GS1 Resolver Community Edition V2.2

Overview and Architecture

Document Summary

Document Item	Current Value
Document Name	GS1 Resolver CE V2.2 Overview and Architecture
Document Date	27 February 2020
Document Version	To be 1.0
Document Issue	0.1
Document Status	
Document Description	

Log of Changes

Release	Date of Change	Changed By	Summary of Change
0.1	February, 2020	Nick Lansley	Initial draft
0.2	April, 2020	Nick Lansley	Created an expanded section on Resolver's Docker Orchestration
0.3	November, 2020	Nick Lansley	Updated Mongo document format to be the new v2.2, and added the new DigitalLink Toolkit Server container

Table of Contents

1 Welcome!

This document forms part of the GS1 Resolver Community Edition v2.0, describing the various components that together form the GS1 Resolver Application which implements the GS1 Digital Link 1.1 standard.

There are three main components of the application:

1. **Data Entry API** and UI – a RESTful application programming interface which provides data entry services for the Resolver. It stored data in a SQL server.
2. **Resolver** – an application which receives incoming HTTP requests conforming to the GS1 Digital Link standard and provides a redirect to an onward (usually web) destination. Resolver relies on lookup data provided by a MongoDB database.
3. **Build** – an application which sits alongside Resolver, reading any updates from the SQL Server database and authoring Resolver documents that are stored in the local MongoDB database.

The three applications have been redesigned and rewritten from an earlier Resolver service that was based on the PHP programming language. The redesign resulted from the settled group of use cases that the GS1 Digital Link steering group has accepted, and has allowed for a great deal of simplification, hosting-cost savings and performance improvements to take place.

2 What is 'Resolving?'

Resolving is the action of intelligently redirecting a web client to the most appropriate target web address based on the data they wish to find more about, and the context (reason) they are making the request. In GS1 terms, the data they wish to find out more about is the GS1 Key Code and GS1 Key Value that globally uniquely identifies an item, product, or location. A common use is the desire to find out more about a product based on its GTIN (Global Trade Item Number) which is a value contained in the barcode printed on its packaging.

In addition to these identifiers, the request may include one or more qualifiers. For products these can include lot or batch numbers, and serial numbers. For example, the request may be wanting to find out if a particular serial number is subject to a recall notice.

2.1 What is the GS1 Digital Link standard?

The GS1 Digital Link standard is a set of rules for taking GS1 identifiers and qualifiers found in GS1 format in 1D barcodes, Datamatrices, and QR-Codes, and converting them into a web address which is then used to 'find out more' about this item, product or location.

2.2 Incoming request breakdown

Here is an example request inbound to the Resolver from the internet:
`https://example.com/gtin/00028028213981/lot/19002/ser/1973?linktype=gs1:pip`

This request conforms to the GS1 Digital Link standard. It consists of:

1. The inbound domain name to the Resolver: ***`https://example.com`***
2. GS1 *identifiers*: /**`gtin/00028028213981`**
3. A list of *qualifiers*: **`/lot/19002/ser/1973`**
4. A set of zero or more query-strings: **`linktype=gs1:pip`**

Resolver uses the information to look for an entry in its document database that matches these values. Each entry is stored by the Data Entry API as a ***ResolverRequest*** entry, so-called because Resolver needs to match one of these entries with the inbound request.

2.3 Responding to the request

If Rsolver finds a suitable *ResolverRequest* entry, it then needs to work out how to provide a suitable response to this request. Each active *ResolverRequest* has a set of one or more related ***ResolverResponse*** entries, each suitable for a different combination of four attributes representing what the request expects from the Resolver. For example, is the request wanting:

- A product information page or a patient info leaflet? *(Linktype)*
- French or South Korean language? *(IANA Language)*
- For use in Germany or Belgium? *(Context/territory)*
- Redirected to a destination URL coded in HTML or JSON? *(Mime-Type)*

These four attributes are compared with incoming values in the request (often in the HTTP Request Headers or as query-strings) in order to choose the most appropriate *ResolverResponse* entry:

- **Linktype**: A word from the GS1 Web Vocabulary of Schema.org vocabulary (such as 'https://gs1.org/voc/pip' for 'product information page').

- **IANA Language**: The 2-character language code from IANA signifying the language in which the destination link is written.
- **Context**: An extra attribute defining the context of the request - at GS1 Resolver we use country/territory codes (although context can be *anything*, a useful component of the GS1 Resolver Community Edition).
- **Mime-Type**: The IANA Media-type at the destination URL; for example, 'text/html' for a web page or 'application/json' for JSON data.

Should any of the four attributes not be discernible or not matched from the incoming request, Resolver can look at the 'default' values for each of the attributes when making its decision about which *ResolverResponse* entry is the most appropriate. For example, should Link-Type be missing, Resolver can see which *ResolverResponse* entry has *defaultLinkType* set to True.
It's important to note that these four attributes are hierarchical, that is:

- For each *Link-Type* there can be more than one *IANA Languages,*
- For each *IANA Language* there can be more than one *Contexts*, and
- For each *Context* there can be more than one *Mime-Types*.

Please note that each of the attributes includes a 'not applicable' option if not required for any particular ResolverResponse entry.

