The Oomnitza Connector

Oomnitza has created a unified connector, lovingly crafted using Python, which is a single application that can be used to pull data from multiple sources and push it to your Oomnitza application. The connector can presently pull data from the following sources, with more planned in the future.

- Airwatch http://www.air-watch.com
- Azure Users https://azure.microsoft.com
- BambhooHR http://www.bamboohr.com
- Casper (Jamf Pro) https://www.jamf.com/products/Jamf-Pro/
- Google Chrome devices https://developers.google.com/admin-sdk/directory/
- Google mobile devices https://developers.google.com/admin-sdk/directory/
- Chef https://www.chef.io/chef/
- Jasper http://www.jasper.com
- LDAP e.g., http://www.openldap.org, Active Directory
- MobileIron http://www.mobileiron.com
- Meraki Systems Manager https://documentation.meraki.com/SM/Systems_Manager_Quick_Start
- Okta https://www.okta.com
- OneLogin https://www.onelogin.com
- Open-AudIT https://www.open-audit.org/
- SCCM http://www.microsoft.com
- ZenDesk https://www.zendesk.com
- Plain CSV files

The Oomnitza Connector can be hosted on Oomnitza's server cloud, free of charge, if the third party server is or can be made accessible from the Oomnitza Cloud. Contact us for more details! Organizations with dedicated internal services may prefer to run this connector in-house, behind the same firewall that prevents outside access.

Getting Started

The most current version of this documentation can always be found on GitHub.

Since Oomnitza is highly customizable, there are many possibilities with the connector. Because of this, it is important to think ahead about what data you want to bring in and how you want to store it. Before we begin, take time to think about what information you want, and what Oomnitza fields you want filled out with data. If the fields you want to map in haven't been created yet, now is a good time to do so. (Refer to our Guide to creating custom fields in Oomnitza to get started.)

Runtime Environment Setup

You will need to install Python 2.7.X as well as the packages which the connector relies upon. Some of the python packages may require build tools to be installed.

Please visit the sections below related to the build tools before installing the additional modules.

We suggest you setup a virtual environment and use `pip` to install the requirements. This can be done as follows (See our documentation on installing additional Python modules for use in Oomnitza.):

```
cd /path/to/connector
virtualenv .
source bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

Linux Environment

On Ubuntu, the build tools are installed using:

```
sudo apt-get install build-essential unixodbc unixodbc-dev
```

Windows Environment

For MS Windows you have to install Windows C++ compilers as build tools. Please visit the Python Wiki To check what is appropriate compiler you have to download and install.

OS X Environment

For OS X environment you have to install the build tools using the following command:

```
xcode-select --install
```

Storage for Connector secrets

To prevent secrets sprawl and disclosure the Oomnitza Connector uses secret backends to securely store credentials, usernames, API tokens, and passwords.

There are two options:

- local KeyRing;
- external the Vault Key Management System by Hashicorp (the Vault KMS).

KeyRing (KeyChain) is a secure encrypted database and the easiest to configure.

The Vault KMS provides an additional layer of security. In this case, all secrets will be stored in the external encrypted backend:

Common recommendations:

Before adding secrets for Connector, first, follow the instructions and setup the Oomnitza Connector. Use a technical role with restricted permissions to run the Connector. It is recommended to use web server for the Vault KMS. To avoid the Vault KMS API call unauthorized using restrict IPs (see section Network Security) for the Vault KMS API call and connected system. For example, in Nginx configuration for the Vault KMS:

```
location / {
  # allow connector workstation IP
  allow    192.168.1.4;
  # drop rest of the world
  deny    all;
}
```

Deployment and receiving secrets

Local KeyRing description

OS Supported:

Ubuntu Linux: SecretStorage (requires installation of additional packages).

Windows: Windows Credential Manager (by default).

OS X: KeyChain. The encryption is AES 128 in GCM (Galois/Counter Mode).

OS X Note: the *keyring==8.7* tested on Mac OS X 10.12.6.

1. To use local KeyRing specify `keyring` as the `vault_backend` in config

```
[oomnitza]
url = https://example.com
vault_backend = keyring
vault_keys = api_token
```

For Linux, you may have to install dbus-python package and configure KeyRing Daemon.

2. To add secrets use the command line utility which enables an easy way to place secrets to the system keyring service.

```
$ python strongbox.py --help
usage: strongbox.py [-h] [--version] --connector=CONNECTOR --key=KEY --value=VALUE

optional arguments:
```

```
  -h, --help            show this help message and exit
  --version             Show the vault version.
  --connector CONNECTOR Connector name under which secret is saved in a vault.
  --key KEY             Secret key name.
  --value VALUE         Secret value. Will be requested.
```

To prevent password disclosure you will be asked to provide your secret value in the console.

```
python strongbox.py --connector=oomnitza --key=api_token --value=
Your secret: your-secret
```

You can add a few secrets to one type of Connector.

The Vault KMS run-up

To use the Vault KMS:

1. Install, initialize and unseal the Vault KMS (use documentation).
2. Mount Key/Value Secret Backend.
3. Write secrets to Key/Value Secret Backend. For example:

```
$ vault write secret/zendesk \
    system_name=oomnitza_user \
    api_token=123456789ABCD$$$
Success! Data written to: secret/zendesk
```

4. Create a json/hcl file with policy:

This section grants all access on "*secret/zendesk**". Further restrictions can be applied to this broad policy.

