your IDs as data.
  - In executions it will check if the action id exists, and if it exists it will use its name as value.

Otherwise it will go empty.

- interval:
  - In the selector you will use the interval selector.
  - Will store time in seconds.
  - In the executions it will use the time in seconds as value.
- credentials.custom:
  - In the selector it will use the custom credential selector.
  - Will store your IDs as data.
  - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
  - The JSON for this type of credentials will have this format:

```
{
    "user":"USER",
    "password":"PASSWORD"
}
```

- credentials.aws:
  - In the picker you will use the AWS credential picker.
  - Will store your IDs as data.
  - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
  - The JSON for this type of credentials will have this format:

```
{
    "access_key_id":"ACCESS_KEY_ID",
    "secret_access_key":"SECRET_ACCESS_KEY"
}
```

- credentials.azure:
  - In the picker you will use the Microsoft Azure credential picker.
  - Will store your IDs as data.
  - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
  - The JSON for this type of credentials will have this format:

```
{
    "client_id":"CLIENT_ID",
    "application_secret":"APPLICATION_SECRET",
    "tenant_domain":"TENANT_DOMAIN",
    "subscription_id":"SUBSCRIPTION_ID"
}
```

- credentials.gcp:
  - In the picker you will use the Google Cloud Platform credential picker.
  - Will store your IDs as data.
  - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
  - The JSON for this type of credentials will have this format:

```
{
    "type":"service_account",
    "project_id":"PROJECT_ID",
```

```
        "private_key_id":"PRIVATE_KEY_ID",
        "private_key":"PRIVATE_KEY",
        "client_email":"CLIENT_EMAIL",
        "client_id":"CLIENT_ID",
        "auth_uri":"AUTH_URI",
        "token_uri":"TOKEN_URI",
        "auth_provider_x509_cert_url":"AUTH_PROVIDER_X509_CERT_URL",
        "client_x509_cert_url":"CLIENT_X509_CERT_URL"
    }
```

- credentials.sap:
    - In the selector you will use the SAP credential selector.
    - Will store your IDs as data.
    - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
    - The JSON for this type of credentials will have this format:

```
    {
        "user":"USER",
        "password":"PASSWORD"
    }
```

- credentials.snmp:
    - In the selector you will use the SNMP credential selector.
    - Will store your IDs as data.
    - On runs it will check if the credential id exists, and if it does it will use a base64 of a JSON with the data of the credentials as value. Otherwise it will go empty.
    - The JSON for this type of credentials will have this format:

```
    {
        "community":"COMMUNITY",
        "version":"VERSION",
        "securityLevelV3":"SECURITY_LEVEL_V3",
        "authUserV3":"USER_AUTH_V3",
        "authMethodV3":"AUTH_METHOD_V3",
        "authPassV3":"AUTH_PASS_V3",
        "privacyMethodV3":"PRIVACY_METHOD_V3",
        "privacyPassV3":"PRIVACY_PASS_V3"
    }
```

- credentials.wmi:
    - In the picker you will use the WMI credential picker.
    - Will store your IDs as data.
    - In executions it will check if the credential id exists, and if it exists it will use a base64 of a JSON with the credential data as value. Otherwise it will go empty.
    - The JSON for this type of credentials will have this format:

```
    {
        "user":"USER",
        "password":"PASSWORD",
        "namespace":"NAMESPACE"
    }
```

- os:
  - In the selector it will use the names of the OS.
  - Will store the IDs as data.
  - On executions it will check if the OS ID exists, and if it exists it will use its name as the value. Otherwise it will go empty.