The Data Entry API does not store these four attributes in a hierarchical format since the API's clients need a simple-to-understand interface that's easy to access with standard 2D rows and columns as they are likely to be authored using a spreadsheet application. The Build application (described in Section 3.3) creates the hierarchical format in the MongoDB database for use by the Resolver application.

2.4 Walking Up the Tree

Should Resolver not find any suitable entry that exactly matches the incoming request, but there are other *ResolverRequest* entries that exist for the *same* GS1 identifiers, it will consider using those entries. To do this, it performs a 'walk up the tree', where it removes more qualifiers from the the 'right hand side' of the request to see if match can be found.

For example, if there is no *ResolverRequest* matching "/lot/19002/ser/1973", it looks for a match with just the qualifiers "/lot/19002". If that fails too, it then looks for the "root variant" - the basic information for this product with a *ResolverRequest* variant URI of "/" which should always be provided by the service (although not mandatory at this time).

2.5 Redirecting at the GCP Level

If a match fails at the GS1 Key Code and GS1 Key Value level, then there is one more data source that Resolver can try: A ***ResolverRedirect*** entry.

ResolverRedirect entries consist of a GS1 Company Prefix (GCP) entry, and a destination URL. Resolver takes the identifiers from the incoming request and removes more and more of characters from its value until it gets a match from *ResolverRedirect* entries (or it reaches as few as 2 characters). If such a match is found, Resolver immediately issues a redirect to the web address using the *ResolverRedirect* entry's *destinationUrl*. It is up to the destination web service to process the request from this point.

For example, should GS1 Switzerland wish to operate its own resolver service, then the GS1 Resolver at *https://id.gs1.org* could have a *ResolverRedirect* entry with GS1 Key Code "01" (for GTIN) and GS1 GCP Value "078" (and no *ResolverRequest*/*ResolverResponse* entries). This would cause Resolver to redirect all incoming requests for GTINs with GCP "078" to be redirected to an onward destination such as *https://id.gs1.ch*

3 Application Details

3.1 Data Entry API

The Data Entry API is coded as a Node/JS v16 'Express' framework to make it easy to create an API based on three classes describing the three types of entries described in section 2

The API documentation is being built and will be available soon as a separate document.

3.2 SQL Server Data Entry Database

The database underlying Data Entry API is a single Microsoft SQL Server capable of running in the Azure cloud computing platform. The database consists of three tables with columns that match the properties of the three classes described in the previous section:

- *ResolverRequest*, represented by table **uri_requests**
- *ResolverResponse*, represented by table **uri_responses**, and
- *ResolverRedirect*, represented by table **gcp_redirects**

In addition, a *synchronising* table provides simple synchronising mechanism for Resolver's Build application to keep up to data with data changes:

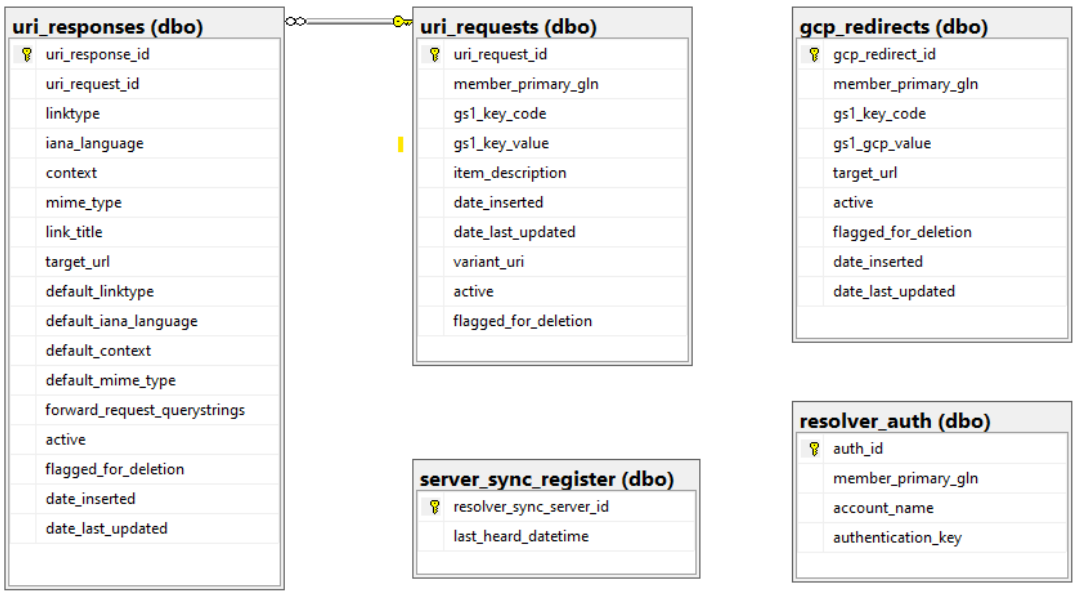
- **server_sync_register** is a table that stores a Build registration event, where a Build application connects to the database in order to request changes. SQL Server stores the hostname and registration date/time of the Build application, and returns the previous most recent date/time that this particular hostname was heard.

This synchronising design allows for several Resolvers to exists, each with their local MongoDB database and Build application, and all able to retrieve their own list of changes.