```
path "secret/zendesk/*" {
  capabilities = ["read", "list"]
}
```

5. To add this policy to the Vault KMS system policies list use the following command or API:

```
vault write sys/policy/zendesk-read policy=@/root/vault/zendesk-read.hcl
```

6. To create a token with assigned policy:

```
vault token-create -policy=zendesk-read -policy=zendesk-read -policy=logs
Token: 6c1247-413f-4816-5f=a72-2ertc1d2165e
```

7. To use Hashicorp Vault as a secret backend set "vault_backend = vault" instead of "keyring".

```
[zendesk]
enable = true
url = https://example.com
vault_backend = vault
vault_keys = api_token username password
```

8. To connect to the Hashicorp Vault the `vault_url` and `vault_token` should be added to system keyring via vault cli.

Use `strongbox.py` cli to add `vault_url` and `vault_token` to system keyring

```
python strongbox.py --connector=zendesk --key=vault_url --value=
Your secret: https://vault.adress.com/v1/secret/zendesk

python strongbox.py --connector=zendesk --key=vault_token --value=
Your secret: 6c1247-413f-4816-5f=a72-2ertc1d2165e
```

It is recomended to use read-only token.

The CyberArk secret storage

In order to use CyberArk as secret storage:

1. Install and configure CyberArk storage (use self-hosted storage or use dedicated storage provided by CyberArk)
2. Write secrets to CyberArk storage
3. Configure connector to use CyberArk secret backend

Self-hosted CyberArk installation

- Use official documentation to install and configure CyberArk Conjur service

```
# In your terminal, download the Conjur Open Source quick-start configuration
curl -o docker-compose.yml https://www.conjur.org/get-started/docker-compose.quickstart.yml

# Pull all of the required Docker images from DockerHub.
docker-compose pull

# Generate a master data key:
docker-compose run --no-deps --rm conjur data-key generate > data_key

# Load the data key into the environment:
export CONJUR_DATA_KEY="$(< data_key)"

# Run the Conjur server, database, and client:
docker-compose up -d
```

  If you want to confirm that CyberArk was installed properly, you can open a browser and go to localhost:8080 and view the included status UI page.

- Create new account (name should be equal to your connector configration section)

```
docker-compose exec conjur conjurctl account create zendesk
# API key for admin: <cyberark_api_key>
```

- Start a bash shell for the Conjur client CLI

```
docker-compose exec client bash
```

- Login into your account

```
conjur init -u conjur -a zendesk
conjur authn login -u admin
```

- Create root policy, e.g. use more complex / nested configuration for granular permissions

```
$ cat /root/policy/zendesk-policy.yml

---
- !variable api_token
- !variable some_secret_key
```

- Apply policy to your account (this allow you manage the specified secrets via command line or api)

```
conjur policy load root /root/policy/zendesk-policy.yml
```

Managing secrets via CyberArk

- Push all required secrets into storage

```
conjur variable values add api_token secret-api-token
conjur variable values add some_secret_key some-secret-value
```

Connector configuration

- To connect to the CyberArk secret storage - the `vault_url` and `vault_token` should be added to system keyring via cli.

Use `strongbox.py` cli to add `vault_url` and `vault_token` to system keyring

```
python strongbox.py --connector=zendesk --key=vault_url --value=
Your secret: https://cyberark-secret-storage-sever.example.com

python strongbox.py --connector=zendesk --key=vault_token --value=
Your secret: <cyberark_api_key>
```

- Update connector configuration to use CyberArk secret storage

```
[zendesk]
enable = true
url = https://example.com
vault_backend = cyberark
vault_keys = api_token some_secret_key
```

Running the connector client

The connector is meant to be run from the command line and as such as multiple command line options:

```
$ python connector.py -h
usage: connector.py [-h] [--record-count RECORD_COUNT]
                    [--version] [--workers WORKERS] [--show-mappings]
                    [--testmode] [--save-data] [--ini INI]
                    [--logging-config LOGGING_CONFIG]
                    {upload,generate-ini} [connectors [connectors ...]]

positional arguments:
  {upload,generate-ini}
                        Action to perform.
  connectors            Connectors to run.

optional arguments:
  -h, --help            show this help message and exit
  --record-count RECORD_COUNT
                        Number of records to pull and process from connection.
  --version             Show the connector version.
  --workers WORKERS     Number of async IO workers used to pull & push
                        records.
  --show-mappings       Show the mappings which would be used by the
                        connector.
  --testmode            Run connectors in test mode.
  --save-data           Saves the data loaded from other system.
  --ini INI             Config file to use.
  --logging-config LOGGING_CONFIG
                        Use to override logging config file to use.
```

The available actions are:

- `generate-ini`: generate an example `config.ini` file.
- `upload`: uploads the data from the indicated connectors to Oomnitza. The connector values are taken from the section names in the ini file.

`--ini` is used to specify which config file to load, if not provided, `config.ini` from the root directory will be used. This option can be used with the `generate-ini` action to specify the file to generate.

`--logging-config` is used to specify an alternate logging config file.

`--show-mappings` is used to print out the loaded mappings. These mappings can be a combination of the built-in mappings, `config.ini` mappings, and mappings setup via the website.

--testmode will print out the records which would have been sent rather than pushing the data to the server. This can be used to see what, exactly, is getting sent to the server.

--record-count is used to limit the number of records to process. Once this number have been processed, the connector will exit. This can be used with --testmode to print out a limited number of records then exit cleanly.

--save-data is used to save the data loaded from the remote system to disk. These files can then be used to confirm the data is being loaded and mapped as expected.

--workers is used to setup the number of workers used to push the extracted data to Oomnitza instance. Default is 2. If you will increase this value it will increase the load generated by connector and decrease the time required to finish the full sync.

Setting the connector to run as an automated task

There are many ways to automate the sync, here are a few:

- OS X: http://www.maclife.com/article/columns/terminal_101_creating_cron_jobs
- OS X: http://superuser.com/questions/126907/how-can-i-get-a-script-to-run-every-day-on-mac-os-x
- OS X: http://launched.zerowidth.com/
- Linux: http://www.cyberciti.biz/faq/how-do-i-add-jobs-to-cron-under-linux-or-unix-oses/
- Windows: http://bytes.com/topic/python/answers/32605-windows-xp-cron-scheduler-python

Running the connector server

It is possible to setup the connector server that will handle webhooks and other requests from external sources and react to them. The connector server is WSGI compliant server (https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface). The connector server is meant to be run from the command line with following command line arguments:

