Explorer of Grid Load

August 2016

Author:
Mayank Sharma

Supervisor(s):
Joao Antunes Pequenao

CERN openlab Summer Student Report 2016

Acknowledgements

Working on EGL as a CERN Openlab summer student has been one the best learning experiences for me so far. A special thanks to my supervisor, Joao Pequenao for pioneering the visualizations and guiding me through the overall project and also for single handedly being the Best Supervisor Ever! I have learnt so much not only about software development and design but also about other aspects of life for which I am forever grateful.

I would also like to thank Mellisa Gaillard for always being supportive and resourceful and for helping us connect with the right resources that really helped expedite a lot of work at our end.

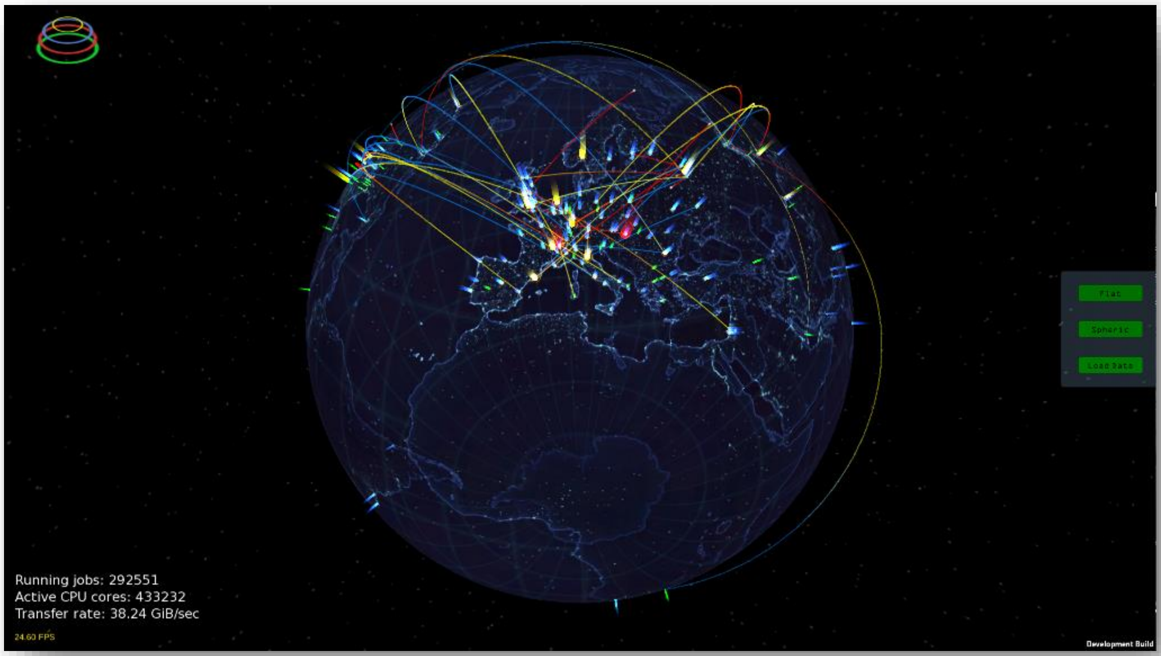
I am also grateful for the help provided by Edward Karawakis in getting us the access to the data sources and helping us work through the Google Earth KML besides the useful tips and feedback. I would like to thank Alberto Aimar and Julien Ludec for their inputs towards EGL.

Lastly, I am really thankful to the CERN Openlab team for the opportunity of being a part of their summer student program. I shall always cherish the entire experience!

Table of Contents

1 Introduction

Big data is a reality scientists face every day. Especially now that CERN projects and collaborations have become Global. The Worldwide LHC Computing Grid processes petabytes of data connecting over 200 active sites in more than 42 countries. It forms the backbone for the data analytics and high processing computing possible through these sites. Therefore, a reliable Grid means reliable operations at collaborating projects, partner companies, institutes and universities. Therefore, it is necessary to have a reliable tool for monitoring and visualizing various performance metrics and other relevant data about the grid. That's where EGL or Explorer of the Grid Load comes into the picture.



```
Running jobs: 292551
Active CPU cores: 433232
Transfer rate: 38.24 GiB/sec
```

1.1 What is EGL?

EGL is a visualization tool specializing in plotting WLCG statistics. It is very flexible in design and gives us a lot of control over the visualizations. It can be customized and adapted for different scenarios. For example, it can be deployed at CERN visit points. As it can also display intricate details about grid operations on a globe, it may be used for monitoring purposes as well.