Finally, a simple authentication mechanism is provided to control access to the API.
The authorisation is provided by:

- resolver_auth – a table which stores allowed authentication keys of up to 60 characters, and a 'member_primary_gln' which is a unique account identifier for each entry. We use GLN (Global Location Number) but any allowed numeric account identifier can be used. This identifier is used to track ownership of data in the database.

3.2.1 SQL Server database diagram for Data Entry API



All access to the database is performed by the Data Entry API and Build applications solely through stored procedures, allowing for maximum performance and minimised security issues (especially through SQL injection).

The script to build the SQL database's tables and stored procedures is available in the Data Entry API repository.

3.2.2 SQL Server database tables and stored procedures

3.2.2.1 SQL table: resolver_auth

Purpose: This table stores simple 'account' data.

[auth_id] [bigint] IDENTITY(1,1) NOT NULL (Primary Key)
A unique identity for each column in this table

[authentication_key] [nchar](64) NOT NULL
The authentication key is supplied by account holders uploading and downloading their data using the data entry API and its web-based UI. The authentication key is supplied in a standard HTTP Authentication Header (see the mechanism in action in resolver_data_entry_server/public/javascripts/upload.js).

[member_primary_gln] [nchar](13) NOT NULL
The account Id is stored in column member_primary_gln. In the world of GS1, all members globally have at least on GLN (Global Location Number) often used as their account Id in many GS1 territories. It is a 13-digit numeric value. However, in terms of Resolver CE's operation, this can be any value at all. Several entries in this table can have the same member_primary_gln value which demonstrates that they are member of the same organisation and have shared access to their data. The same value is stored in all entries in the uri_requests table, so as to identify ownership.

[account_name] [nvarchar](255) NOT NULL
This is a free text column into which should be stored details of the account holder.

3.2.2.2 SQL table: uri_requests

Purpose: This table stores the 'ResolverRequest' entries, used by Resolver to identify an incoming request to the service.

[uri_request_id] [bigint] IDENTITY(1,1) NOT NULL (Primary Key)
This is an internally generated unique identifier for each entry in this table.

[member_primary_gln] [nchar](13) NOT NULL
This value links ownership of this entry to the organisation identified in the resolver_auth table.

[gs1_key_code] [nvarchar](20) NOT NULL
This column stores the Identifier of the GS1 Key. This value is numeric although stored a string as its number has no arithmetic significance. Example "01" which signifies that the data in column gs1_key_value is a GTIN.

[gs1_key_value] [nvarchar](45) NOT NULL
The value of the GS1 key, for example the GTIN "05051089990565"

[item_description] [nvarchar](2000) NOT NULL
This column stores the administrative description of the item being stored. It is not normally shown to the public, but for recognition of the product by someone uploading or downloading their data. Data can be stored as plain text, but in this column you will often see values such as: '[]UkFJRCBGbHkgJiBXYXNwIEtpbGxlciAzMDBtbA==='. This is a base64 representation of the item's description, enabling the storage of text data in many different locale and language formats. The base64-encoded text is preceded by two special characters '[]' which tells Resolver to decode the text before displaying it. You can store ordinary text such as ' My Product' directly in this column if you need to edit in the database directly. As long as you don't start the text with '[]' then Resolver will leave the text as is and not try and decode it.

[date_inserted] [datetime] NOT NULL
An internally generated date/time value representing the UTC date and time that the record was inserted into the table.

[date_last_updated] [datetime] NOT NULL
An internally generated date/time value representing the UTC date and time that the record was last updated in the table.

[variant_uri] [nvarchar](255) NOT NULL
This column stores the 'key qualifiers' in Digital Link format. The most common use of this column is just to store '/' which represents the fact that there are no key qualifier such as lot number or serial number. Alternative examples are "/10/AB1234" which represents Lot (or Batch) 'AB1234', and '/10/AB1234/21/958410989714' which includes the precious lot number and a serial number too.

[active] [bit] NOT NULL
Set to 1 (logical true) if Resolver is store this ResolverRequest (and all of its responses) in its document database. If set to 0 (logical false) Resolver will remove this entry from its document database. This flag is a useful way of adding or removing live entries without deleting them.

[flagged_for_deletion] [bit] NOT NULL

If this entry is no longer needed, setting this column to 1 (logical true) will filter it out of any requests, and an end-of-day job can remove them when the database is not under load.

3.2.2.3 SQL table: uri_responses

Purpose: Stores entries for ResolverResponse. That is, for each ResolverRequest entry (stored in the uri_requests table) this table stores one or more responses for the Resolver to consider when replaying to an incoming request.

[uri_response_id] [bigint] IDENTITY(1,1) NOT NULL
This is an internally generated unique identifier for each entry in this table.

[uri_request_id] [bigint] NOT NULL
This column uses a foreign key constraint to link this table to the uri_requests table, so that the data is easy to join when retrieving requests and their responses.