```
$ python server.py -h
usage: server.py [-h] [--host HOST] [--port PORT] [--version]
                 [--show-mappings] [--testmode] [--save-data] [--ini INI]
                 [--logging-config LOGGING_CONFIG]

optional arguments:
  -h, --help            show this help message and exit
  --host HOST
  --port PORT
  --version             Show the connector version.
  --show-mappings       Show the mappings which would be used by the
                        connector.
  --testmode            Run connectors in test mode.
  --save-data           Saves the data loaded from other system.
  --ini INI             Config file to use.
  --logging-config LOGGING_CONFIG
                        Use to override logging config file to use.
```

The available arguments for the connector server are serving for the same purposes as for the connector client, except 2 new server-specific arguments:

--host is used to specify the server's host. Default is 127.0.0.1

--port is used to specify the server's port. Default is 8000

The url pointing to the connector server instance should ends with the name of the connector:

Examples:

```
https://my-connector-server.com/it/does/not/matter/casper
https://my-connector-server.com/it/does/not/matter/casper.MDM
https://my-connector-server.com/it/does/not/matter/casper.1
```

Note: Now only the Casper (JAMF Pro) Webhooks are supported out of the box by the connector server. First you have to enable webhooks with JSON payloads (http://docs.jamf.com/9.96/casper-suite/administrator-guide/Webhooks.html) Out of the box the following webhooks are supported:

- ComputerAdded
- ComputerCheckIn

- ComputerInventoryCompleted
- ComputerPolicyFinished
- ComputerPushCapabilityChanged
- MobileDeviceCheckIn
- MobileDeviceCommandCompleted
- MobileDeviceEnrolled
- MobileDevicePushSent
- MobileDeviceUnEnrolled

As soon as the connector server is receiving the message from the Casper (JAMF Pro) it is initiating the request to fetch the details of the asset triggered that webhook request and sync it with Oomnitza. This allows efficiently and fast keep the Oomnitza in the up-to-date state.

Also please keep in mind that the webhooks from the Casper (JAMF Pro) do not add any authorization information in the requests to the connector server. Because of that server cannot verify incoming requests in any way, so if you want somehow forbid access to the connector's server interface configured to listen to Casper (JAMF Pro) webhooks, you have to use something else, like firewalls with configured allowed IPs, etc.

Connector Configs

Now you should be able to generate a default config file. Running `python connector.py generate-ini` will regenerate the `config.ini` file, and create a backup if the file already exists. When you edit this file, it will have one section per connection. You can safely remove the section for the connections you will not be using to keep the file small and manageable.

If you require multiple different configurations of a single connector, such as the need to pull from two different LDAP OUs, additional sections can be added by appending a '.' and a unique identifier to the section name. For example, having both a `[ldap]` and `[ldap.Contractors]` section will allow you to pull users from a default and Contractor OU.

An example generated `config.ini` follows.

```
[oomnitza]
url = https://example.oomnitza.com
api_token =
username = oomnitza-sa
password = ThePassword

[airwatch]
enable = False
url = https://apidev.awmdm.com
username = username@example.com
password = change-me
api_token = YOUR AirWatch API TOKEN
sync_field = 24DCF85294E411E38A52066B556BA4EE
dep_uuid =

[azureusers]
enable = False
tenant_id =
client_id =
secret =
default_role = 25
default_position = Employee
sync_field = USER

[bamboohr]
enable = False
url = https://api.bamboohr.com/api/gateway.php
system_name = YOUR BambooHR SYSTEM NAME
api_token = YOUR BambooHR API TOKEN
default_role = 25
sync_field = USER

[casper]
enable = False
url = https://jss.jamfcloud.com/example
username = username@example.com
password = change-me
sync_field = 24DCF85294E411E38A52066B556BA4EE
sync_type = computers
update_only = False

[chef]
enable = False
url = https://chef-server/organizations/ORG/
client = user
```

```
key_file = /home/user/user.pem
sync_field = 24DCF85294E411E38A52066B556BA4EE
attribute_extension =

[chromebooks]
enable = False
service_account_impersonate = username@example.com
service_account_json_key = {}
sync_field = 24DCF85294E411E38A52066B556BA4EE

[jasper]
enable = False
wsdl_path = http://api.jasperwireless.com/ws/schema/Terminal.wsdl
username = username@example.com
password = change-me
storage = storage.db
api_token = YOUR Jasper API TOKEN
sync_field = 24DCF85294E411E38A52066B556BA4EE
update_only = False

[ldap]
enable = False
url = ldaps://ldap.com:389
username = cn=read-only-admin,dc=example,dc=com
password =
base_dn = dc=example,dc=com
group_dn =
protocol_version = 3
filter = (objectClass=*)
default_role = 25
default_position = Employee
page_criterium =
groups_dn = []
group_members_attr = member
group_member_filter =
sync_field = USER

