G.A.I.A.
User Manual

v1.0.0

D.A.M.O.F.

September 2022

Abstract

This user manual describe how to use G.A.I.A. and more in details how to control the behaviour using configuration file.

Contents

Document Management

Author	FDA	16/09/2022
Issued by		
Revised by		
Approved by		

Document Status Sheet

Issue	Date	Comment
1.0	23/09/2022	First Release

Document Change Record

Issue	Reason for change	Paragraph	Type of Modification

Introduction

GTFS Accurate Insight Analyser, in short G.A.I.A., is a tool intended to help all professionists working with General Transit Feed Specification. Transportation Agencies, Delivery, Systems Integrators, Developers ... all of them can take an advantage from using G.A.I.A. !

Purpose

All applications that are GTFS based usually are also able to validate provided inputs in order to check all specific application constraints, and this is perfectly fine since external validators can only partially cover applications requirements and also is a good best practice, but what if inputs preparation is not in charge of developers, and then responsibility for such input is in charge to a different team or in charge to the final users? ... these circumstances are really common and in most of the cases applications access is restricted to specialized workers only, creating the needs to have continuous exchanges between who is in charge of producing "GTFS feeds" and who is in charge to validate those feeds with/for the target application.

Scope

Let's imagine to have something the middle, enough detailed to cover all specific application constraints and enough simple to be used from anyone, without the needs to have specific knowledge or preparation! Here we are ... this is where G.A.I.A. is coming to help! Starting from specification, will be easy for anyone of the stakeholders, usually developers or functional engineers, to prepare a configuration file that describe all application constraints, then such configuration file can be shared with developers as well as with other teams and with the customer if needed. Once we have a configuration file, it will be possible to detect all "anomalies" in the GTFS feeds using G.A.I.A. and then avoiding issues with the specific application. In addiction to standards GTFS files, it is possible to handle also extra CSV files that may be required from the application, as well as to provide a default content if such files are missing. Moreover, leveraging G.A.I.A. configuration will be possible also to automatically fix different anomalies.

References

[1] `https://developers.google.com/transit/gtfs`
[2] `https://en.wikipedia.org/wiki/List_of_tz_database_time_zones`
[3] `https://github.com/fe-dagostino/libcsv`

Document Overview

This manual is intended for GAIA users that would like to customise their own *gtfs.cfg* in order to match with their specific application requirements.

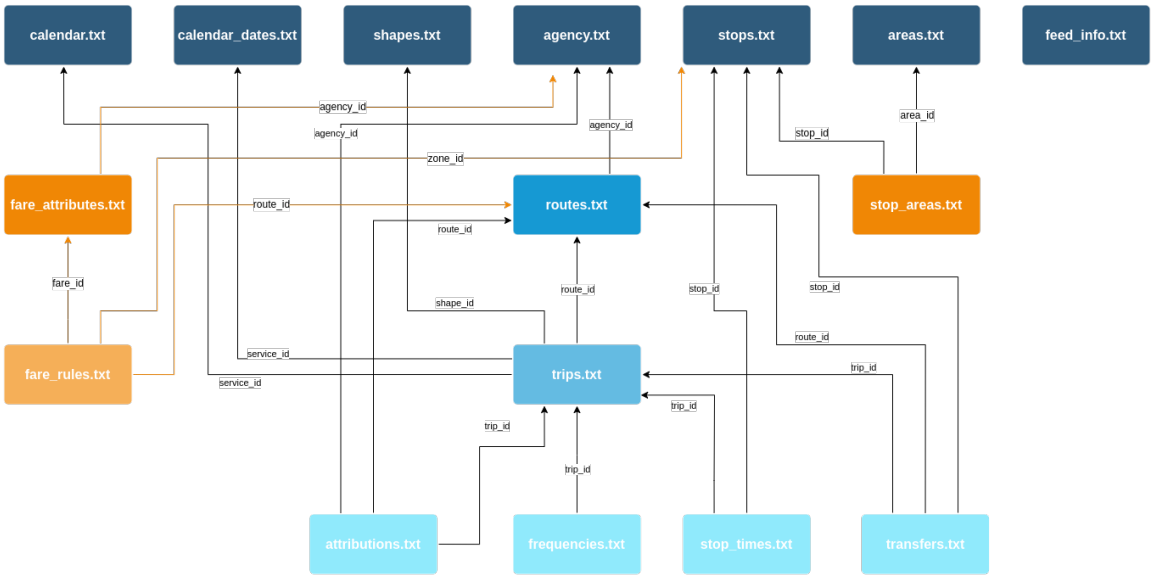

Figure 1: GTFS Dependencies

The above figure highlight dependencies between GTFS feeds foreseen in the standard [1] and will be a helpful reference in this manual.