[linktype] [nvarchar](100) NOT NULL
This column stores the 'linktype' the GS1 web vocabulary word, or Schema,.org word representing the type of link. Examples are "*https://gs1.org/voc/relatedVideo*" and "*https://schema.org/mpn*"

[iana_language] [nchar](2) NOT NULL
The two-character language subtag identifier from *https://www.iana.org/assignments/language-subtag-registry/language-subtag-registry* examples: "en", "fr"

[context] [nvarchar](100) NOT NULL
The context is a freeform text column that can be used for all kinds of contexts depending ion the use of Resolver. The official GS1 Resolver uses territory as its context – such as "gb" or "us". Combined with language it means that a combination of language and territory can be used to help an end user get the relevant local information in their own language. Other Resolvers might consider contexts such as department category, or perhaps the 'Dewey' classification if this resolver was storing book data for an organisation.

[mime_type] [nvarchar](45) NOT NULL
The document (IANA Media) type at the target web address. Examples include "text/html", "application/json". See here for a full list: *https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types*

[link_title] [nvarchar](100) NOT NULL
The title of the link, supplied in the language that matches the one set in the iana_language column. To cope with many language character sets and locales, Resolver stores this as base64 encoded string preceded by "[]" just like it does for item_description in uri_requests.

[target_url] [nvarchar](1024) NOT NULL
The destination web address which Resolver will redirect to.

[default_linktype] [bit] NOT NULL
Set to 1 (logical true) if this entry represents the default link type.

[default_iana_language] [bit] NOT NULL
Set to 1 (logical true) if this entry represents the default link language.

[default_context] [bit] NOT NULL,
Set to 1 (logical true) if this entry represents the default link context.

[default_mime_type] [bit] NOT NULL
Set to 1 (logical true) if this entry represents the default mime (IANA Media) type.

[forward_request_querystrings] [bit] NOT NULL

IF set to 1 (logical true), any query string data in the incoming request is added to the outgoing response, appended to the end of the web address stored in target_url. If set to 0 (logical false), no query strings are appended, although if query strings are stored as part of the web address in target_url, they are not removed.

[active] [bit] NOT NULL
If set to 1 (logical true), this entry is deemed as active. If set to 0 (logical false), the request will still work with other responses. If all the responses for a request are set to 0, then the request is also deactivated even if its active flag is set to 1 in the uri_requests table.

[flagged_for_deletion] [bit] NOT NULL
If this entry is no longer needed, setting this column to 1 (logical true) will filter it out of any requests, and an end-of-day job can remove them when the database is not under load.

[date_inserted] [datetime] NOT NULL
An internally generated date/time value representing the UTC date and time that the record was inserted into the table.

[date_last_updated] [datetime] NOT NULL
An internally generated date/time value representing the UTC date and time that the record was last updated in the table.

3.2.2.4 SQL table: gcp_redirects

Purpose: This table stores Prefixes – part od the 'left hand side' of a GS1 Key Value which can be used to issue a redirect if the full value is not in the uri_requests table. When members join GS1 they are given a 'prefix' from which their own values are generated according to strict mathematical rules. For example, if an organisation wishes to set up its own Resolver to resolve their product GTIN barcodes, this Resolver can store just the GCP prefix(es) they were given, causing all requests for any of their products to be redirected from this Resolver to their Resolver.

[gcp_redirect_id] [bigint] IDENTITY(1,1) NOT NULL
This is an internally generated unique identifier for each entry in this table

[member_primary_gln] [nchar](13) NOT NULL
This value links ownership of this entry to the organisation identified in the resolver_auth table.

[gs1_key_code] [nvarchar](20) NOT NULL
This column stores the Identifier of the GS1 Key. This value is numeric although stored a s string as its number has no arithmetic significance. Example "01" which signifies that the data in column gs1_gcp_value is a GTIN prefix.

[gs1_gcp_value] [nvarchar](45) NOT NULL
The partial key prefix value.

[target_url] [nvarchar](1024) NOT NULL
The web address which a matched GCP will be redirected by Resolver

[active] [bit] NOT NULL
If set to 1 (logical true), this entry is deemed as active. If set to 0 (logical false), Resolver won't respond to this entry even if there is a match.

[flagged_for_deletion] [bit] NOT NULL
If this entry is no longer needed, setting this column to 1 (logical true) will filter it out of any requests, and an end-of-day job can remove them when the database is not under load.

[date_inserted] [datetime] NOT NULL

An internally generated date/time value representing the UTC date and time that the record was inserted into the table.

[date_last_updated] [datetime] NOT NULL
An internally generated date/time value representing the UTC date and time that the record was last updated in the table.

3.2.2.5 SQL table: server_sync_register

Purpose: Allows many Resolvers to independently synchronise their data from this database.

GS1 Resolver was designed from the ground up with scale in mind. GS1's own official Resolver is likely to be running in data centres around the world but still accessing a single SQL database to source their data, in order to store local copies in their local (MongoDB) databases.

When initialised, each Resolver instance generated its own unique 'server_id' which gets stored in this server_sync_register table.

As a result, it can find out when it previously connected, so it can find all the ResolverRequests, ResolverResponses and ResolverRedirects that were inserted or updated since that date/time, using the last_updated_datetime column in each table.

The action of registering and retrieving the last_heard_datetime causes the stored procedure that gets this information to update the record with the current UTC data and time.

[resolver_sync_server_id] [nchar](12) NOT NULL
A 12-character unique but consistent identity for each running Resolver.

