Egeria Documentation

Release 0.11.0

MySidesTheyAreGone

December 13, 2016

Contents

1 THIS IS PRE-PRE-PRE-PRE-ALPHA SOFTWARE. DO NOT USE IT. 3

2 About 5

3 Contents 7

3.1 Overview . . . 7
3.2 Learning . . . 9
3.3 The systematic consumption of Knowledge . . . 10
3.4 Controlling the flow of Knowledge . . . 13
3.5 Configuration . . . 14
3.6 Input plugins catalog . . . 15
3.7 Mutator plugins catalog . . . 23
3.8 Output plugins catalog . . . 23

She bestows knowledge and wisdom.

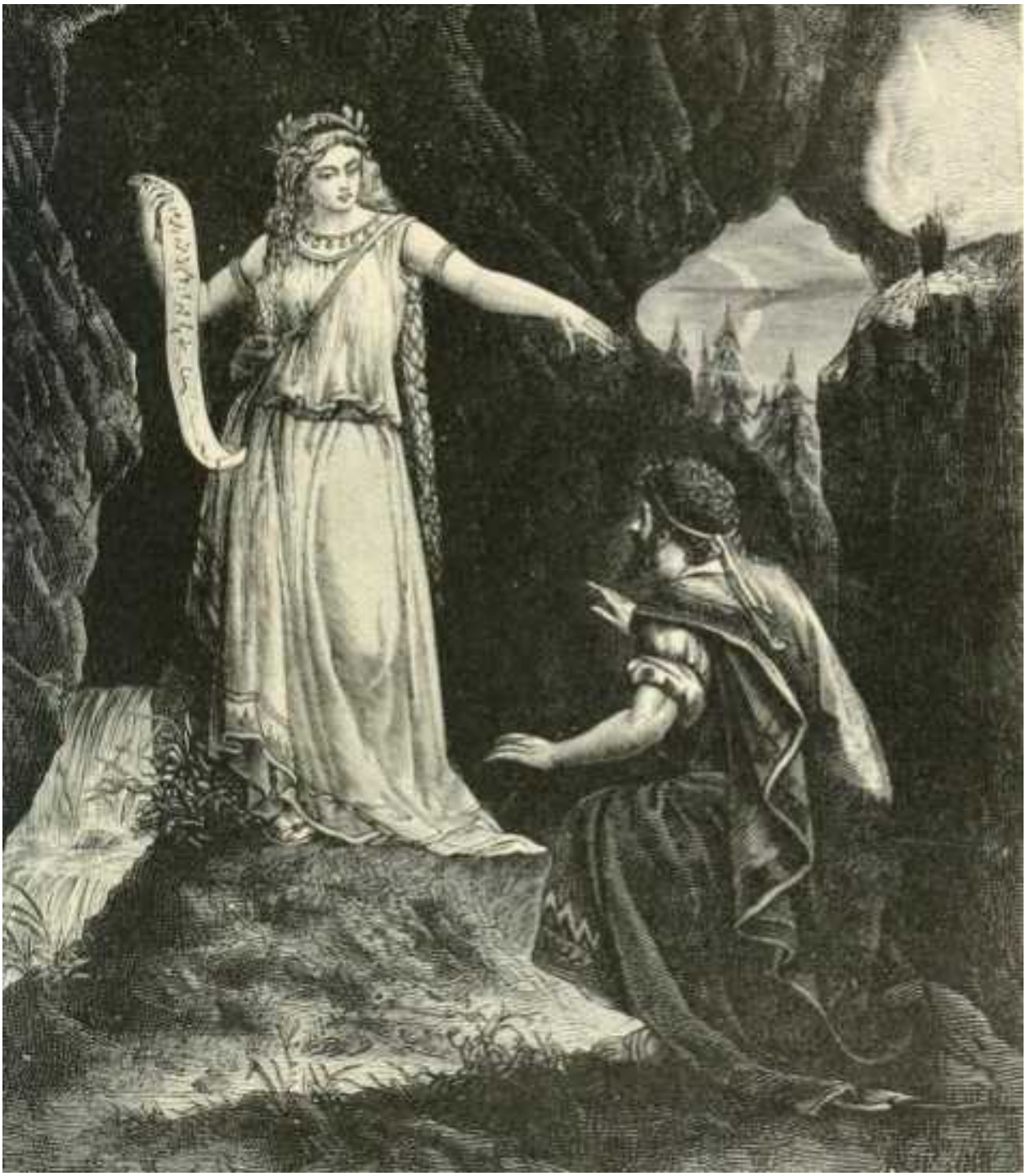

Fig. 1: *Egeria, the SFW nymph, counseling Numa Pompilius*

THIS IS PRE-PRE-PRE-PRE-ALPHA SOFTWARE. DO NOT USE IT.