[ldap_assets]
enable = False
url = ldaps://ldap.com:389
username = cn=read-only-admin,dc=example,dc=com
password =
base_dn = dc=example,dc=com
group_dn =
protocol_version = 3
filter = (objectClass=*)
sync_field = 24DCF85294E411E38A52066B556BA4EE
page_criterium =
groups_dn = []
group_members_attr = member
group_member_filter =

[merakism]
enable = False
meraki_api_key =
network_id = N_**********
sync_field = 24DCF85294E411E38A52066B556BA4EE

[mobileiron]
enable = False
url = https://na1.mobileiron.com
username = username@example.com
password = change-me
partitions = ["Drivers"]
sync_field = 24DCF85294E411E38A52066B556BA4EE
api_version = 1

[okta]
enable = False
url = https://example-admin.okta.com
api_token = YOUR Okta API TOKEN
default_role = 25
default_position = Employee
deprovisioned = false
sync_field = USER

[onelogin]
enable = False
url = https://app.onelogin.com/api/v2/users.xml
api_token = YOUR OneLogin API TOKEN
default_role = 25
default_position = Employee
sync_field = USER
```

```
[open_audit]
enable = False
url = http://192.168.XXX.XXX
username = change-me
password = change-me
sync_field = 24DCF85294E411E38A52066B556BA4EE

[sccm]
enable = False
server = server.example.com
database = CM_DCT
username = change-me
password = change-me
authentication = SQL Server
sync_field = 24DCF85294E411E38A52066B556BA4EE

[zendesk]
enable = False
system_name = oomnitza
api_token = YOUR Zendesk API TOKEN
username = username@example.com
default_role = 25
default_position = Employee
sync_field = USER

[csv_assets]
enable = False
filename = /some/path/to/file/assets.csv
directory = /some/path/to/files
sync_field = BARCODE

[csv_users]
enable = False
filename = /some/path/to/file/users.csv
directory = /some/path/to/files
default_role = 25
default_position = Employee
sync_field = USER
```

The `[oomnitza]` section is where you configure the connector with the URL and login credentials for connecting to Oomnitza. You can use an existing user's credentials for username and password, but best practice is to create a service account using your standard naming convention. (See the (documentation)[http://docs) for managing user accounts in Oomnitza.)

The remaining sections each deal with a single connection to an external service. The "enable" field is common to all connections and if set to "True" will enable this service for processing. Some fields are common to a type of connection. For example, "default_role" and "default_user" are fields for connections dealing with loading People into the Oomnitza app.

Each section can end with a list of field mappings. Simple mappings which just copy a field from the external system to a field inside Oomnitza can be defined here or in the System Settings within Oomnitza. Simple mappings are as follows:

```
mapping.[Oomnitza Field] = {"source": "[external field]"}
```

For fields which require processing before being brought into Oomnitza must be defined in the INI. These mappings are more involved. Please contact support@oomnitza.com for more information. The format is:

```
mapping.[Oomnitza Field] = {"source": "[external field]", "converter": "[converter name]"}
```

Oomnitza Configuration

`url`: the url of the Oomnitza application. For example: `https://example.oomnitza.com`

`username`: the Oomnitza username to use

`password`: the Oomnitza password to use

`api_token`: The API Token belonging to the Oomnitza user. If provided, `username` and `password` will not be used.

`user_pem_file`: The path to the PEM-encoded certificate containing the both private and public keys of the user. Has to be used only if there is enabled two factor authentication in your environment. The certificate has to be also uploaded to Oomnitza in the "Settings / Certificates" page.

Airwatch Configuration

`url`: the url of the Airwatch server

`username`: the Airwatch username to use

`password`: the Airwatch password to use

`api_token`: API token for the connection

`sync_field`: The Oomnitza field which contains the asset's unique identifier (we typically recommend serial number).

`dep_uuid`: Additional id of the Apple DEP group used to extend the data pulling from the Airwatch with additional details. Feature is supported by Airwatch starting from v9.2

Default Field Mappings

```
No default mappings
```

CSV Assets Configuration

`filename`: CSV file with assets inside

`directory`: directory with CSV files with assets inside. Note: `filename` and `directory` are mutually exclusive

`sync_field`: The Oomnitza field which contains the asset's unique identifier (we typically recommend serial number).

Default Field Mappings

```
No default mappings. Everything should be defined in the config
```

CSV Users Configuration

`filename`: CSV file with assets inside

`directory`: directory with CSV files with assets inside. Note: `filename` and `directory` are mutually exclusive

`default_role`: The numeric ID of the role which will be assigned to imported users. For example: `25`.

`default_position`: The position which will be assigned to the user. For example: `Employee`.

`sync_field`: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
No default mapping. Everything should be defined in the config
```

BambooHR Configuration

`url`: the url of the BambooHR server

`system_name`: Identifier of your system in the Bamboo HR environment

`api_token`: API token for the connection

`default_role`: The numeric ID of the role which will be assigned to imported users. For example: `25`.

`default_position`: The position which will be assigned to the user. For example: `Employee`.