[last_heard_datetime] [datetime] NOT NULL
The UTC date and time that this Resolver was previously heard.

3.2.2.5.1 Stored Procedures

This database is designed for Microsoft SQL Server, but the tables can be rebuilt in any modern SQL flavour from Postgres to MySQL.

The stored procedures are a little more 'SQL Server' focussed but their intentions are simple and could be re-written to work in other SQL flavours. Alternatively, theses procedures can be removed completely and their logic transferred into application code. You will need to read through the code of each stored procedure to see what it is doing, as many perform more than one task.

[BUILD_GET_URI_Requests_using_gs1_key_code_and_value]
Returns all the ResolverRequests with a given GS1 Key Code and Value. Use by the BUILD service which creates a single MongoDB document from these entries.

[BUILD_Get_GCP_Redirects]
Used by the BUILD service to get changed ResolverRedirect entries

[BUILD_Get_URI_Requests]
Use by the BUILD service to get changed ResolverRequest entries

[BUILD_Register_Sync_Server]
Used by the BUILD service to retrieve the previously heard date and time. In doing so this stored procedure updates the data and time to now.

[COUNT_URI_Requests_using_member_primary_gln]
Uses by the Data Entry API to get a count of all entries for a particular account

[CREATE_GCP_Redirect]
Creates a new ResolverRedirect (includes calling into UPDATE_GCP_Redirect if the entry already exists).

[CREATE_URI_Request]
Creates a new ResolverRequest (includes calling into UPDATE_URI_Request if the entry already exists).

[CREATE_URI_Response]
Creates a new ResolverResponse (includes calling into UPDATE_URI_Response if the entry already exists).

[DELETE_GCP_Redirect]
Marks a ResolverRedirect as deleted

[DELETE_URI_Request]
Marks a ResolverRequest as deleted

[DELETE_URI_Response]
Marks a ResolveResponse as deleted

[ENDOFDAY_Delete_Flagged_GCP_Entries]
An end-of-day process to delete ResolverRedirect entries marked for deletion

[ENDOFDAY_Delete_Flagged_URI_Entries]
An end-of-day process to delete ResolverRequest and ResolverResponse entries marked for deletion

[GET_GCP_Redirects]
Gets all the ResolverRedirects for a given account

[GET_URI_Requests_using_gln_and_gs1_key_code_and_value]
Gets the ResolverRequests for a given account and GS1 Key & Value

[GET_URI_Requests_using_gs1_key_code_and_value]
Gets the ResolverRequests for a given GS1 Key & Value

[GET_URI_Requests_using_member_primary_gln]
Gets the ResolverRequests for a given account

[GET_URI_Responses]
Gets all the ResolverResponses for a given ResolverRequest Id

[INTERNAL_Ensure_Request_Has_Active_Responses]
A Stored procedure used by other stored procedures to mark a ResolverRequest as inactive if all its ResolverResponses are marked as inactive.

[READ_GCP_Redirect]
Gets a ResolverRedirect entry based on its Id

[READ_Resolver_Auth]
Gets the details of an account if the supplied Authentication Key matches.

[READ_URI_Request]
Gets a ResolverRequest using its Id

[READ_URI_Response]
Gets a ResolverResponse based on its Id

[UPDATE_GCP_Redirect]

Updates a ResolverRedirect

[UPDATE_URI_Request]
Updates a ResolverRequest

[UPDATE_URI_Response]
Updates a ResolverResponse

3.3 BUILD Application

The ***Build*** application resides in its own Docker container called *build-sync-server* and runs alongside a local MongoDB database (in a separate locally available container called *id-mongo-server*) and the Resolver application itself (in a separate locally available container called *id-web-server*) - see section 4.

Build is written in JavaScript for Node v13.8 and uses a Docker container based on an official Docker image from the OpenJS Foundation.

Build takes data from the SQL Server Database and writes it into a local MongoDB database. The format is different to the simple two-dimensional sets of rows and columns in SQL Server and is covered in section 3.4.

Build has direct access to the SQL Server Database (through stored procedures) and can batch updates from the database in order to keep the load on the single, central SQL Server manageable.

By default, *Build* asks for updates every 60 seconds, although this can be changed in its Dockerfile configuration.

3.4 MongoDB Database

MongoDB is document database (as compared to a relational database). It was chosen to allow for the Resolver to perform high speed lookups of data without having to join tables and otherwise cause load on the single, central SQL Server database.

MongoDB resides in its own Docker container called *id-mongo-server* and runs alongside the Build application (in a separate locally available container called *build-sync-server*) and the Resolver application itself (in a separate locally available container called *id-web-server*) - see Section 4.

MongoDB uses a container based on an official Mongo Docker 4.2.3 image from the Docker Community.

The format of MongoDB document for Resolver and created by *Build* is designed for high performance access to the appropriate Request, Response and Redirect data.