CHAPTER 2

About

Egeria is a personal, plugin-driven, software automation system. Its configuration file defines a number of tasks, each task being composed of one or more input plugins that fetch data from popular services such as Youtube or RSS feeds, any number of mutator plugins that are capable of modifying or complementing that data and one or more output plugins that can perform actions based on the data that was fetched and modified. Each task has its own memory space (backed by Redis) making it possible to avoid processing the same information more than once.

Egeria is stil in alpha and can only be used with a substantial amount of effort. Writing an Egeria configuration file is challenging and requires a rather advanced level of knowledge of the system. This may change in the future.

CHAPTER 3

Contents

3.1 Overview

One task called `youNotify` contains the following plugins:

- `youtube`, an input plugin configured to monitor a user's subscriptions. 10 videos are fetched from each subscribed channel every 2 hours.
- `filter`, a mutator configured to discard any video that contains the word "live" in the title.
- `deduplicate`, a mutator that has no configuration.
- `pushbullet`, an output plugin configured to send a notification to the user's smartphone every time a new video is found.
- `emailDigest`, an output plugin configured to gather all information and release it every day at 18:30 in a single email, sent to the user's email address.

The data fetched from Youtube is packaged as a series of distinct **facts**. Each fact contains metadata about a single video.

Facts are Egeria's atoms; they can be manipulated and changed, or thrown away, but one atom will always remain one atom. It cannot be divided.

Facts are Knowledge. **Knowledge** is **learned** by input plugins.

Knowledge learned by `youtube` is then processed by the `filter` mutator. Some of that Knowledge will be thrown away.

Whatever Knowledge survives the selection will be handed over to `deduplicate`, which will automatically remove any fact that has been processed successfully in the past.

Since after `deduplicate` there aren't any other mutators, any fact left will be fed to all of the output plugins in the task.

`pushbullet` will process each fact one at a time, acting on it the only way it knows: it will use some of the metadata to tell Pushbullet to send a notification to the user's smartphone. `pushbullet` will not retain any facts; once used, they are lost. If something goes wrong, they are lost *anyway*.

`emailDigest` will instead accumulate Knowledge. Accumulated Knowledge is called **Wisdom**; a unit of Wisdom - which usually contains several facts but can be empty - is called an **experience**.

According to its configuration, every evening at 18:30 `emailDigest` will use its experience to generate an email that will be sent immediately. It will then forget its current experience and start accumulating a new one.

Since the `youtube` plugin always learns about the latest 10 videos from each subscribed channel and most channels will upload only 1 or 2 videos in the entire day, the task will be dealing with a lot of duplicates. The `deduplicate`

mutator makes sure that only new facts will be handed over to the output plugins, but there are limits. Currently, Egeria will remember a fact for 6 months; once a fact is that old, it will simply disappear. Should `youtube` learn about that fact again 7 months later, `deduplicate` will let it through.

If one or more plugins fail, the fact being processed at that time will instead be forgotten entirely, as will any facts that are filtered out on purpose by any plugin (usually mutators). Those facts will find their way into the task again and again, until the input plugins stop learning them.

A task can have any number of plugins of any kind. It can even miss input (or output) plugins entirely but... it won't do much.

3.1.1 Example: a task

Let's go for something a little simpler. This task learns about Youtube videos; it will only retain facts that are at most 2 days old, it will reject duplicates and send notifications via Pushbullet.

```
{
    system: {},
    tasks: [{
        name: 'youNotify',
        chain: [{
            plugin: 'youtube',
            plan: '0 */2 * * *',  //every 2 hours
            configuration: {
                mode: 'subscriptions',
                limit: 10,
                identity: 'gglMyName'
            }
        },{
            plugin: 'filter',
            configuration: {
                'youtube:video:publishedAt': {after: [-2, 'days']}
            }
        },{
            plugin: 'deduplicate'
        },{
            plugin: 'pushbullet',
            configuration: {
                identity: 'pbMyName',
                tag: "some pushbullet tag",
                mode: "link",
                title: "{{youtube:channel:title}} - {{youtube:video:publishedAt!!MM/DD/YYYY}} - {{you
                body: "{{youtube:video:description}}"
            }
        }
    }]
}
```

While all plugins are placed in a chain, only the order of mutators matters; input plugins such as `youtube` will just learn facts whenever they run and output plugins will always receive facts after they have gone through all of the mutators.