GAIA Configuration

There are two configuration files that are required by the application in order to work properly:

- **gaia.cfg**: with all global settings
- **gtfs.cfg**: to be customised accordingly with data set and application needs.

Settings in gaia.cfg

This configuration file contains all global settings such as directories to be used and settings for both logging and parser. Let's see all of this sections in details starting from ***global***.

```
global = {
  src_dir="gtfs_in/";
  dst_dir="gtfs_out/";
  feed_extension=".txt"

  logging = { ... };
  parser  = { ... };
}
```

- **src_dir**: specify *source directory* where to find original ***GTFS feeds***.
- **dst_dir**: specify *destination directory* used to write GTFS output feeds once all filters have been applied.
- **feed_extension**: file extension type for the ***GTFS feeds***.
- **logging**: subsection containing general logging general settings.
- **parser**: subsection containing general parser general settings.

global.logging

In this subsection we can find all global settings for the logger.

```
logging = {
  log_directory="log/";
  log_filename="_gaia.log";
  log_enable_console=true
};
```

- **log_directory**: specify the directory where to write outputl logs. If specified directory does not exists it will be created.
- **log_filename**: suffix appended to each log filename, more in general the application will generate a new log file at each run and format used is **YYYYmmGGHHMMSS**, so final log filename will have the following form *20221120104006_gaia.log* where we have information about the date **2022/11/20** and time **10:40:06** in which log have been generated.
- **log_enable_console**: enable or disable log on standard output.

global.parser

In this subsection are listed settings for the csv parser, such information will be used to initialize ***libcsv***.

```
parser = {
  delimeter         = ",";
  quote             = "\"";
  eol               = "\n";
  comment           = "#";
  whitespaces       = "\a\b\t\v\f\r\n";
  skip_whitespaces  = true;
  trim_all          = true;
  allow_comments    = false;
};
```

- **delimeter**: characther used as *field delimeter* when parsing *csv row*
- **quote**: characther used as *quote delimeter* when parsing the single *field* in a *row*
- **eol**: End Of Line character used to identify when a *row* has ended
- **comment**: *csv* standard do not make any mention for remarks inside the *csv* itself, anyway the parser have this capability.
- **whitespaces**: define the list of whitespaces
- **skip_whitespaces** = *enable/disable* skip whitespaces. When true all whitespaces will be not anymore present in the *output*.
- **trim_all**: *enable/disable trim all*. When enabled all spaces at before and after a field will be removed unless the field is quoted.
- **allow_comments**: *enable/disable* comments inside the *csv*.

Settings in gtfs.cfg

While *gaia.cfg* have a fixed number of subsections, this is not the case for *gtfs.cfg* in which we can have a variable number of subsection describing feeds, dependencies and filters to be applied. This file have been organized in two main parts:

- **feeds**: containing the exhaustive list of all *csv* files that we want to process. Here we shall list both standard GTFS files as well as extra files not included in the standard that we want to process with GAIA, assuming that those files are also in *csv* format.
  Note: that files not listed here will be not included in the output.
- **feeds.name**: we should have as many as we have listed in **feeds**

Let's see an example in order to provide details for all parameters and to clarify usage.

```
gtfs =
{
  feeds = (
            {
              name          = "agency"
              filename      = "agency.txt";
              presence      = "required";
              dependencies  = [];
            },
            {
              name          = "routes";
              filename      = "routes.txt";
              presence      = "required";
              dependencies  = ["agency"];
            },
            {
              name          = "trips";
              filename      = "trips.txt";
              presence      = "required";
              dependencies  = ["calendar","calendar_dates"];
            },
..
...
```

In the above example we have 3 feeds listed and for each one of them have been provided the following information:

- **name**: this is the name we will use to specify rules on this feed;
- **filename**: used to specify filename for the feed;
- **presence**: determine if such file must be present *"required"*, is required under certain conditions *"conditional"* or if not essential and then *"optional"*. Such field have been set accordingly with the GTFS specification;
- **dependencies**: the list of feeds to be processed before to start processing current feed. In the above, *"agency"* do not have any dependency, *"routes"* will be processed only after *"agency"* and *"trips"* will be processed only after both *"calendar"* and *"calendar_dates"*.