Base file

The following example can be used as a basis for creating discovery_definition.ini files, as it includes all the possible blocks and parameters explained above with comments for each case:

```
;-------------------------------------;
; DISCOVERY APPLICATION INI FILE BASE ;
;-------------------------------------;

; Mandatory
; Defines global application data

[discovery_extension_definition]

; Mandatory
; Defines discovery application short name
; Short name must be unique
; Short names with "pandorafms." prefixes are used by Pandora FMS for official
applications

short_name = DISCOVERY_APPLICATION_UNIQUE_NAME

; Mandatory
; Defines the section where application will be shown in console
; Possible values:
; apps
; clouds
; custom

section = app

; Mandatory
; Defines discovery application name, shown in console

name = DISCOVERY_APPLICATION_NAME

; Mandatory
; Defines discovery application version, shown in console

version = VERSION

; Optional
; Defines discovery application description, shown in console
```

```
description = DESCRIPTION

; Optional
; Defines execution files inside .disk
; Several execution files can be defined

execution_file[_EXEC_MACRO_N_] = SCRIPT.pl

; Mandatory
; Defines execution for discovery server
; Several executions can be defined
; At least 1 is required for server execution

exec[] = SERVER_EXECUTION

; Optional
; Define password encrypt script

passencrypt_script = PASS_ENCRYPT_SCRIPT.pl

; Optional
; Defines password encrypt script execution format
; _passencrypt_script_ is replaced with the passencrypt_script file path
; _password_ is replaced with the string (password) to encrypt

passencrypt_exec = EXECUTION

; Optional
; Define password decrypt script

passdecrypt_script = PASS_DECRYPT_SCRIPT.pl

; Optional
; Defines password encrypt script execution format
; _passdecrypt_script_ is replaced with the passdecrypt_script file path
; _password_ is replaced with the string (password) to encrypt

passdecrypt_exec = EXECUTION

; Optional
; Defines the default values for the fields when a new task is created
; By default all values are empty or not selected
; Several default values can be defined
; Possible values depending on field type:
; string - STRING
; number - NUMBER
; password-STRING
; textarea-STRING
; checkbox-BOOL
; select-VALUE_N
; multiselect - [VALUE_1,VALUE_N]
; tree - [VALUE_1,VALUE_N]
```

```
default_value[_FIELD_MACRO_N_] = DEFAULT_VALUE

;-------------------------------------;

; Optional
; Defines application configuration steps

[config_steps]

; Following parameters can be setup for each configuration step
; Several steps can be defined using a different key (N)

; Mandatory
; Defines configuration step name

name[N] = STEP_NAME

; Mandatory if not custom_fields defined
; Defines configuration fields retrieved to console by a command execution
; execution_file scripts can be used to retrieve data
; Command execution output must be JSON as follows
; [
; {
; "macro": "_FIELD_MACRO_N_",
; "mandatory_field": "BOOL",
; "name": "FIELD_NAME",
; "tip": "FIELD_TIP",
; "type": "FIELD_TYPE",
; "placeholder": "PLACEHOLDER",
; "show_on_true": "_FIELD_MACRO_N_",
; "encrypt_on_true": "_FIELD_MACRO_N_",
; "select_data": {
; "VALUE_1": "TEXT_1",
; "VALUE_N": "TEXT_N"
; },
; "tree_data": [
; {
; "name": "TEXT_1",
; "selectable": "BOOL_1",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1",
; childin": [
; {
; "name": "TEXT_1_1",
; "selectable": "BOOL_1_1",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1_1",
; "children": []
; },
; {
; "name": "TEXT_1_N",
; "selectable": "BOOL_1_N",
```

```
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_1_N",
; "children": []
; }
; ]
; },
; {
; "name": "TEXT_N",
; "selectable": "BOOL_N",
; "macro": "_FIELD_MACRO_N_",
; "value": "VALUE_N",
; "children": []
; }
; ]
; }
; ]

script_data_fields[N] = CONSOLE_EXECUTION_FIELDS

; Mandatory if not script_data_fields defined
; Defines custom configuration fields

custom_fields[N] = CUSTOM_FIELDS_N

; Optional
; Defines the number of fields columns for the configuration steps (1 or 2)

fields_columns[N] = M

;-------------------------------------;

; Mandatory if not script_data_fields defined
; Defines custom configuration fields to be used by configuration steps

[CUSTOM_FIELDS_N]

; Mandatory
; Defines configuration field unique macro
; Macros can be used for script executions, using their value in place