1.2 Why EGL?

EGL makes life easier for a lot of people who have previously relied on the visualization capabilities of Google Earth to try to achieve similar results. In fact, EGL goes way beyond the scope and capabilities of Google Earth type visualizations and gives complete control to the developers. It allows parsing, processing and validating the data fetched from multiple sources through the API itself rather than relying on separate servers to do the heavy lifting. It not only reduces load on servers by performing most calculations and data manipulations by itself but also significantly cuts down on the size of files transferred by the servers.

From a developer's perspective, EGL API is based on the latest trends in software development. It is **highly modular** and can be broken down into components with clearly defined roles and responsibilities. It is therefore easy to diagnose bugs and patch them up quickly. EGL API is **Event Driven**. This means that different API components can subscribe to events and get notifications when they occur. Events are triggered whenever anything happens in EGL API. For example, different events are announced when data fetching is complete or parsing is complete or processing of data is done or when the fetched data is ready to be plotted by the visualization engine. This is implemented through the **Publisher-Subscriber/ Observer pattern**. To synchronize different events i.e. control when they are triggered is vital for proper operation of the API. EGL implements a **Sequence Pipeline** (Pipes and Filters/ Chain of Responsibility pattern) to control the order in which the events occur. A pipeline typically can be broken into a number of **stages** such as: Data Fetch, Data Parse, Data Processing, Data Validation. Each pipeline **stage controls certain events** and a stage is only triggered when the relevant events from the previous stage have been completed. To abstract the internal working of EGL API from the visualization engine, it implements a **Service Layer** that provides convenience functions to directly obtain the latest processed data at any point in time and also allows to enquire about the state of EGL API in a request-response fashion. From the perspective of the visualization engine, the service layer essentially reduces the entire EGL API into a single class (DaVinciPallete.cs) which provides functions to fetch latest processed data.

EGL is **Multi-Threaded**. It implements Coroutines for fetching, parsing and processing data to ensure there is no unnecessary load on the UI thread that may make the visualization laggy. As a lot of operations are performed in parallel, the Event System and the Sequence Pipeline provide a neat solution to control the asynchronous activities occurring in the core API.

1.3 Advantages over Google Earth

EGL has the following advantages over the Google Earth tool that has been around:

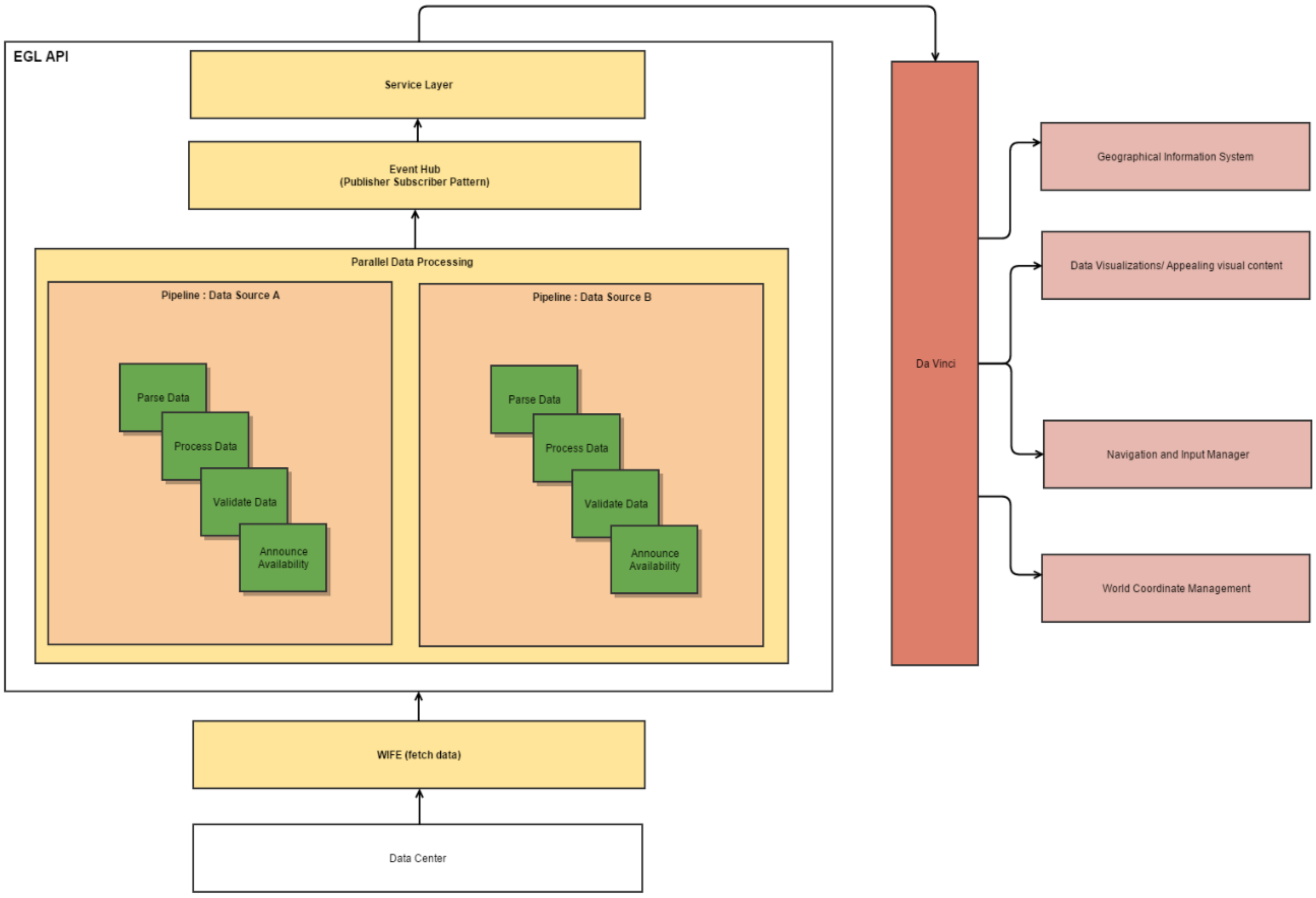
1. We are not limited by the visualization and data handling capabilities of Google Earth as EGL has been built with Unity, which is among the most powerful game engines of today
2. EGL is Multi-Platform. It can be deployed for Android/ iOS/ WebGL/ Standalone
3. As we have more control over the data and visualizations, there is less load on servers that generate huge xml files for Google Earth. A lot of the calculations

performed by these servers for preparing animation data and for interpolating data links is not required by EGL as it internally manages both of these.