On the other hand, the order of mutators does matter; in this example placing the `filter` plugin, which forgets facts based on their own properties, before the `deduplicate` plugin, which hits the database once for every fact, is more sensible than the opposite (which would run more database queries, all of them useless).

3.2 Learning

3.2.1 On tasks

A task can be seen both as a container of plugins or as a container of all the facts that have been learned and processed successfully by those plugins. It can therefore be *controlled* - started (so that it will run on schedule), stopped (so that it won't) or run immediately.

At this point in Egeria's development tasks are just names given to plugin chains. They can be used to address those plugins and they're implicitly used to isolate facts learned by a specific task's plugins from any other fact.

3.2.2 On input plugins

All input plugins are **service-alikes** that can be started, stopped or run. Note: some output plugins behave like services too. Also note that starting, stopping or running an entire task means starting or stopping or running all of the service-like plugins in its chain.

Inputs, regardless of what they do and how they do it, are always configured with a `plan` - a *cron* definition that looks like this: '*/2 * * * *' (this means "every 2 minutes"). When an input plugin is in the `started` state, it will automatically run according to its `plan`. Every time it runs, it learns facts - some of them new, some of them old.

Here's a complete guide of the Cron format.

Facts learned by input plugins are fed to the rest of the plugin chain.

3.2.3 On mutators

Mutators receive a fact from the preceding plugin (an input or another mutator), process it and return it to the chain so that it can be processed by the next plugin in line. Mutators can forget facts, effectively removing them from the chain. Forgotten facts are gone forever - unless an input plugin learns them again.

3.2.4 On output plugins

Outputs make the magic happen: a file is downloaded, a torrent is sent to a BT client, a notification is pushed, a CSV table is created.

A few outputs are service-like; meaning they are also configured with a `plan`. Those plugins usually need to collect a number of facts over time and process them all at specified intervals (sending an email digest, writing a CSV file etc.). Those plugins can also be started, stopped or run just like inputs, but instead of learning facts they will accumulate them over time and then consume and forget them.

3.2.5 On Knowledge

Facts are small collections of metadata. They describe something in reality or on the Internet. Facts won't include a video, but they can tell you where it is and what its name and description are. In a few cases facts do contain all of the data (e.g. Reddit private messages).

Facts are highly expendable; they are usually learned several times by the same plugin and should your Internet connection or electricity fail they still have a high chance of being processed just a little while later, as long as Egeria is kept running at all times, restarting it as needed.

Here's what a `youtube` fact looks like (printed in a somewhat legible format, omitting a few fields):

Field	Value
youtube:video:id:videoId:	eWEJb_3IFBw
youtube:video:publishedAt:	Thu Mar 31 2016 19:00:02 GMT+0200 (W. Europe Daylight Time)
youtube:video:title:	Batman V Superman Discussion (SPOILERS)
youtube:video:description:	From the March 30th stream, we talk about Batman V Superman at length. – Watch live at https://www.twitch.tv/previouslyrecorded_live.
youtube:video:thumbnails:default:url:	https://i.ytimg.com/vi/eWEJb_3IFBw/default.jpg
youtube:video:thumbnails:medium:url:	https://i.ytimg.com/vi/eWEJb_3IFBw/mqdefault.jpg
youtube:video:thumbnails:high:url:	https://i.ytimg.com/vi/eWEJb_3IFBw/hqdefault.jpg
youtube:video:link:	https://www.youtube.com/watch?v=eWEJb_3IFBw
youtube:channel:title:	Previously Recorded
youtube:channel:description:	Red Letter Media presents... Previously Recorded! A channel mainly starring Rich Evans and Jack from RLM doing some gaming action. [...]
youtube:channel:customUrl:	previouslyrecorded
youtube:channel:publishedAt:	Mon Jul 14 2014 03:27:40 GMT+0200 (W. Europe Daylight Time)
youtube:channel:thumbnails:default:url:	https://yt3.ggpht.com/-ibOtuEASBgo/AAAAAAAAAAI/AAAAAAAAAAA/tPqxcB90mPY/s88-c-k-no-rj-c0xffffff/photo.jpg
youtube:channel:thumbnails:medium:url:	https://yt3.ggpht.com/-ibOtuEASBgo/AAAAAAAAAAI/AAAAAAAAAAA/tPqxcB90mPY/s240-c-k-no-rj-c0xffffff/photo.jpg
youtube:channel:thumbnails:high:url:	https://yt3.ggpht.com/-ibOtuEASBgo/AAAAAAAAAAI/AAAAAAAAAAA/tPqxcB90mPY/s240-c-k-no-rj-c0xffffff/photo.jpg