Note: feeds subsection have been already populated and comprises all files in *Figure 1* with related dependencies, so in general, starting from a default *gtfs.cfg* users should remove all files not needed and eventually add extra files if any.

Since we listed all *feeds* required by the application, now let's check what we have to specify for each *feed*. Again the best way to start is from default *gtfs.cfg*, removing all unnecessary field and updating rules as for our needs.

Moving forward with the example, just we just take agency.txt in order to explain how GAIA works.

```
// agency.txt
 agency = {
   // Mandatory fields required for the input feed
   mandatory-fields = ["agency_id","agency_name","agency_url",
                       "agency_timezone"];
   output-header    = ["agency_id","agency_name","agency_url",
                       "agency_timezone","agency_lang",
                       "agency_phone","agency_fare_url",
                       "agency_email"];
   filter-fields    = ["agency_id","agency_name","agency_url",
                       "agency_timezone"];
   //default-content  = ( [], [] );

   ...
   ....

 };
```

As described before each subsection start with the name of the feed we listed in the first subsection "feeds", so here in our example we have "agency = {..." and then all details:

- **mandatory-fields**: the list of all mandatory field that expected in the agency.txt provided as input.
- **output-header**: the list of all fields for agency.txt once processed. Fields will be exported in the same order as listed in here, for many applications this is not important but for some use cases it is a constraint to have a specific order.
- **filter-fields**: here the list of fields that GAIA should process. Listed order will match with processing order and this is something to keep in mind if we have filters that use other fields values to build the content. For each single field listed in this GAIA will expect to find the next level of details that correspond to which filters to apply and in which sequence.
- **default-content**: if specified file (in the above example agency.txt) do not exist in the input set, then GAIA will create such file with a default content. Could be possible to specify more than one row.

It is foreseen one more optional section here that allow to enable specific functionalities most of the time related to dependencies between feeds. Agency do not have any dependecy, so in this specific case we don't have any function associated to this specific feed, but there are other feeds in which this section can be really important, files such as *trips.txt* in which we have many dependencies from other files.

Here how the optional "*processing*" looks like:

```
processing = {
    enable = [ "find_orphan_routes", "find_orphan_services",
               "find_orphan_shapes", "delete_unused_shapes",
               "optimize_shapes_usage" ];

    //  Raise a log message when for the specific trip is not
    //  possible to retrieve route information,
    //  since specified route_id do not exist in routes.
    find_orphan_routes = {
      level  = "error";
      params = [];
    };
    ..
    ...
}
```

The "*processing*" section allow us to enable specific functionalities for the csv files we are currently processing. It is important to know that each *csv* files is subject to two different phases:

- **parsing**: operations performed during parsing. More in details as soon as a new *row* have been parsed, GAIA check for filters chains and apply all of them to such row.
- **post-parsing**: starting from the assumption that rows in a *csv* files can be disordered, then we have some operations that can be performed only once we have *csv* file and its dependencies fully parsed.

All functionalities available in post-parsing will be detailed forward in the this document.

Coming back to "agency.txt" example, let's complete to describe how it works describing *filter_chains* and *filters*.

```
// agency.txt
 agency = {
    ..
    ...
    filter-fields     = ["agency_id","agency_name","agency_url",
                         "agency_timezone"];

    agency_id = {
      filters_chain= ("empty","length","unique");
      empty = {
        level  = "error";
        stop   = true;        // do not process next filter
        params = ["false"];
      };
      length = {
        level  = "error";
        stop   = false;
        params = ["-1","30"];
      };
      unique = {
        level  = "error";
        stop   = false;
        params = [];
      };
    };
```

```
    agency_name = {
      filters_chain= ("empty","length");
      empty = {
        level  = "error";
        stop   = true;      // do not process next filter
        params = ["false"];
      };
      length = {
        level  = "error";
        stop   = false;
        params = ["1","50"];
      };
    };

    agency_url = {
      filters_chain= ("empty","length");
      empty = {
        level  = "error";
        stop   = true;      // do not process next filter
        params = ["false"];
      };
      length = {
        level  = "error";
        stop   = false;
        params = ["7","1000"];
      };
    };

    // Valid Time Zones are listed at the following url.
    // https://en.wikipedia.org/wiki
    // /List_of_tz_database_time_zones
    agency_timezone = {
      filters_chain= ("empty");
      empty = {
        level  = "error";
        stop   = true;      // do not process next filter
        params = ["false"];
      };
    };

  };
```

Accordingly with *filter-fields* GAIA is expecting to find out details for all four field listed here and more in details are required: - a subsection with the name of the field - the list of filters to be applied to such field - filters' parameters So, let's take an example providing details on each part and then we will have all the basis to customise our *gtfs.cfg*.