4. EGL comes with a flexible core API which can be adapted visualizing any kind of statistics on a Globe.

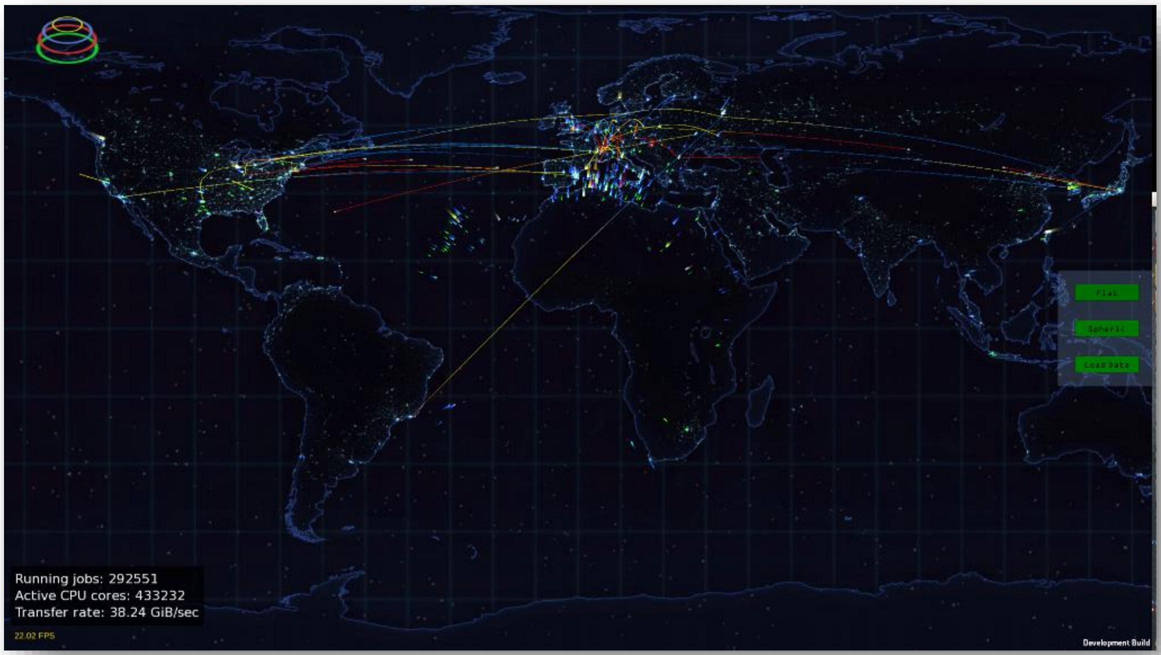
1.4 Current status

The big picture (simplified) for EGL API is as follows:



At present, we are between the alpha and beta release. The overall architecture for EGL has been set up and the EGL API fluidly interacts with the Visualization Engine. So, the main task remaining is to keep adding more data sources and visualization features as per the requirements that may arise.

p.s. In the above diagram, WIFE i.e. *Web Interface for EGL* consists of *ReconChewbacca.cs* and *Han.cs*. Here is a screenshot of the latest EGL implementation:



```
Running jobs: 292551
Active CPU cores: 433232
Transfer rate: 38.24 GiB/sec
```

To test drive EGL, you can use the following link:

http://ml-server01.cern.ch/files/EGL/

1.5 Future Work/ Scope

- Refine data sourced and incorporate changes in EGL API to make data fetching and parsing more efficient especially for Google Earth KML and Kathy Noble's KML
- Implement country/ region wise boundaries i.e the GIS information
- Adapt EGL for different visit points at CERN
- Explore the potential of EGL to become a generic standalone tool for visualizing any type of statistics on the Globe.

2 EGL Tech Manual

Now that we have a big picture for EGL. Let's jump into some technical details of the EGL API and understand how it works and why it works the way it works. This manual is **complementary** to the detailed **Doxygen API docs**.

The EGL API consists of 3 major components

- Events: (contained in namespace EGL.Events)
- Sequence Pipeline: (Managed and implemented by Sequence.cs)
- Services: (contained in namespace EGL.Service)