macro[N] = _FIELD_MACRO_N_

; Optional
; Defines if configuration field is mandatory or not
; By default all configuration fields are mandatory

mandatory_field[N] = BOOL

; Mandatory
; Defines configuration field name to be displayed in console
; Macro will be used as name if this field is empty
```

```
name[N] = FIELD_NAME

; Optional
; Define a tip for the field, to be displayed in console

tip[N] = FIELD_TIP

; Mandatory
; Defines the field type to be displayed in console
; Possible values:
; string
; number
; password
; textarea
; check box
; select
; multi select
; tree

type[N] = FIELD_TYPE

; Optional if type is string or textarea
; Defines a placeholder for the field, to be displayed in console

placeholder[N] = PLACEHOLDER

; Optional
; Field is shown in console only if assigned field macro exists, is checkbox and
is true

show_on_true[N] = _FIELD_MACRO_N_

; Optional if type is password
; Field value is encrypted when stored into database if assigned field macro
exists, is checkbox and is true
; Password encrypt and decrypt depends on scripts uploaded to the discovery
application

encrypt_on_true[N] = _FIELD_MACRO_N_

; Mandatory if type is select or multiselect
; Defines select data to be displayed in console
; Possible values:
; agent_groups - Uses agent groups names
; agents - Uses agents names
; module_groups - Uses module groups
; modules - Uses modules names
; module_types - Uses module types names
; tags - Uses module tags
; status - Uses module status names
; alert_templates - Uses alert templates names
; alert_actions - Uses alert actions names
```

```
; interval - Uses time interval selector
; credentials.custom - Uses pandora custom credentials selector
; credentials.aws - Uses pandora AWS credentials selector
; credentials.azure - Uses pandora Microsoft Azure credentials selector
; credentials.gcp - Uses pandora Google Cloud Platform credentials selector
; credentials.sap - Uses pandora SAP credentials selector
; credentials.snmp - Uses pandora SNMP credentials selector
; credentials.wmi - Uses pandora WMI credentials selector
; os - Uses pandora OS names
; CUSTOM_SELECT_N - Uses custom selector values
; multiselect fields value is a comma separated list of selected values

select_data[N] = FIELD_DATA

; Mandatory if type is tree
; Defines tree data to be displayed in console
; tree fields value is a comma separated list of selected values

tree_data[N] = CUSTOM_TREE_DATA_N

;-------------------------------------;

; Mandatory if custom tree defined
; Defines custom tree values to be used by configuration fields

[CUSTOM_TREE_DATA_N]

; Mandatory
; Defines the name for the tree element
; Several names can be defined

name[N] = TEXT_N

; Optional
; Defines if tree element is selectable or not
; By default all tree elements are selectable
; Several selectables can be defined

selectable[N] = VALUE_N

; Optional if selectable is true
; Defines the macro where value is stored for the tree element
; Several macros can be defined
; Same macro can be defined for several tree elements inside the same tree
; Macro can't be the same as other outside the tree
; If no macro is defined, value is stored for the global tree macro
; Tree values are stored and used the same way as multiselect values do

macro[N] = FIELD_MACRO_N

; Mandatory if selectable is true
; Defines the value for the tree element
```

```
; Several values can be defined

value[N] = VALUE_N

; Optional
; Defines the children elements for the tree element
; CUSTOM_TREE_DATA_M can't be the same than in an upper level
; Several children can be defined

children[N] = CUSTOM_TREE_DATA_M

;-------------------------------------;

; Mandatory if custom select or multiselect defined
; Defines custom select or multiselect values to be used by configuration fields

[CUSTOM_SELECT_N]

; Mandatory
; Defines the value-text pair for the select or multiselect
; Several value-text pairs can be defined

option[VALUE_N] = TEXT_N

;-------------------------------------;

; Optional
; Defines temporary configuration files to be created and used during script
executions

[tempfile_confs]

; Mandatory
; Defines the content for the temporary file
; File will be used where temporary file macro is specified during executions
; File content replaces fields macros with their values

file[_TEMP_FILE_MACRO_N_] = _FIELD_MACRO_N_,_FIELD_MACRO_M_
```