From the about let's take the first field "agency_id":

```
// agency.txt
 agency = {
    ..
    ...
    filter-fields     = ["agency_id","agency_name","agency_url",
                         "agency_timezone"];

    agency_id = {
      filters_chain= ("empty","length","unique");
      empty = {
        level  = "error";
        stop   = true;        // do not process next filter
        params = ["false"];
      };
      length = {
        level  = "error";
        stop   = false;
        params = ["-1","30"];
      };
      unique = {
        level  = "error";
        stop   = false;
        params = [];
      };
    };
```

The subsection in *agency* start with the the name of the field *agency_id* and contains the following elements:

- **filters_chain**: the list of *filters* to be applied to each value under *agency_id* in *agency.txt*. **Note**: that filters will be applied from left to right, so in this example "*empty*" will be the first filter, "*length*" the second and "*unique*" the third. Moreover, the same filter can be repeated and this can be useful in case we make some composition.

- ***filter***: for each *filter* listed in *filters_chain* we will have one more subsection with its configuration:
  - **level**: can be one of the following values "*debug*", "*info*", "*warning*", "*error*", "*raw*". Values do not affect the behaviours, but it is useful in the final report or for automatic system. This is completely in charge to the user to classify the severity level for the specific check. In this case since "*agency_id*" is a *key* element for *agency.txt* it have been classified as *error*. If there is an automatic system reading the results, then can stop the process in presence of errors and send email or notifications in order to fix it.
  - **stop**: can be *true* that means to do not proceed with other filters since no make sense to check other rules on a field that already have such error; *false*, it means continue anyway even in cases of failure continue with next filter. Just note that each failure will generate log information and in some cases can be convenient to keep logs with necessary information. In the above example, we are telling to GAIA to stop executing next filter in case the field is *empty* since no make sense to check length on an empty field or to check if such field is unique.
  - **params**: here will be possible to provide a parameters list for the filter. Each filter has its own parameters or no parameters at all.

Filters

Filters are organized by version in order to know since which version a specific filter is available.

Version 0.2.0

empty

Check emptiness for the specified field.

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: [*”true”* | *”false”*]
    - **true**: expected that such field will be empty, then if populated with any information filter will fail
    - **false**: expected that such field will be not empty, then if no data will be found filter will fail

length

Check if specified field length is in the specified range of (min,max).

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 2 (two) parameters respectively ”*min*” and ”*max*”
  - 1° parameter: [”-1” | ”*min value*”]
    - **”-1”**: min value will be ignore
    - ***value***: expected minimum number of characters for this field
  - 2° parameter: [”-1” | ”*max value*”]
    - **”-1”**: max value will be ignore
    - ***value***: expected maxmum number of characters for this field

swap

Swap two items, supposing A and B the two values, all occurrence of 'A' will be replaced with 'B' and all occurrence of 'B' with replaced with 'A'.

- **level**: [" *debug*" | " *info*" | " *warning*" | " *error*" | " *raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected 2 (two) parameters respectively " *value A*" and " *value B*"
  - 1° parameter: " *A*"
  - 2° parameter: " *B*"

replace

An extension to *swap function* for multiple replacements. Specify a list of replacements for specific field 'A' will be replaced with 'B', 'C' will be replaced with 'D' and so on ... Moreover is also possible to specify a default value will be addressed with '*' that, if present, will replace all other occurrence with a default value.

- **level**: [" *debug*" | " *info*" | " *warning*" | " *error*" | " *raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected N parameters
  - 1° parameter: ["A|B"]
  - 2° parameter: ["C|D"]
  - N° parameter: ["*|E"]

substring

Specify a list of replacement for specific field. In a field containing '**a**', this last value will be replaced with '**b**'.
Eg. For *block_id* we can specify the following two paramenters "/|.", "\|--", then supposing to have a *block_id* defined as "T100/T200" then it will be modified in "T100**.**T200", also if *block_id* contains "T100\T200" it will be modified in "T100--T200".