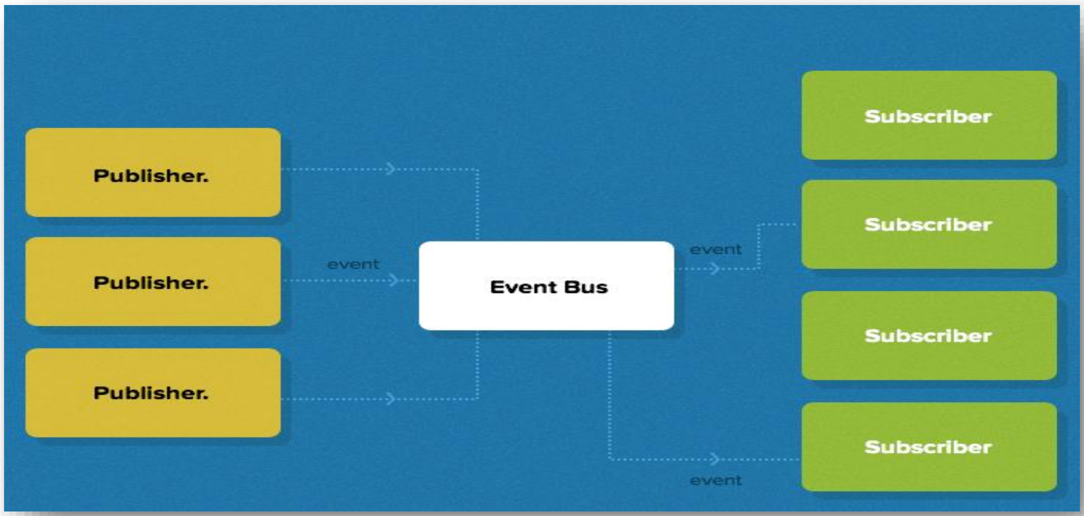
2.1 Events

An Event represents that an action occurred in the EGL API ecosystem. For example, NewDataAvailableOnlineEvent announces availability of new data from online sources and SequenceCompletionEvent announces that the sequence of operations for parsing and processing of the fetched data has been completed.

The Event System in EGL API is based on the Publisher Subscriber/ Observer Pattern.

2.1.1 Publisher Subscriber Pattern

Publisher Subscriber pattern allows to create a push-notification system. Services can subscribe to different events and get notifications when they occur. This is different from the conventional request-response/ polling approach where services would poll a central hub to check if an event has occurred or to enquire about status of different components of the API.



There are 4 main components of a Publisher Subscriber pattern,

1. Event
2. Event Publisher/Announcer
3. Event Hub/ Event Bus
4. Event Subscribers/ Listeners

Event Subscribers subscribe to different events by registering callbacks with the EventHub. Event Publisher announces that an event has occurred by notifying the EventHub about it. The EventHub then invokes the callbacks for all the subscribers that were registered to listen for that event.

Now that the higher level picture of the Event System is clear, let's get into more details:

Event Publisher is represented by any class that implements IEGLEventAnnouncer. IEGLEventAnnouncer describes one function with the signature: public void announce(). The responsibility of announce() is define how to announce the Event, let's say Event XX. Generally, this is done by invoking one of EventHub's announceXXEvent(IEGLEvent eglEvent) functions that corresponds to event XX.

Event Subscriber should implement IEGLEventListener interface and override the notify(IEGLEvent event) function. The responsibility of notify(IEGLEvent eglEvent) is to describe the code (callback) to be executed when a particular event occurs.

2.1.2 EGL API Event System

EventHub is the central hub that glues the entire system together. For each type of Event, it contains 3 functions:

1. annouceXXEvent(IEGLEvent eglEvent) : invokes notify() for all registered EventSubscribers
2. registerXXEventListerer(IEGLEventListener listener): allows Subscribers to register and receive notifications when XX Event occurs.
3. unregisterXXEventListnerer(IEGLEventListener listener) : unregisters listener from receiving notifications about occurrence of XX Event

In EGL API, any class that implements IEGLEvent interface is an event. It should have separate function en EventHub to register listeners, unregister listeners and announce its occurrence.

All classes that define and implement the Event System are located in the **EGL.Events namespace**. Please refer to Doxygen documentation and source code for more detailed explanation and usage examples.

2.2 Sequence Pipeline

The Sequence pipeline takes care of the order in which different events occur i.e. it is responsible for sequencing the Events. The pipeline can be broken down into the following stages

1. Data Fetch
2. Data Parse
3. Data Processing
4. Data Test/ Validation
5. Sequence Completion

The order in which different stages occur is as follows

1. Offline Data Fetch (occurs only once; on application boot)
2. Offline Data Parse (occurs only once; on application boot)
3. Online Data Fetch
4. Online Data Parse
5. Data Processing
6. Data Test/ Validation
7. Sequence Completion
8. Repeat from 3

Each stage consists of Events that can be associated with it.

Order in Sequence	Pipeline Stage	Associated Events
1	Offline Data Parse	**OfflineParsingTier0SiteCompletionEvent**, **OfflineParsingTier1SiteCompletionEvent**, **OfflineParsingTier2SiteCompletionEvent**, **OfflineParsingAllSitesCompletionEvent**
2	Online Data Fetch	**NewDataAvailableOnlineEvent**
3	Online Data Parse	**ParsingDataLinkCompletionEvent**, **ParsingDataTransferCompletionEvent**, **ParsingProductionJobCompletionEvent**, **ParsingSiteCapacityCompletionEvent**, **ParsingSiteCompletionEvent**, **ParsingSitePledgeCompletionEvent**, **ParsingSiteTopologyCompletionEvent**
4	Data Processing	**ActiveSitesDataProcessingCompletionEvent**, **REBUSDataProcessingCompletionEvent**
5	Sequence Completion	**ActiveSitesDataProcessingCompletionEvent**

The sequence logic is implemented by Han.cs and Sequence.cs. Please read up Doxygen docs and source code for these classes.