Example

Below we show an example of the discovery_definition.ini file and how its form would be displayed in the Pandora FMS console:

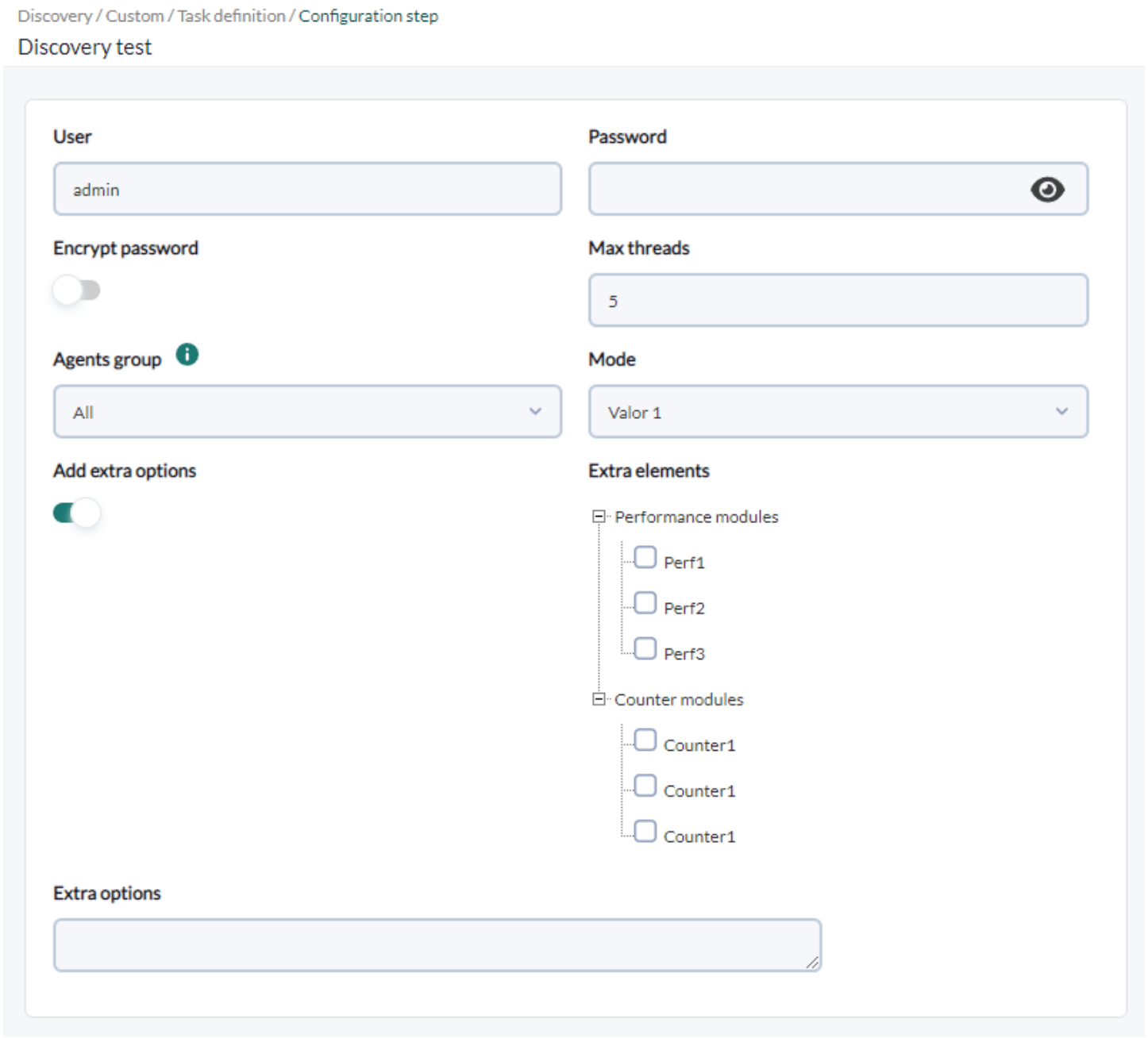
```
[discovery_extension_definition]

short_name = discoveryTest
section = custom
name = Discovery test
version="1.0"
```

```
description = A test discovery plugin

execution_file[_exec1_] = test.py

exec[] = "'_exec1_' -c '_tempConf_'"

passencrypt_script = pass_encrypter.py
passencrypt_exec = "'_passencrypt_script_' --encrypt '_password_'"

passdecrypt_script = pass_encrypter.py
passencrypt_exec = "'_passdecrypt_script_' --decrypt '_password_'"

default_value[_param1_] = "admin"
default_value[_param2_] = ""
default_value[_param3_] = false
default_value[_param4_] = 5
default_value[_param5_] = 0
default_value[_param6_] = v1
default_value[_param7_] = true
default_value[_param8_] = "[]"
default_value[_param9_] = ""
default_value[_param10_] = "[]"
default_value[_param11_] = "[]"

[config_steps]

name[1] = Configuration step
custom_fields[1] = custom_fields_1

[tempfile_confs]

file[_tempConf_] = "user _param1_
password _param2_
encrypt_pass _param3_

threads _param4_

group _param5_

mode _param6_
extra_options _param7_
extra_elements _param8_
extra_perfs _param10_
extra_counters _param11_

log __temp__/__taskMD5__.log

_param9_"

[custom_fields_1]

macro[1] = _param1_
```