Here is an example Resolver document in MongoDB (with annotations):

```
{
  "_id": "/01/09506000134376", ← This is the 'primary key' consisting of an identifiers path of keycode and Key
  "unixtime": 1604940000,  ← this was the unixtime (count of seconds since /01/01/1970) when the entry was last updated
  "/": {                ← the qualifiers path - in this case there are no qualifier so we use the 'root' path '/'
    "active": true,     ← this entry is active
    "itemDescription": "Dal Giardino Medicinal Compound 50 x 200mg",       <- name of the item
    "responses": [                <- an array of responses that Resolver can choose from to satisfy the request
      {
        "link": "https://dalgiardino.com/medicinal-compound/pil.html",
        "title": "Product Information Page",
        "linkType": "gs1:epil",
        "ianaLanguage": "en",
        "context": "xx",
        "mimeType": "text/html",
        "active": true,
        "fwqs": true,
        "defaultLinkType": false,
        "defaultIanaLanguage": true,
        "defaultContext": true,
        "defaultMimeType": true
      },
      {
        "link": "https://dalgiardino.com/medicinal-compound/",  ← a link that the request will be resolved to if..
        "title": "Product Information Page",
        "linkType": "gs1:pip",                                  ← the request required a link-type of 'pip'
        "ianaLanguage": "en",                                   ←                     English language
        "context": "xx",                                        ←                     "Any" context (signified by 'xx')
        "mimeType": "text/html",                                ←                     A MIME type of text/html
        "active": true,                                         ← This response is active
        "fwqs": true,                                           ← Any query strings in the request will be appended to the response
        "defaultLinkType": true,                                ← If no linktype arrived with the request, this response is one that can be used
        "defaultIanaLanguage": true,                            ← If no lang arrived with the request, this response is one that can be used
        "defaultContext": true,                                 ← If no context arrived with the request, this response is one that can be used
        "defaultMimeType": true                                 ← If no MIME arrived with the request, this response is one that can be used
      },
      {
        "link": "https://dalgiardino.com/medicinal-compound/index.html.ja",
        "title": "Product Information Page",
        "linkType": "gs1:pip",
        "ianaLanguage": "ja",
        "context": "xx",
        "mimeType": "text/html",
        "active": true,
        "fwqs": true,
        "defaultLinkType": false,
        "defaultIanaLanguage": true,
        "defaultContext": true,
        "defaultMimeType": true
      }
    ]
  },
  "/21/{serial}": {    ← here is a different qualifier path for requests with a serial number. Here we see a template variable
    "active": true,       called {serial} which we will match with the same variable in the link below.
    "itemDescription": "Dal Giardino Medicinal Compound 50 x 200mg with serial number",
    "responses": [
      {
        "link": "https://dalgiardino.com/medicinal-compound/index.html?serialnumber=%7Bserial%7D", ← A urlencoded link with variable {serial}.
        "title": "Product Information Page",                                                          This will be substituted with the value that
        "linkType": "gs1:pip",                                                                        arrived with the request in the qualifier path.
        "ianaLanguage": "en",
        "context": "xx",
        "mimeType": "text/html",
        "active": true,
        "fwqs": true,
        "defaultLinkType": false,
        "defaultIanaLanguage": true,
        "defaultContext": true,
        "defaultMimeType": true
      }
    ],
    "variables": [
      {
        "21": "{serial}"  ← This array helps Resolver substitute variable names with actual values in the response.
      }
    ]
  }
}
```

This design allows Resolver to make lightning decisions because all the data for this GS1 Key Code and GS1 Key Value are available in a single document.

In the above document we have the representation of a single *ResolverRequest* with GS1 Key Code 'gtin' or '01', GS1 Key Value "05011157888163" and Variant Uri "/" (the 'root qualifier'). We also have five *ResolverResponse* entries, each related to that same *ResolverRequest*.

Some documents have two or more *ResolverRequest* entries embedded in them, each with the same GS1 Key Code and GS1 Key Value, but with different Variant URIs. Embedded within each Variant Uri will be the associated *ResolverResponse* entries.

This is useful for Resolver to 'walk up the tree' as described earlier should the exact requested Variant Uri be unavailable in the document.

3.5 The Resolver Application

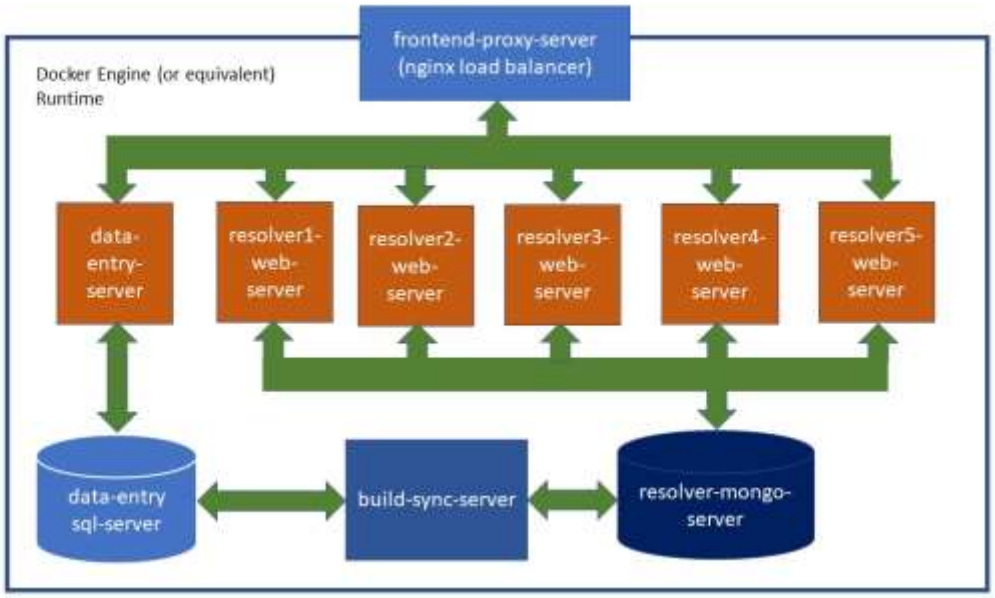
Based on the output of the other applications, the Resolver application has these tasks:

1. Receive an incoming request from the internet
2. Detect whether it is a valid GS1 Digital Link-compliant URL and, if so,
3. Extract the various elements of that URL.
4. Lookup the GS1 Key Code and GS1 Key Value in its local MongoDB database
5. If found, detect if a matching Variant URI is found OR 'walk up the tree' to get a match
6. Match the four requested attributes of link-type, IANA language, context and Mime-type OR match using defaults in order to get the required (destination) link
7. Send the link as an HTTP redirect back to the calling client.

The Resolver application is written in Node v15.1 and uses a container based on an official Docker image from the OpenJS Foundation. Node was chosen because of its high performance in a small CPU footprint, keeping the costs of running the service in a cloud computing platform as low as possible.

Node v15 takes advantage of JavaScript's latest feature set that simplifies coding tasks and makes source code more 'readable', along with improved performance from V8 engine's new optimising compiler.

4 Resolver's Orchestration



GS1 Resolver's orchestration involves the following containers:

- **Frontend_proxy_server** – this container takes inbound traffic and routes it to appropriate containers inside the orchestration. It uses a load-balancing algorithm to ensure that load is spread across the other containers. This container houses the official nginx Docker image provided by NGINX Inc.
- **Data-entry-server** – the data entry API and an upload web page available for triallists to add or update their trial data. It is connected to a SQL Server database using a secure data link outside the orchestration but in a nearby Azure data centre. The service is written in JavaScript Node/JS v13.
- **ResolverN-web-server** – five instances of the same container run in parallel with each other, recieving inbound resolver calls to Five containers are used as, under load, this has been shown to be the most efficient use of CPU time. Too few container and their shared resolver-mongo-server idles under load, whereas more than 5 causes the resolver-mongo-server to be at maximum load while containers start idling while they wait.
- **Resolver-mongo-server** – a document database into which resolver documents are stored in manner that allows high speed access and resolving. This server has its own persistent volume (that is, an area of disk space that 'survives' the mongo-server being restarted by Docker Engine or Kubernetes). This container houses Mongo Community Edition v4.2 only, it contains no additional applications for Resolver.
- **Resolver-data-entry-server** – an instance of Microsoft SQL Server 2017 set (by Dockerfile) to 'Express' edition – free to use but with a 10GB limit. This container is the one you are most likely to move to an external cloud-based SQL data service.
- **Build-sync-server** – every minute this server requests new, updated and deleted records from the s1l-server and reformats the data for storage in resolver-mongo-server. This application has been coded in Node/JS v13.
- **DigitalLink-toolkit-server** (not shown). An internal library service available to the other containers that wish to test incoming data against a reference implementation of the GS1 Digital Link Standard.

5 Build and launch on your desktop

5.1 Step by step instructions using Docker Engine

The quickest way to get to know how Resolver works is to set it running on your own computer. Please follow these instructions:

1. Install the Docker system on your computer. Head to https://www.docker.com/products/docker-desktop for install details for Windows and Mac. If you are using Ubuntu Linux, follow install instructions here: https://docs.docker.com/install/linux/docker-ce/ubuntu/

2. git clone the repository (containing this document) onto your computer.

3. Open a terminal prompt (Mac and Linux) or PowerShell (Windows 10) and change directory to the one at the 'root' of this repository, so you can see the file docker-compose.yml in the current folder.

   Type this command:

   ```
   docker-compose config
   ```

   ...which should simply list the docker-compose.yml without error, and then type this command
   docker info
   which will Docker to check that all is well with the service and give some run-time statistics. If you're not seeing any errors then we're good to go.

4. Make sure you have a good internet connection, and then type this command:

   ```
   docker-compose build
   ```

   ...which will cause Docker to build the complete end-to-end GS1 Resolver service. This will take quite a while with lots of text flowing up the terminal window as downloading and compiling of the service takes place. Even on a high speed connection the build-from-scratch will take 10-15 minutes.

5. You are nearly ready to start the application. Before you do, make sure you have no SQL Server service, MongoDB service or port 80 web server running on your computer as they will clash with Docker as it tries to start the service up. Once completed, type this to start everything up:

   ```
   docker-compose up -d
   ```

   (the -d means 'disconnect' - docker-compose will start up everything then hand control back to you).

6. Now wait 10 seconds while the system settles down (the SQL Server service takes a few seconds to initialise when 'new') then copy and paste this command (ensuring it is on one terminal line), which will run a program inside the SQL Server container, creating the database and some example data.