There's a lot you could do with even this little amount of information. Notifications, keeping a log, downloading stuff... The point of Egeria is to automate the systematic consumption of data, letting you spend more time exploring new things. Egeria wants you to stop clicking on the same thing every day to see if there's an update. Go click on entirely new stuff instead.

3.2.6 On Wisdom

Wisdom equals Knowledge plus time. You don't need to know anything else about it.

3.3 The systematic consumption of Knowledge

3.3.1 Willful ignorance

Plugins of all kinds are largely ignorant of each other. `pushbullet` doesn't know how the facts learned by `youtube` look like; in order for it to work, the user has to configure it so that it will get the right values from the right fields and concatenate them to form meaningful messages.

Similarly, `transmission` doesn't know where the link to a torrent will be, nor will it search for it. `rarbgLookup` won't ever try to guess how to put together a meaningful search term.

This is by design.

3.3.2 Templating

In order to generate messages, links, file paths and anything else that might be needed to make plugins work together, templates need to be written. Here's what a template looks like:

'The field "youtube:video:link" contains the value {{youtube:video:link}}'

If this template is applied to the fact shown in the previous chapter, the result will be this:

'The field "youtube:video:link" contains the value https://www.youtube.com/watch?v=eWEJb_3IFBw'

The full docs of the templating system are on the `egeria-temple` project's page, but here's the part you need if all you care about is learning how to write templates.

A template is a single string that contains zero or more tokens. Text present between tokens won't be changed. "Rendering" a template means replacing all tokens in it with a string of characters.

In a template, "||" and "!!" are reserved sequences. "|||", "||||" and so on are all illegal. The same goes for "!!" and "!!!", "!!!!" and so on. It's OK if those sequences *are part of the value that will be rendered*, but they must not appear in the template itself.

A token is written like this: `{{PRE||FIELD!!FORMAT||POST}}`

The order of the various elements must be respected:

`PRE` is a string that will precede the value if the value is not null.

`FIELD` is a field of the fact being processed and resolves to wahtever value is present in that field. The value could be null.

`FORMAT` is the format that will be used to format the `FIELD` if and only if such value is a Date. It will be ignored otherwise.

`POST` is a string that will be appended to the value if the value is not null.

`PRE`, `FORMAT` and `POST` are optional. If the value is a Date, the default `FORMAT` is 'YYYYMMDD' (e.g. "20151124").

`FIELD` is mandatory, but it doesn't have to exist on `data`. If it's null, undefined or empty, the entire token will be replaced by an empty string, meaning that `PRE` and `POST` **will not be rendered either**.

Malformed tokens will remain untouched in the final output.

The smallest token you can possibly write is `{{fieldName}}`. The biggest is `{{On day ||happeningDate!!DD-MM-YYYY|| something happened}}`.

Now let's look again at the `pushbullet` configuration from the first example that appears in this guide:

```
{
    plugin: 'pushbullet',
    configuration: {
        identity: 'pbMyName',
        tag: "some pushbullet tag",
        mode: "link",
        title: "{{youtube:channel:title}} - {{youtube:video:publishedAt!!MM/DD/YYYY}} - {{youtube:vi
        body: "{{youtube:video:description}}"
    }
}
```

The parameters `title` and `body` will apply their templates to any fact sent to `pushbullet`. The results, if the fact is the one learned by `youtube` that we saw earlier, will be this:

// value generated by the template assigned to "title":

'Previously Recorded - 03/31/2016 - Batman V Superman Discussion (SPOILERS)'

// value generated by the template assigned to "body":

'From the March 30th stream, we talk about Batman V Superman at length. – Watch live at https://www.twitch.tv/previouslyrecorded_live.'

To know how to configure plugins properly you can refer to their documentation. It will tell you what you can configure, which parameter will accept templates and what the resulting value will be used for. The documentation of plugins that can learn new facts or add to them will tell you what fields will be available and what they contain.

Remember that something like `'This is a template'` is **also** a valid template; it will always output `'This is a template'` regardless of what fact it's applied to. Think of it as a constant. Moreover, `''` is also a valid template that will always output an empty string.