- **level**: ["*debug*" | "*info*" | "*warning*" | "*error*" | "*raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected N parameters
  - 1° parameter: ["A|B"]
  - 2° parameter: ["C|D"]
  - N° parameter: ["*|E"]

unique

Specify that such field is a key value and must be unique in the file.

- **level**: ["*debug*" | "*info*" | "*warning*" | "*error*" | "*raw*"]
- **stop**: [*true* | *"false"*]
- **params**: *no parameters*

concat

Concatenate fields and string in order to obtain a new string that will replace field value. The following example for **trip_headsign** params = ["route_id"," - ","trip_headsign"] will replace the content of **trip_headsign** field with a concatenation of *route_id* plus the fixed string " - " plus *trip_headsign* value.

- **level**: ["*debug*" | "*info*" | "*warning*" | "*error*" | "*raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected N parameters

- 1° parameter: ["*field name*" |" *string*"]
- 2° parameter: ["*field name*" |" *string*"]
- N° parameter: ["*field name*" |" *string*"]

fix_real

Fix floating values to specified form in terms of decimal digits. So if a value appears with 20 decimal digits but decimal part is specified as 8 digits then such value will be truncated to 8 decimal digits.

- **level**: ["*debug*" | "*info*" | "*warning*" | "*error*" | "*raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: ["decimal digits"]

format

Allow use of a regular expression in order to validate field value.

- **level**: ["*debug*" | "*info*" | "*warning*" | "*error*" | "*raw*"]
- **stop**: [*true* | *"false"*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: ["rexgex"]

expired

Determine if date field compared with current date is already expired. Due to usage of local time a valid Time Zone shall be provided. A full list can be retrieved at the following URL: *https://en.wikipedia.org/wiki/List_of_tz_database_time_zones*.

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: [”time zone”]

ne

The function check if current field is NE (not equal) to a specified field on the same row.

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: [”field name”]

le

The function check if current field is LE (**L**ess than or **E**qual) to a specified field on the same row. **Note**: comparison is between strings, so even numeric values will be treated as string.

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: [”field name”]

ge

The function check if current field is GE (**G**reater than or **E**qual) to a specified field on the same row. **Note**: comparison is between strings, so even numeric values will be treated as string.

- **level**: [”*debug*” | ”*info*” | ”*warning*” | ”*error*” | ”*raw*”]
- **stop**: [*true* | *”false”*]
- **params**:
  - expected 1 (one) parameter
  - 1° parameter: [”field name”]

Functions

Functions are defined and optimized at feeds level, so a function that is available for a specific file, not necessarily is available also for other files.

Version 0.2.0 - trips.txt

"find_orphan_routes"

Raise a log message when for a specific *trip* is not possible to retrieve *route* information, since specified *route_id* does not exist in *routes.txt*.

"find_orphan_services"

Raise a log message when for a specific *trip* is not possible to retrieve *service* information, since specified *service_id* does not exist in both *calendar.txt* and *calendar_dates.txt*.

"find_orphan_shapes"

Raise a log message when for a specific *trip* is not possible to retrieve *shape* information, since specified *shape_id* does not exist in *shapes.txt*.

"delete_unused_shapes"

Mark as *deleted* all *shape_id* that are not linked with a *trip_id*. All of such *shapes* will be excluded from output.

"optimize_shapes_usage"

Merge all *shapes* that are duplicated in *shapes.txt* and replace the link in *trips.txt*. This optimization, in some cases, can drastically reduce the size for *shapes.txt*.

Version 0.2.0 - stop_times.txt

"find_orphan_links"

Raise a log message when for a specific *trip* is not possible to retrieve *stop* or *trip* information, since specified *stop_id* or *trip_id* does not exist respectively in *stops.txt* or *trips.txt*.

Version 0.2.0 - shapes.txt

"check_distance"

Compute the distance for all the shapes replacing current values.

Note: be sure to enable this function only if you want to replace current values. Take care that computed distance should be consistent also with *stop_times.txt*, so an error in *shapes.txt* most likely means that such error is also present in *stop_times.txt*.

Parameters: only take one parameter in order to specify the number of decimal digits for the precision (*default is 10 digits*).