`sync_field`: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
mapping.USER =           {'source': "workEmail"}
mapping.FIRST_NAME =     {'source': "firstName"}
mapping.LAST_NAME =      {'source': "lastName"}
mapping.EMAIL =          {'source': "workEmail"}
mapping.PHONE =          {'source': "mobilePhone"}
mapping.POSITION =       {'source': "jobTitle"}
mapping.PERMISSIONS_ID = {'setting': "default_role"}
```

Casper Configuration

The [casper] section contains a similar set of preferences; your JSS URL, and the login credentials for an auditor account in Casper (See the Casper Suite Administrator's Guide, pg. 42).

The identifier section of the config.ini file should contain a mapping to a unique field in Oomnitza, which you want to use as the identifier for an asset. Serial Number is the most commonly used identifier since no two assets should share one. This will determine if the script creates a new record for a given serial number on its next sync, or if it updates an existing record that has new information.

url: the url of the Casper server

username: the Casper username to use

password: the Casper password to use. Note: the Casper API will NOT work with a password which contains % or *. ! is an acceptable character to use.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend serial number).

sync_type: Sets the type of data to pull from Casper. Options are computers or mobiledevices. Note: If you need to pull computers AND mobile devices info from Casper, copy Casper configuration section to the same config.ini and name it as 'casper.MDM'. Set the field mapping related to computers in the 'casper' section and set sync_type = computers. Set the field mapping related to mobile devices in the 'casper.MDM' section and set sync_type = mobiledevices

group_name: Specifies the Group from which to load assets. If group_name is missing or empty, all assets will be loaded. If present, only assets from this Group will be processed.

verify_ssl: set to false if the Casper server is running with a self signed SSL certificate.

update_only: set this to True to only update records in Oomnitza. Records for new assets will not be created.

List of currently supported Casper external fields (computers)

```
'general.alt_mac_address'
'general.asset_tag'
'general.barcode_1'
'general.barcode_2'
'general.distribution_point'
'general.id'
'general.initial_entry_date'
'general.initial_entry_date_epoch'
'general.initial_entry_date_utc'
'general.ip_address'
'general.jamf_version'
'general.last_cloud_backup_date_epoch'
'general.last_cloud_backup_date_utc'
'general.last_contact_time'
'general.last_contact_time_epoch'
'general.last_contact_time_utc'
'general.mac_address'
'general.mdm_capable'
'general.name'
'general.netboot_server'
'general.platform'
'general.report_date'
'general.report_date_epoch'
'general.report_date_utc'
'general.serial_number'
'general.sus'
'general.udid'
'hardware.active_directory_status'
'hardware.available_ram_slots'
'hardware.battery_capacity'
'hardware.boot_rom'
'hardware.bus_speed'
'hardware.bus_speed_mhz'
```

```
'hardware.cache_size'
'hardware.cache_size_kb'
'hardware.make'
'hardware.model'
'hardware.model_identifier'
'hardware.nic_speed'
'hardware.number_processors'
'hardware.optical_drive'
'hardware.os_build'
'hardware.os_name'
'hardware.os_version'
'hardware.processor_architecture'
'hardware.processor_speed'
'hardware.processor_speed_mhz'
'hardware.processor_type'
'hardware.service_pack'
'hardware.smc_version'
'hardware.total_ram'
'hardware.total_ram_mb'
'location.building'
'location.department'
'location.email_address'
'location.phone'
'location.position'
'location.real_name'
'location.room'
'location.username'
'purchasing.applecare_id'
'purchasing.is_leased'
'purchasing.is_purchased'
'purchasing.lease_expires'
'purchasing.lease_expires_epoch'
'purchasing.lease_expires_utc'
'purchasing.life_expectancy'
'purchasing.os_applecare_id'
'purchasing.os_maintence_expires'
'purchasing.po_date'
'purchasing.po_date_epoch'
'purchasing.po_date_utc'
'purchasing.po_number'
'purchasing.purchase_price'
'purchasing.purchasing_account'
'purchasing.purchasing_contact'
'purchasing.vendor'
'purchasing.warranty_expires'
'purchasing.warranty_expires_epoch'
'purchasing.warranty_expires_utc'
```

List of currently supported Casper external fields (mobile devices)

```
'general.airplay_password'
'general.asset_tag'
'general.available'
'general.available_mb'
'general.battery_level'
'general.bluetooth_mac_address'
'general.capacity'
'general.capacity_mb'
'general.bluetooth_mac_address'
'general.cloud_backup_enabled'
'general.device_id'
'general.device_name'
'general.device_ownership_level'
'general.display_name'
'general.do_not_disturb_enabled'
'general.id'
'general.initial_entry_date_epoch'
'general.initial_entry_date_utc'
'general.ip_address'
'general.itunes_store_account_is_active'
'general.last_backup_time_epoch'
'general.last_backup_time_utc'
'general.last_cloud_backup_date_epoch'
'general.last_cloud_backup_date_utc'
'general.last_inventory_update'
'general.last_inventory_update_epoch'
'general.last_inventory_update_utc'
'general.locales'
'general.managed'
'general.model'
'general.model_display'
```

```
'general.model_identifier'
'general.modelDisplay'   # looks like the same as 'general.model_display'
'general.modem_firmware'
'general.name'
'general.os_build'
'general.os_type'
'general.os_version'
'general.percentage_used'
'general.phone_number'
'general.serial_number'
'general.supervised'
'general.tethered'
'general.udid'
'general.wifi_mac_address'
'location.building'
'location.department'
'location.email_address'
'location.phone'
'location.position'
'location.real_name'
'location.room'
'location.username'
'network.carrier_settings_version'
'network.cellular_technology'
'network.current_carrier_network'
'network.current_mobile_country_code'
'network.current_mobile_network_code'
'network.data_roaming_enabled'
'network.home_carrier_network'
'network.home_mobile_country_code'
'network.home_mobile_network_code'
'network.iccid'
'network.imei'
'network.roaming'
'network.voice_roaming_enabled'
'purchasing.applecare_id'
'purchasing.is_leased'
'purchasing.is_purchased'
'purchasing.lease_expires'
'purchasing.lease_expires_epoch'
'purchasing.lease_expires_utc'
'purchasing.life_expectancy'
'purchasing.po_date'
'purchasing.po_date_epoch'
'purchasing.po_date_utc'
'purchasing.po_number'
'purchasing.purchase_price'
'purchasing.purchasing_account'
'purchasing.purchasing_contact'
'purchasing.vendor'
'purchasing.warranty_expires'
'purchasing.warranty_expires_epoch'
'purchasing.warranty_expires_utc'
'security.block_level_encryption_capable'
'security.data_protection'
'security.file_level_encryption_capable'
'security.passcode_compliant'
'security.passcode_compliant_with_profile'
'security.passcode_present'
```

Default Field Mappings