3.3.3 Private information

A number of services require their users to authenticate before they let them access the data. Egeria has a simple system by which it can learn and remember how to authenticate with various services on behalf of the user. This is set up through `egeriactl auth <type> [credentials...]`.

When setting up authentication, always run `egeriactl` from a console within a graphical environment with a working browser. All OSes are supported. This is necessary because of how modern authentication works.

A few services such as Google may require you to create an API app. This documentation will help you do that.

3.3.4 How plugins can be addressed: plugin paths

Once the configuration is complete, a way is needed to address different plugins or different tasks. For each instance of a plugin, a path is assigned:

> <task name>/<plugin name>/<instance number>

In the following example:

```
{   system: {},
    tasks: [{
        name: 'test',
        chain: [{
            plugin: 'youtube',
            configuration: {...}
        }, {
            plugin: 'youtube',
            configuration: {...}
        }, {
            plugin: 'filter',
            configuration: {...}
        }, {
            plugin: 'youtubeDownloader',
            configuration: {...}
        }],
    }]
}
```

The paths are:

- test/youtube/0
- test/youtube/1
- test/filter/0
- test/youtubeDownloader/0

The instance number always respects the order set by the user. yuotube/0 will always be the first `youtube` plugin that appears inside the task configuration and youtube/1 will always be the second.

3.4 Controlling the flow of Knowledge

Facts would normally be routed through all of the mutators and finally handed to the output plugins. This is highly restrictive and would force users to write multiple tasks with different filters to handle slight differences in use cases. Let's not do that.

3.4.1 Tagging facts based on the input plugins that learned them

All plugins accept a configuration option called `tags`. This option is always an Array of one or more Strings, each one a "tag".

```
{   system: {},
    tasks: [{
        name: 'test',
        chain: [{
            plugin: 'youtube',
            plan: '0 0 * * *',
            tags: ['subs'],
            configuration: {
                mode: 'submissions',
                limit: 5
            }
        }, {
            plugin: 'reddit',
            plan: '0 0 * * *',
            tags: ['movies'],
            configuration: {
                subreddit: 'fullmoviesonyoutube',
                mode: 'new',
                limit: 5
            }
        }]
    }]
}
```

Every fact learned by `test/youtube/0` will be tagged with `subs`, while every fact learned by `test/reddit/0` will be tagged `movies`.

Mutators will interact with all of those facts **if the mutator hasn't been given any tag**; or, if the mutator has been tagged, only with facts that share **at least one tag** with it.

Let's discard any `subs` older than 2 days and any `movies` that don't have "1080p" in the title:

```
{   system: {},
    tasks: [{
        name: 'test',
        chain: [{
            plugin: 'youtube',
            plan: '0 */12 * * *',
            tags: ['subs'],
            configuration: {
                mode: 'submissions',
                limit: 5
            }
        }, {
            plugin: 'reddit',
            plan: '0 0 * * *',
```

```
            tags: ['movies'],
            configuration: {
                subreddit: 'fullmoviesonyoutube',
                mode: 'new',
                limit: 5
            }
        }, {
            plugin: 'filter',
            tags: ['subs'],
            configuration: {
                'youtube:video:publishedAt': { after: [-2, 'days'] }
            }
        }, {
            plugin: 'filter',
            tags: ['movies'],
            configuration: {
                'reddit:title': { matches: /1080p/i }
            }
        }],
    }]
}
```

3.4.2 Tagging facts based on their own properties

Using the `tagger` mutator, it's possible to virtually split facts into different groups by tagging them differently based on their own characteristics.

More in-depth documentation of this mutator is available in its specific section.

```
{
    plugin: 'tagger',
    configuration: {
        schemas: [{
            'youtube:video:publishedAt': {after: [-7 days], mandatory: true}
        }, {
            'youtube:video:publishedAt': {before: [-7 days], mandatory: true}
        }],
        schemaTags: [
            'recent',
            'old'
        ]
    }
}
```

3.5 Configuration

Before you can do anything, you need to set up a couple of things.

First of all, you need a Redis (Windows x64 port) instance running. Egeria doesn't require you to modify the default configuration in order to work, but if you want to learn about Redis knock yourself out. All Egeria cares about is the host running Redis and the port Redis is listening on. They go in `redisHost` and `redisPort`.

Egeria will itself listen on a port of your choosing for connections from its commandline interface (egeria-ctl). That number goes in `port`.