   ```
   docker exec -it  resolver-sql-server bash
   ```
   to enter the container and get a terminal prompt, then run this command:

   ```
   /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P its@SECR3T! -i
   /gs1resolver_sql_scripts/sqldb_create_script.sql
   ```

7. Head to http://localhost/ui and select the Download page.

   In the authorization key box, type: "12345" and click the Download button. Save the file to your local computer.

8. Click the link to go back to the home page, then choose the Upload page.

9. Type in your authorization key (12345), then choose the file you just downloaded. The Upload page detects 'Download' -format file and will set all the columns correctly for you. Have look at the example data in each column and what it means (see next section for details of the file columns).

10. Click 'Check file' followed by 'Upload file' and the file should be uploaded with no errors. **Completing these steps means that your installation of Resolver has been successful.**

11. By now the local Mongo database should be built (a build event occurs every one minute) so try out this request in a terminal window:

```
curl -I http://localhost/gtin/09506000134376?serialnumber=12345
```

which should result in this appearing in your terminal window:

```
HTTP/1.1 307 Temporary Redirect
Server: nginx/1.19.0
Date: Mon, 09 Nov 2020 17:39:55 GMT
Connection: keep-alive
Vary: Accept-Encoding
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: HEAD, GET, OPTIONS
Access-Control-Expose-Headers: Link, Content-Length
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
X-Resolver-ProcessTimeMS: 9
Link: <https://dalgiardino.com/medicinal-compound/pil.html>; rel="gs1:epil";
type="text/html"; hreflang="en"; title="Product Information Page",
<https://dalgiardino.com/medicinal-co
mpound/>; rel="gs1:pip"; type="text/html"; hreflang="en"; title="Product
Information Page", <https://dalgiardino.com/medicinal-
compound/index.html.ja>; rel="gs1:pip"; type="text/htm
l"; hreflang="ja"; title="Product Information Page",
<https://id.gs1.org/01/09506000134376>; rel="owl:sameAs"
Location: https://dalgiardino.com/medicinal-compound/?serialnumber=12345
```

This demonstrates that Resolver has found an entry for GTIN **09506000134376** and is redirecting you to the web site shown in the 'Location' header. Can also see this in action if you use the same web address - you should end up at our fictional brand site!

12. Shutting down the service

    To close the entire application down type this:
    docker-compose down

    Since the data is stored on Docker volumes, the data will survive the shutdown and be available when you 'up' the service again.

    If you wish to delete the volumes and thus wipe the data, type these commands:
    docker volume rm gs1_digitallink_resolver_ce_resolver-document-volume
    docker volume rm gs1_digitallink_resolver_ce_resolver-sql-server-dbbackup-volume
    docker volume rm gs1_digitallink_resolver_ce_resolver-sql-server-volume-db-data

```
docker volume rm gs1_digitallink_resolver_ce_resolver-sql-server-volume-db-log
docker volume rm s1_digitallink_resolver_k8s_resolver-document-volume
```

If the above volume are the only ones in your Docker Engine then it's quicker to type:

```
docker volume ls
```

to confirm, then to delete all the volumes type:

```
docker volume prune
```

5.2 Kubernetes installation pointers

Running your Resolver application in a Kubernetes cluster is an advanced topic and not within the scope of this document; however if you are skilled and experienced with K8s clusters and orchestration, then the included YAML files are templates that you can adjust to being Resolver to life in a K8s cluster without much work.

Kubernetes pulls images of containers from a public or private container registry. You will need to build, tag, and push the Resolver container images created by **`docker-compose build`** into your own container repository, then add the path to those containers into placeholders in each of the YAML files.

Add necessary secrets and other authentication items to access your container registry and K8s cluster, and finally use the command **`kubectl apply -k .\`** which will read the 'root' YAML file *kustomization.yaml*, which includes links to all the YAML files in the /k8s folder in this repository, and build your cluster.

6 Data Entry File Upload/Download columns

Here are all the columns that make up a download file, and can make up an upload file:

- gs1_key_code – the name or AI number of a code – "01", "gtin"
- **gs1_key_value** – the value of the code – "05052004033992"
- **item_description** – a description of the item, for internal / admin use
- variant_uri – a string containing the qualifiers for the entry - "/", "/lot/{lotnumber}"
- linktype - the type of link from the GS1 web vocabulary - "https://gs1.org/voc/pip"
- iana_language – the 2-character language identifier – "en", "fr"
- context – the context of the request such as territory "GB", "US"
- mime_type – the document type required – "text/html", "application/json"
- link_title – the title describing the destination link
- **target_url** – the target web address which Resolver will redirect you
- default_linktype – indicates this entry is to be used if no linktype is found – "Y", "N"
- default_iana_language - indicates this entry to be used if no lang found - "Y", "N"
- default_context - indicates this entry to be used if no context found - "Y", "N"
- default_mime_type - indicates this entry to be used if no doc type found - "Y", "N"
- fwqs – indicates that any query strings in the request are to be forwarded in response
- active – indicates that Resolver should act / not act on the request – "Y", "N"
- date_inserted – read only for information in download file
- date_last_updated – read only for information in download file

Columns in bold are mandatory; the remainder can be set as defaults on the upload web page. or can be derived by Resolver. For example, variant_uri is derived as "/" if not included.