```
No default mappings
```

Chef Configuration

The `[chef]` section contains a similar set of preferences.

The identifier section of the `config.ini` file should contain a mapping to a unique field in Oomnitza, which you want to use as the identifier for an asset.

`url`: the full url of the Chef server with organization e.g. https://chef-server/organizations/ORG/

`client`: the Chef username for authentication

`key_file`: the Chef RSA private key for authentication

sync_field: The Oomnitza field internal id which contains the asset's unique identifier.

attribute_extension: [optional] dictionary of additional node attributes to extract

List of currently supported Chef attributes

```
'hardware.name'
'hardware.ip_address'
'hardware.mac_address'
'hardware.hostname'
'hardware.fqdn'
'hardware.domain'
'hardware.platform'
'hardware.platform_version'
'hardware.serial_number'
'hardware.model'
'hardware.total_memory_mb'
'hardware.total_hdd_mb'
'hardware.cpu'
'hardware'cpu_count'
'hardware.uptime_seconds'
```

Attribute Extension

The connector config.ini allows for additional node attributes to be extracted.

Example: attribute_extension = {"__default__": {"kernel_name": "automatic.kernel.name"}}

The above example will introduce a new mappable attribute "hardware.kernel_name". If a particular platform does not have this node attribute, it will processed as empty.

attribute_extension = { "mac_os_x": {"machine_name": "automatic.machinename"}, "windows": {"machine_name": "automatic.foo.bar"} }

The above example will introduce a new mappable attribute "hardware.machine_name" for mac_os_x and windows nodes only.

Google Chrome Devices

NOTE: The Google chrome connector does not have a UI and must be configured using the config file. The Google Chrome device API reference with attributes description can be found here.

The following steps need to be taken to allow automated retrieval from Google Admin API:

1) Enable API access for your G Suite domain 2) Create a new project in Google Developers Console and enable the Admin SDK. 3) Create a service account in this project and delegate a domain-wide authority to it. For more details, please refer to Using OAuth 2.0 for Server to Server Applications and follow the instructions. The Oomnitza connector requires read-only API scope enable to successfully operate *https://www.googleapis.com/auth/admin.directory.device.chromeos.readonly*

The [chromebooks] section contains the following attributes:

service_account_impersonate: the email of the real G Suite administrator to be impersonated by the service account.

service_account_json_key: the content of the JSON key file generated for service account as one line. This key is generated when you create the service account and also you can create additional keys later.

sync_field: The Oomnitza field which contains the asset's unique identifier (i.e. serial number).

Default Field Mappings

```
No default mappings. All mappings need to be defined in the config file.
```

Google Mobile Devices

NOTE: The Google mobile devices connector does not have a UI and must be configured using the config file. The Google mobile devices management API reference with attributes description can be found here.

The following steps need to be taken to allow automated retrieval from Google Admin API:

1) Enable API access for your G Suite domain 2) Create a new project in Google Developers Console and enable the Admin SDK. 3) Create

a service account in this project and delegate a domain-wide authority to it. For more details, please refer to Using OAuth 2.0 for Server to Server Applications and follow the instructions. The Oomnitza connector requires read-only API scope enable to successfully operate *https://www.googleapis.com/auth/admin.directory.device.mobile.readonly*

The [google_mobile_devices] section contains the following attributes:

service_account_impersonate: the email of the real G Suite administrator to be impersonated by the service account.

service_account_json_key: the content of the JSON key file generated for service account as one line. This key is generated when you create the service account and also you can create additional keys later.

sync_field: The Oomnitza field which contains the asset's unique identifier (i.e. serial number).

Default Field Mappings

```
No default mappings. All mappings need to be defined in the config file.
```

Jasper Configuration

wsdl_path: The full URL to the Terminal.wsdl. Defaults to: http://api.jasperwireless.com/ws/schema/Terminal.wsdl.

username: the Jasper username to use

password: the Jasper password to use

storage: The path to the storage file used to maintain state about the connector. Defaults to: storage.db

api_token: The Jasper API Token.

sync_field: The Oomnitza field which contains the asset's unique identifier.

update_only: set this to True to only update records in Oomnitza. Records for new assets will not be created.

Default Field Mappings

```
No default mappings
```

LDAP Configuration

url: The full URI for the LDAP server.

username: the LDAP username to use. Can be a DN, such as cn=read-only-admin,dc=example,dc=com.

password: the LDAP password to use

base_dn: The Base DN to use for the connection.

group_dn: Identifies the group to which the users we want to fetch have to belong. The attribute is a legacy one, see the *groups_dn* attribute below

groups_dn: Identifies the list of groups to which the users we want to fetch have to belong.

NOTE: the attributes *base_dn*, *group_dn* and *groups_dn* define the DN where to fetch the data from. And these attributes have a priority. If the *groups_dn* is defined, the data will be fetched from the mentioned groups and the *group_dn* and *base_dn* will be ignored. Next goes the *group_dn* attribute. The lowest priority has the *base_dn*, it will be taken into the account only if *groups_dn* and *group_dn* are empty.

group_members_attr: Identifies the name of the attribute in the LDAP linking the record inside the group with the group. Default is "member" but can vary in different LDAP systems.

group_member_filter: Identifies the additional optional filter used to extract the details of the user in the group. Empty by default.

protocol_version: The LDAP Protocol version to use. Defaults to: 3.

filter: The LDAP filter to use when querying for people. For example: (objectClass=*) will load all objects. This is a very reasonable default.

default_role: The numeric ID of the role which will be assigned to imported users. For example: 25.

default_position: The position which will be assigned to the user. For example: Employee.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
mapping.USER =          {'source': "uid", 'required': True, 'converter': 'ldap_user_field'},
mapping.FIRST_NAME =    {'source': "givenName"},
mapping.LAST_NAME =     {'source': "sn"},
mapping.EMAIL =         {'source': "mail", 'required': True},
mapping.PERMISSIONS_ID = {'setting': "default_role"},
```