Choose a log level; I recommend `info` but if you want to take a peek inside you can use `debug`. This goes in `logLevel`.

```
{
    "system": {
        "redisHost": "localhost",
        "redisPort": 6379,
        "port": 54321,
        "logLevel": "debug"
    },
    "tasks": []
}
```

Save this configuration as a JSON file anywhere you want. This is called the "null configuration" - because it contains no tasks - but it's enough to get Egeria running. Your next step is setting up authentication for all the services you wish Egeria to use on your behalf.

3.5.1 Authentication setup

TODO

You can now configure tasks and plugins.

3.5.2 Tasks

A task only has a `name` and a `chain` of plugins. The `chain` can be empty:

```
{
    "system": {
        "redisHost": "localhost",
        "redisPort": 6379,
        "port": 54321,
        "logLevel": "debug"
    },
    "tasks": [{
        "name": "test",
        "chain": []
    }]
}
```

If you actually want the task to do something, you need to put plugins in its `chain`. While only the order of mutators matters, it's best to sort the chain putting all the input plugins at the beginning, then all of the mutators in the right order, and finally all of the output plugins.

3.6 Input plugins catalog

3.6.1 concerts

Choose any number of bands to monitor and `concerts` will learn a fact for each date.

Configuration

Option	Description
bands	Specify a list of bands in Array form or a single band as a String.

Output

Field	Type	Notes	Description / Value
concert:image	URL	Full address	
concert:author	String		This is just th band's name
concert:link	URL	Full address	A link to the concert's page
concert:title	String		Name of the concert, often it's the band name again
concert:date	Date		
concert:venue	String		Name of the venue
concert:location	String		Address of the venue, often it's just the city

3.6.2 fileList

Generate a fact about each of the files inside a directory. Can recurse into subfolders.

Configuration

Option	Type	Description
path	String	A valid, existing file path. It won't be created for you. The plugin will generate one fact for each file found there.
recurse	Boolean	Recurse into subdirectories. Defaults to *false*.

Output

Field	Type	Description / Value
file:fullPath	String	/home/username/somefile.txt
file:name	String	somefile.txt
file:dev	Number	ID of the device containing the file
file:mode	Number	Protection bits
file:nlink	Number	Number of hard links pointing to this file
file:uid	Number	Id of the owner
file:gid	Number	Group id of the owner
file:rdev	Number	Device id (if the file is special)
file:size	Number	File size in bytes
file:blocks	Number	Blocks occupied
file:blksize	Number	Size in bytes of each block
file:atime	Date	Last access time
file:mtime	Date	Last modification time
file:ctime	Date	Last inode modification time
file:birthtime	Date	Not always supported, file creation time

3.6.3 fileRead

Generate a fact about each of the rows of a CSV file. The CSV format needs to be compatible with the CSVs generated with fileWrite. The delimiter must be ';' and both strings and empty fields must be quoted. The first line of the file must contain the column names.

Configuration

Option	Type	Description
path	String	A valid path leading to a valid CSV file.

Output

Fields available in the output depend on the contents of the CSV. The CSV header determines the name of each field.

3.6.4 rarbgSearch

Perform a search on rarbg.to.

Configuration

Option	Type	Description
categories	Array	Restrict the search to one or more categories.
search	Array	Every object in this array describes a search criteria. Every criteria is described by a `type` ('query', 'imdb', 'tvdb' or 'tmdb') and a term.
sort	String	Choose how results will be sorted. Defaults to 'seeders' (most seeded first); you can also choose 'leechers' (most leeched first) or 'last' (most recently added first).
minSeeders	Integer	Minimum number of seeders. Torrents seeded less than this will be discarded. Must be greater than zero.
minLeechers	Integer	Minimum number of leechers. Torrents leeched less than this will be discarded. Must be greater than zero.
ranked	Boolean	If true, restrict results to scene releases.
quality	Object	An object describing which level of quality is acceptable. The audioCodec field is a list of acceptable codecs, from most to least preferred. videoCodec , videoResolution and videoSource work in the same way.