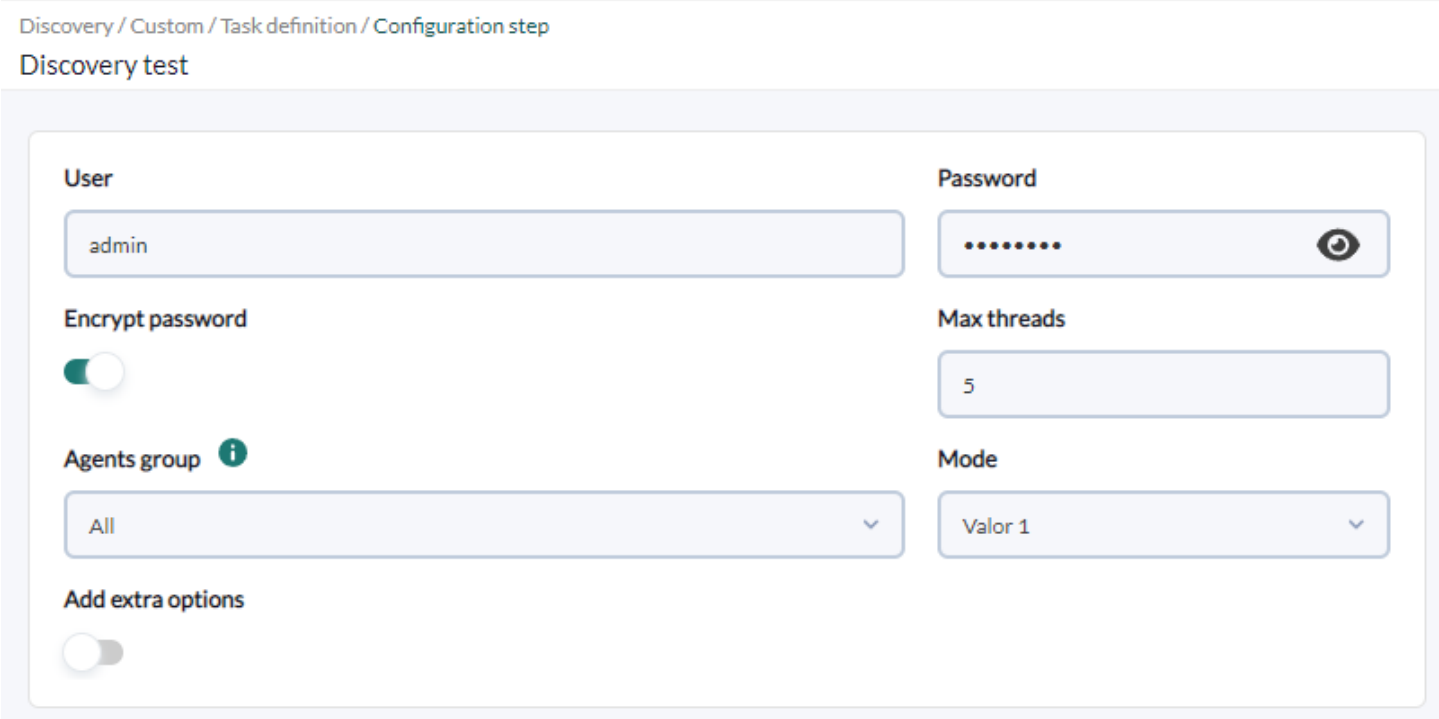
```
name[1] = User
type[1] = string

macro[2] = _param2_
name[2] = Password
type[2] = password
encrypt_on_true[2] = _param3_

macro[3] = _param3_
name[3] = Encrypt password
type[3] = checkbox

macro[4] = _param4_
name[4] = Max threads
type[4] = number
mandatory_field[4] = false

macro[5] = _param5_
name[5] = Agents group
tip[5] = Agents are generated in this group
type[5] = select
select_data[5] = agent_groups

macro[6] = _param6_
name[6] = Mode
type[6] = select
select_data[6] = custom_select_1

macro[7] = _param7_
name[7] = Add extra options
type[7] = checkbox

macro[8] = _param8_
name[8] = Extra elements
type[8] = tree
tree_data[8] = custom_tree_1
show_on_true[8] = _param7_

macro[9] = _param9_
name[9] = Extra options
type[9] = textarea
placeholder[9] = "Add extra options here"
show_on_true[9] = _param7_

[custom_select_1]

option[v1] = Value 1
option[v2] = Value 2
option[v3] = Value 3
option[v4] = Value 4
option[v5] = Value 5
```

```
[custom_tree_1]

name[1] = Performance modules
selectable[1] = false
children[1] = custom_tree_1_A

name[2] = Counter modules
selectable[2] = false
children[2] = custom_tree_1_B

[custom_tree_1_A]

name[1] = Profile1
selectable[1] = true
macro[1] = _param10_
value[1] = p1

name[2] = Perf2
selectable[2] = true
macro[2] = _param10_
value[2] = p2

name[3] = Perf3
selectable[3] = true
macro[3] = _param10_
value[3] = p3

[custom_tree_1_B]

name[1] = Counter1
selectable[1] = true
macro[1] = _param11_
value[1] = c1

name[2] = Counter1
selectable[2] = true
macro[2] = _param11_
value[2] = c2

name[3] = Counter1
selectable[3] = true
macro[3] = _param11_
value[3] = c3
```