Azure Active Directory Users Configuration

tenant_id: The ID of your tenant.

client_id: The ID of the Service Principal used to access the user's data. Check the official MS docs of how to create a service principal using Azure Portal.

secret: The Service Principal secret/key value

default_role: The numeric ID of the role which will be assigned to imported users. For example: 25.

default_position: The position which will be assigned to the user. For example: Employee.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
No default mappings
```

LDAP Assets Configuration

This is for [ldap_assets] section and actually contains the same set of configuration as for [ldap] used to fetch the user records.

url: The full URI for the LDAP server.

username: the LDAP username to use. Can be a DN, such as cn=read-only-admin,dc=example,dc=com.

password: the LDAP password to use

base_dn: The Base DN to use for the connection.

group_dn: Identifies the group to which the users we want to fetch have to belong. The attribute is a legacy one, see the *groups_dn* attribute below

groups_dn: Identifies the list of groups to which the users we want to fetch have to belong.

NOTE: the attributes *base_dn*, *group_dn* and *groups_dn* define the DN where to fetch the data from. And these attributes have a priority. The highest priority is for *groups_dn*, the data will be fetched from the mentioned groups and the *group_dn* and *base_dn* will be ignored. Next goes the *group_dn* attribute. The lowest priority has the *base_dn*, it will be taken into the account only if *groups_dn* and *group_dn* are empty.

group_members_attr: Identifies the name of the attribute in the LDAP linking the record inside the group with the group. Default is "member" but can vary in different LDAP systems.

group_member_filter: Identifies the additional optional filter used to extract the details of the user in the group. Empty by default.

protocol_version: The LDAP Protocol version to use. Defaults to: 3.

filter: The LDAP filter to use when querying for people. For example: (objectClass=*) will load all objects. This is a very reasonable default.

sync_field: The Oomnitza field which contains the asset's unique identifier.

Default Field Mappings

```
No default mappings
```

Meraki Systems Manager Configuration

`meraki_api_key` = The API key used to access the Meraki API. Please follow the instructions from the "Enable API access" paragraph from the official documentation.

`network_id` = Meraki network identifier.

`sync_field` = The Oomnitza field which contains the asset's unique identifier.

Default Field Mappings

```
No default mappings
```

MobileIron Configuration

`url`: The full URI for the MobileIron server. For example: `https://na1.mobileiron.com`

`username`: the MobileIron username to use.

`password`: the MobileIron password to use.

`partitions`: The MobileIron partitions to load. For example: `["Drivers"]` or `["PartOne", "PartTwo"]`. Used for API v1 and ignored for API v2.

`sync_field`: The Oomnitza field which contains the asset's unique identifier.

`api_version`: The version of MobileIron API used to fetch the records. Available options are `1` and `2`. The cloud instances are using v1 by default. For the CORE instances (on-premise installations) you have to use v2.

Default Field Mappings

```
No default mappings
```

Okta Configuration

`url`: The full URI for the Okta server. For example: `https://oomnitza-admin.okta.com`

`api_token`: The Okta API Token.

`default_role`: The numeric ID of the role which will be assigned to imported users. For example: `25`.

`default_position`: The position which will be assigned to the user. For example: `Employee`.

`deprovisioned`: When it is `false` (default) the users with status `DEPROVISIONED` in Okta will not be pushed to Oomnitza.

`sync_field`: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
mapping.USER =           {'source': "profile.login"},
mapping.FIRST_NAME =     {'source': "profile.firstName"},
mapping.LAST_NAME =      {'source': "profile.lastName"},
mapping.EMAIL =          {'source': "profile.email"},
mapping.PHONE =          {'source': "profile.mobilePhone"},
mapping.PERMISSIONS_ID = {'setting': "default_role"},
mapping.POSITION =       {'setting': "default_position"},
```

OneLogin Configuration

`url`: The full URI for the OneLogin server. For example: `https://api.us.onelogin.com/api/1/users`

`client_id`: The Client ID used to connect to the API.

client_secret: The Client Secret used to connect to th API.

api_token: The OneLogin API Token. Note: OUTDATED. Is used for the old and outdated version of OneLogin API and left for compatibility reasons. if this old API is used another url should be set in the configuration as url: https://app.onelogin.com/api/v2/users.xml. If you have an issues during the connection to the OneLogin, please switch to the new API by defining the correct client_id and client_secret instead of api_token.

default_role: The numeric ID of the role which will be assigned to imported users. For example: 25.

default_position: The position which will be assigned to the user. For example: Employee.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
mapping.USER =           {'source': "username"}
mapping.FIRST_NAME =     {'source': "firstname"}
mapping.LAST_NAME =      {'source': "lastname"}
mapping.EMAIL =          {'source': "email"}
mapping.PHONE =          {'source': "phone"}
mapping.PERMISSIONS_ID = {'setting': "default_role"}
mapping.POSITION =       {'setting': "default_position"}
```

Open-AudIT Configuration

url: The full URI for the Open-AudIT server. For example: http://192.168.111.145

username: The Open-AudIT username used to connect to the API.

password: The Open-AudIT password used to connect to the API.

Default Field Mappings