Available categories:

- All movies
- All TV shows
- XXX
- Movies/XVID
- Movies/XVID/720
- Movies/x264
- Movies/x264/1080

- Movies/x264/720
- Movies/x264/3D
- Movies/Full BD
- Movies/BD remux
- TV Episodes
- TV HD Episodes
- Music/MP3
- Music/FLAC
- Games/PC ISO
- Games/PC RIP
- Games/PS3
- Games/XBOX-360
- Software/PC ISO
- e-Books

Search types:

Type	Term	Effect
query	An arbitrary String	Restricts the results to those containing the text provided
imdb	Any valid http://imdb.com ID	Restrict the results to tv shows or movies matched by the ID
tvdb	Any valid http://thetvdb.com ID	Same as above, different website
tmdb	Any valid http://www.themoviedb.org ID	Same as above, different website

Example:

```
{
    plugin: 'rarbgSearch',
    configuration: {
        search: [
            {type: 'query', term: 'S01'},
            {type: 'imdb', term: 'tt123456ABCD'}
        ]
    }
}
```

You can find a list of supported qualities on the `egeria-divine` project page.

Example:

```
{
    plugin: 'rarbgSearch',
    configuration: {
        search: [{type: 'query', term: 'Something perfectly legal'}],
        quality: {
            videoResolution: ['1080p', '720p'],
            videoCodec: ['h265', 'h264']
        }
    }
}
```

Output

Field	Type	Notes	Description / Value
rarbg:title	String		
rarbg:category	String		
rarbg:download	URL	Magnet link	Torrent link
rarbg:seeders	Number		
rarbg:leechers	Number		
rarbg:size	Number	bytes	
rarbg:pubdate	Date		
rarbg:episode_info:imdb	String		imdb ID
rarbg:episode_info:tvdb	String		tvdb ID
rarbg:episode_info:themoviedb	String		tmdb ID
rarbg:ranked	Number	0 or 1	Is it a scene release? 1 = true
rarbg:videoResolution	String		
rarbg:videoSource	String		
rarbg:videoCodec	String		
rarbg:audioCodec	String		

3.6.5 reddit

Fetch submissions or private messages from Reddit.

Configuration

Warning: markAsRead currently has no effect. This is a bug.

Choose a `mode` between `new` (get the latest `limit` submissions), `hot` (get the `limit` hottest submissions - set `limit` to 25 to fetch the entire front page) or `messages` (ignore the chosen subreddit and get the latest `limit` private messages instead).

In `messages` mode a Reddit `identity` is mandatory. Set it up with `egeriactl`. It doesn't make any difference for the other modes.

Option	Type	Description
mode	String	Mandatory. One of `new`, `hot` or `messages`.
subreddit	String	Mandatory unless mode is `messages`. The name of the subreddit without the `/r/`, e.g. `all` or `funny`.
limit	Number	How many submissions or messages should be fetched every time the plugin runs.
markAsRead	Boolean	Only effective if the mode is `messages`, defaults to false. Mark fetched messages as read.
identity	String	Mandatory for mode `messages`. Must be an Egeria identity of type "reddit".

Output

Field	Type	Notes	Description / Value
reddit:subreddit	String		The subreddit where this link was submitted
reddit:created	Date		Tells you when the link or message was submitted.
reddit:over_18	Boolean		Is the content NSFW? This may mean different things depending on the subreddit, e.g. it could identify spoilers.
reddit:stickied	Boolean		Is this a sticky?
reddit:is_self	Boolean		Is it a self submission (i.e. not an external link)?
reddit:title	String		Submission title.
reddit:thumbnail	URL	Full address, unless nsfw	If the submission is NSFW this field will contain the word "nsfw"; otherwise, it's a link to the submission thumbnail.
reddit:url	URL	Full address	Submitted link.
reddit:domain	String		Domain of the submitted link (e.g. "youtube.com" for Youtube videos)
reddit:link_flair_text	String		Submission flair.
reddit:selftext_html	String		For text submissions, this contains the HTML version of the submission body.
reddit:selftext	String		For text submissions, this contains the plaintext version of the submission body.
reddit:author	String		Username of the submitter.
reddit:author_flair_text	String		Submitter's flair.
reddit:score	String		Karma! Delicious karma! This is the total.
reddit:ups	Number		Number of upvotes.
reddit:downs	Number		Number of downvotes.
reddit:gilded	Number		The number of times it was gilded.
reddit:locked	Boolean		Was the discussion locked?
reddit:archived	Boolean		Is the submission archived?
reddit:edited	Boolean		Was the submission edited?
reddit:hidden	Boolean		Did you hide this submission? Only makes sense if you're using an identity.
reddit:permalink	URL	Full address.	Permalink to the comments page - this is not the submitted link.
reddit:num_comments	Number		Number of comments
reddit:id	String		Unique ID used by Reddit to identify this submission/comment
reddit:subreddit_id	String		Unique ID of the subreddit.