With the definition above, when creating a task for the application, this would be the form that would be displayed:



The “Extra elements” and “Extra options” fields are hidden when “Add extra options” is unselected, and the “Password” field would encrypt its value by selecting the “Encrypt password” field:



When this task is executed by the server, it generates a temporary file (for example `/var/spool/pandora/data_in/discovery/tmp/6d7fce9fee471194aa8b5b6e47267f03`) whose content could be:

```
user admin
password MjZhYjBkYjkwZDcyZTI4YWQwYmExZTIyZWU1MTA1MTAgIC0K
encrypt_pass 1

thread 2

group Applications

mode v1
extra_options 1
extra_elements[]
extra_perfs["p1", "p2"]
extra_counters["c1"]

log /tmp/b026324c6904b2a9cb4b88d6d61c81d1.log
```

And which it would use during execution, which would be similar to this:

```
'/var/spool/pandora/data_in/discovery/discoveryTest/test.py' -c
'/var/spool/pandora/data_in/discovery/tmp/6d7fce9fee471194aa8b5b6e47267f03'
```

Scripts and executables

The Pandora FMS server will be in charge of executing the scripts and executables defined in the `discovery_definition.ini` file and will determine the result and summary of the execution of

each task based on the output and error code of each of the executions carried out.

To determine the status, it will check the error code returned by each of the executions it performs. If at least one of the executions returns an error code other than `0`, the task status will be considered failed.

To show the summary of the task, for each of the executions carried out by the Pandora FMS server it will collect its output, both the standard output (STDOUT) and the error output (STDERR). Typically you would expect output in the following minimal JSON format:

```
{
     "summary": {
         "SUMMARY_FIELD_1": "SUMMARY_VALUE_1",
         "SUMMARY_FIELD_N": "SUMMARY_VALUE_N"
     },
     "info": "ADDITIONAL_INFO"
}
```

In the console the summary key will be read as a two-column table, considering its keys as the elements on the left and its values as the elements on the right.

The info key will be read as additional information, adding one more row to the summary table.