```
No default mappings
```

List of currently supported Open-AudIT attributes

```
'hardware.name'
'hardware.ip'
'hardware.dbus_identifier'
'hardware.description'
'hardware.dns_hostname'
'hardware.domain'
'hardware.end_of_life'
'hardware.end_of_service'
'hardware.environment'
'hardware.first_seen'
'hardware.form_factor'
'hardware.fqdn'
'hardware.hostname'
'hardware.identification'
'hardware.last_seen'
'hardware.mac_vendor'
'hardware.mac'
'hardware.manufacturer'
'hardware.memory_count'
'hardware.model'
'hardware.os_bit'
'hardware.os_family'
'hardware.os_group'
'hardware.os_installation_date'
'hardware.os_name'
'hardware.os_version'
'hardware.processor_count'
'hardware.purchase_invoice'
'hardware.purchase_order_number'
'hardware.serial_imei'
'hardware.serial_sim'
'hardware.serial'
'hardware.storage_count'
'hardware.uptime_formatted'
'hardware.uptime'
```

```
'hardware.uuid'
'hardware.warranty_expires'
```

SCCM Configuration

The account used to connect to the SCCM database requires at least read-only access.

server: The server hosting the SCCM database.

database: The SCCM database from which to pull data.

username: The username to use when connecting to the server using SQL Server authentication. This user requires read-only access to the DB. Ignored when using Windows authentication.

password: The password to use when connecting to the server using SQL Server authentication. Ignored when using Windows authentication.

driver: The driver name used to communicate with SCCM database. In most of the cases has to be left empty; if left empty the connector will try to use the most recent driver.

In Windows environment there is always should be at least one legacy driver named "SQL Server". In other environments you may have to explicitly install drivers (download drivers).

Please refer to this page for the list of currently supported drivers.

authentication: Sets the type of authentication to use when connecting to the server. Options are SQL Server or Windows. The default is to use SQL Server Authentication. When using Windows authentication, the username and password fields are ignored and the credentials for the currently logged in user will be used when making the connection to the SCCM database.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend serial number).

Default Field Mappings

```
No default mappings
```

Zendesk Configuration

system_name: The Zendesk system name to use. For example: oomnitza

api_token: The Zendesk API Token.

username: the Zendesk username to use.

default_role: The numeric ID of the role which will be assigned to imported users. For example: 25.

default_position: The position which will be assigned to the user. For example: Employee.

sync_field: The Oomnitza field which contains the asset's unique identifier (we typically recommend username or email).

Default Field Mappings

```
mapping.USER =           {'source': "email"}
mapping.FIRST_NAME =     {'source': "name", 'converter': "first_from_full"}
mapping.LAST_NAME =      {'source': "name", 'converter': "last_from_full"}
mapping.EMAIL =          {'source': "email"}
mapping.PHONE =          {'source': "phone"}
mapping.PERMISSIONS_ID = {'setting': "default_role"}
mapping.POSITION =       {'setting': "default_position"}
```

Advanced usage

Logging

The Oomnitza Connector uses the standard python logging module. This modules is configured via the logging.json file. This file can be edited, or copied to a new file, to change the logging behavior of the connector. Please see the python docs for information of configuring

python logging.

SSL Protocol Version

If the service to be connected to requires a particular SSL protocol version to properly connect, the connection's section in the ini file can include a `ssl_protocol` option. The value can be one of: `ssl`, `sslv23`, `sslv3`, `tls`, `tls1`.

Custom Converters

It is possible to create a completely custom complex converter that will be used to convert values extracted from external system to before pushing them to the Oomnitza. To use this option you have to define the name of this converter in the mapping, like this

```
mapping.MY_AWESOME_FIELD =  {"source": "name", "converter": "my_custom_converter"}
```

next you have to define new `[converters]` section in the config with the `my_custom_converter:`. Under this converter name you have to define a valid Python 2.X function, that has to return some value - this value is a result of the converter. In the converter function a "record" object is available, it is the whole record extracted from external system as Python dict object. Example:

```
[ldap]

... here goes config ...

mapping.POSITION = {"source": "position", "converter": "my_custom_converter"}

[converters]
my_custom_converter:
    return record.get("position", "Unknown position")
```

If an exception is raised inside the custom converter's code, a None value is returned as the result

Record Filtering

Support has been added for filtering the records passed from the connector to Oomnitza. By default, all records from the remote system will be sent to Oomnitza for processing. To limit the records based on values in those records, a special `recordfilter` value can be added to a connector section in the ini file. This filter is written using the Python programming language.

For example, the following filter will only process records with the `asset_type` field set to "`computer`":

```
recordfilter:
    return record.asset_type == "computer"
```

This is a very new feature, with many options, and we are still working on the documentation. If you are interested in using this feature, please contact support@oomnitza.com for assistance.

Current limitations

Software mapping

There is no possibility to set the mapping for the software info associated with IT assets (SCCM, JAMF). The only thing can be done is to disable the mapping at all. To do this set the following custom mapping in your `config.ini` file:

```
mapping.APPLICATIONS = {"hardcoded": []}
```

MS Windows environment

In MS Windows environment the "Task Scheduler" is the natural tool to schedule the connector execution. Please note that when the task is scheduled via "Task Scheduler" the "working directory" is not the directory of connector executable, but other one. So if you are using this tool to schedule the connector job, please also set the path to the configuration files via the command line arguments. Or schedule not the connector itself, but the .bat file which is triggering the connector.