3.6.6 rss

Fetch an RSS feed and learn one fact for each item.

Configuration

The RSS feed is always fetched in its entirety.

Option	Type	Description
url	String	Mandatory. The feed address.

Output

RSS feeds vary wildly because of how malleable the underlying language is, but the following fields are usually available:

Field	Type	Notes	Description / Value
rss:title	String		Resource title
rss:description	String		Description of the content
rss:summary	String		Some sort of summary, like the first couple of paragraphs of an article
rss:date	Date		When the resource was published
rss:link	URL	Full address	Link to the resource
rss:author	String		
rss:categories	String		Categories; multiple categories are joined with commas

3.6.7 youtube

Requires an Egeria identity of type `google` to work. Set it up with:

```
>>> egeriactl auth google <id> <secret>
```

Configuration

Option	Type	Description
mode	String	Mandatory. Choose between "subscriptions" (monitor your subs), "playlists" (monitor one or more playlists), "search" (monitor the results of one or more searches), "channels" (monitor one or more channels).
limit	Integer	Mandatory. How many videos to fetch from each subscribed channel or playlist or search result or channel.
ids	Array	Mandatory for all modes except "subscriptions". Must be one or more playlist ids or search terms or channel names
sort	String	Defaults to 'date'; can be 'relevance'. Decide if you want the latest videos first, or the most relevant.
identity	String	Mandatory. Must be an Egeria identity of type "google".

Upstream API docs

Yotube Data API v3

Output: `subscriptions`, `channels` and `search` mode

Field	Type	Notes	Description / Value
youtube:channel:publishedAt	Date		When the channel was created
youtube:channel:title	String		
youtube:channel:description	String		
youtube:channel:thumbnails:default:url	URL	Full address	
youtube:channel:thumbnails:medium:url	URL	Full address	
youtube:channel:thumbnails:high:url	URL	Full address	
youtube:video:id	String		Unique video ID, part of the video's page URL
youtube:video:publishedAt	Date		When the video was uploaded to the channel
youtube:video:title	String		
youtube:video:description	String		
youtube:video:thumbnails:default:url	URL	Full address	
youtube:video:thumbnails:medium:url	URL	Full address	
youtube:video:thumbnails:high:url	URL	Full address	
youtube:video:link	URL	Full address	Link to the video page

Output: `playlists` mode

Everything present in the output of the other modes, plus:

Field	Type	Notes	Description / Value
youtube:playlist:publishedAt	Date		When the playlist was created
youtube:playlist:title	String		
youtube:playlist:description	String		
youtube:playlist:thumbnails:default:url	URL	Full address	
youtube:playlist:thumbnails:medium:url	URL	Full address	
youtube:playlist:thumbnails:high:url	URL	Full address	
youtube:playlist:thumbnails:standard:url	URL	Full address	
youtube:playlist:thumbnails:maxres:url	URL	Full address	

Examples

Fetch 4 videos from each subscribed channel every 12 hours:

```
"tasks": [{
    "name": "test",
    "chain": [{
        "plugin": "youtube",
        "configuration": {
            "identity": "gglMyName",
            "mode": "subscriptions",
            "limit": 4,
            "plan": "0 */12 * * *"
```

```
        }
    }, {
        "plugin": "inspect"
    }]
}]
```

Monitor two playlists, fetching 4 videos from each one every 24 hours at midnight:

```
"tasks": [{
    "name": "test",
    "chain": [{
        "plugin": "youtube",
        "configuration": {
            "identity": "gglMyName",
            "mode": "playlists",
            "ids": [
                "PLAbMhAYRuCUhawCEV2oXZGrienoKTN16X",
                "PLFPEDTXyQKoPZvHSBdoRLJAlUqdWVVgYT"
            ],
            "limit": 4,
            "plan": "0 0 * * *"
        }
    }, {
        "plugin": "inspect"
    }]
}]
```

Every day at midgnight get the 6 most relevant results of searching for "DOOM trailer":

```
"tasks": [{
    "name": "test",
    "chain": [{
        "plugin": "youtube",
        "configuration": {
            "identity": "gglMyName",
            "mode": "search",
            "ids": ["DOOM trailer"],
            "limit": 6,
            "sort": "relevance",
            "plan": "0 0 * * *"
        }
    }, {
        "plugin": "inspect"
    }]
}]
```

3.7 Mutator plugins catalog

TODO

3.8 Output plugins catalog

TODO