Any other key of the JSON or in case of not returning a JSON or one that does not comply with that structure, the rest of the information will be added in one more row in the summary table.

A summary table will be generated for each of the executions, listing them based on the order of executions performed by the server. In this way, the “Summary 1” table would correspond to the first execution, the “Summary 2” table to the second, and so on.

This is intended to ensure that the result of the last execution is visible in the console at all times, whether it was successful or failed.

Finally, apart from the actions carried out by the scripts or executables, the Pandora FMS server will be able to process agents and modules from the output of the executions. For this, the Pandora FMS server will expect to receive in the JSON output an additional key monitoring_data with the information of each of the agents and modules that it must process, but the data of this key will not be stored in the execution summary. .

Thus, the expected JSON output format for Discovery plugins is this:

```
{
     "summary": {
         "SUMMARY_FIELD_1": "SUMMARY_VALUE_1",
         "SUMMARY_FIELD_N": "SUMMARY_VALUE_N"
     },
     "info": "ADDITIONAL_INFO",
```

```
    "monitoring_data": [
        {
            "agent_data": {
                "agent_name": "AGENT_NAME",
                "agent_alias": "AGENT_ALIAS",
                "Bears",
                "os_version": "OS_VERSION",
                "interval": "INTERVAL",
                "id_group": "ID_GROUP",
                "address": "ADDRESS",
                "description": "DESCRIPTION",
                "agent_version": "AGENT_VERSION",
                "parent_agent_name": "PARENT_AGENT_NAME",
                "timezone_offset": "TIMEZONE_OFFSET"
            },
            "module_data": [
                {
                    "name": "NAME",
                    "data": "DATA",
                    "type": "TYPE",
                    "description": "DESCRIPTION",
                    "max": "MAX",
                    "min": "MIN",
                    "post_process": "POST_PROCESS",
                    "module_interval": "MODULE_INTERVAL",
                    "min_critical": "MIN_CRITICAL",
                    "max_critical": "MAX_CRITICAL",
                    "min_warning": "MIN_WARNING",
                    "max_warning": "MAX_WARNING",
                    "disabled": "DISABLED",
                    "min_ff_event": "MIN_FF_EVENT",
                    "datalist": "DATALIST",
                    "status": "STATUS",
                    "unit": "UNIT",
                    "timestamp": "TIMESTAMP",
                    "module_group": "MODULE_GROUP",
                    "custom_id": "CUSTOM_ID",
                    "str_warning": "STR_WARNING",
                    "str_critical": "STR_CRITICAL",
                    "critical_instructions": "CRITICAL_INSTRUCTIONS",
                    "warning_instructions": "WARNING_INSTRUCTIONS",
                    "unknown_instructions": "UNKNOWN_INSTRUCTIONS",
                    "tags": "TAGS",
                    "critical_inverse": "CRITICAL_INVERSE",
                    "warning_inverse": "WARNING_INVERSE",
                    "quiet": "QUIET",
                    "module_ff_interval": "MODULE_FF_INTERVAL",
                    "alert_template": "ALERT_TEMPLATE",
                    "crontab":"CRONTAB",
                    "min_ff_event_normal": "MIN_FF_EVENT_NORMAL",
                    "min_ff_event_warning": "MIN_FF_EVENT_WARNING",
                    "min_ff_event_critical": "MIN_FF_EVENT_CRITICAL",
```

```
                    "ff_timeout": "FF_TIMEOUT",
                    "each_ff": "EACH_FF",
                    "module_parent": "MODULE_PARENT",
                    "module_parent_unlink": "MODULE_PARENT_UNLINK",
                    "cron_interval": "CRON_INTERVAL",
                    "ff_type": "FF_TYPE",
                    "min_warning_forced": "MIN_WARNING_FORCED",
                    "max_warning_forced": "MAX_WARNING_FORCED",
                    "min_critical_forced": "MIN_CRITICAL_FORCED",
                    "max_critical_forced": "MAX_CRITICAL_FORCED",
                    "str_warning_forced": "STR_WARNING_FORCED",
                    "str_critical_forced": "STR_CRITICAL_FORCED"
                }
            ]
        }
    ]
}
```