2.2.1 Data Fetch Stage

2.2.1.1 Data Sources

Data is fetched from the following sources:

Mode	Name	Maintainer	Location
Offline	Kathy Noble's KML	Kathy Noble	Offline Location :/Assets/Resources/offlineSites2016.xml Online Location: https://www.google.com/maps/d/viewer?mid=zjqJjT4W0LqY.kr0ms1czctbw
Online	Google Earth KML	Edward	`http://dashb-earth.cern.ch/dashboard/dashb-earth-all.kml`
Online	REBUS JSON: Capacity JSON	Edward	`https://wlcg-rebus.cern.ch/apps/capacities/sites/ALL/2016/9/json`
Online	REBUS JSON: Pledge JSON	Edward	`https://wlcg-rebus.cern.ch/apps/pledges/resources/2016/all/json`
Online	REBUS JSON: Topology JSON	Edward	`https://wlcg-rebus.cern.ch/apps/topology/all/json`

2.2.2 Pledge JSON Anomaly and Resolution

Here is a snippet from Pledge JSON indicating the type of JSON objects contained in it

```
{
        "PledgeType": "CPU",
        "Federation": "CH-CERN",
        "Country": "Switzerland",
        "PledgeUnit": "HEP-SPEC06",
        "ALICE": 215000,
        "ATLAS": 257000,
        "LHCb": 51000,
        "Tier": "Tier 0",
        "CMS": 317000
}
```

It is important to note that Pledge JSON does not tell us with what WLCG Site the Pledge object is associated with. We only get information about the Federation and Country for a *SitePledge*. Currently, during the data processing stage, the *REBUSDataProcessor* finds *Site* with matching country and federation values and associates *SitePledges* to it. In simpler words, individual pledge data for every site is not available and we are reverse-engineering this information from the country and federation values.

This means, if a WLCG Federation contains more than 1 Site then all of them get the same values of SitePledge.

However, on manual scanning, I was not able to find any federation with more than 1 Site. So we can assume the data is not repeated for any Site. Please confirm this with the IT Department.

2.2.3 FetchedDataHolder vs WineCellar

WineCellar contains latest processed data while *FetchedDataHolder* contains unparsed and unprocessed data after from all the online sources i.e. Google Earth KML and REBUS JSON. So for visualization purposes, only *WineCellar* has significance. Wines taste better with age, so, data in *WineCellar* appears late i.e after Data Processing Stage and contains more structured form of the data contained in *FetchedDataHolder* (populated after Data Fetch stage).

2.3 Data Parse Stage

During this stage, data fetched from different sources is parsed into their respective Data Structures.

Data Being Parsed	Data Structure Parsed into	Parser	Data Source
All Tier-0, Tier-1, Tier-2 Sites	List<Site>	AllSitesKMLParser	Kathy Noble's KML (available offline, occurs only once)
Active Sites (Tier-0 to Tier-3)	List<Site>	SitesParser	Google Earth KML
Site Capacities	List<SiteCapacity>	SiteCapacityParser	REBUS JSON
Site Topology	List<SiteTopology>	SiteTopologyParser	REBUS JSON
Site Pledge	List<SitePledge>	SitePledgeParser	REBUS JSON
Data Links	List<DataLink>	DataLinksParser	Google Earth KML
Data Transfers	List<DataTransfer>	DataTransfersParser	Google Earth KML
Production Jobs	List<ProductionJob>	ProductionJobsParser	Google Earth KML

Data Parsing is done in parallel via coroutines. The identically coloured cells in "Data Being Parsed" column represent that these parsing operations are performed in parallel.

2.3.1 JSON Parsing with WebGL status

We are currently using the Unity ported version of Newtonsoft JSON.Net as the JSON library to parse REBUS JSON. For Standalone builds, JSON.NET works perfectly. However, on implementation, this library turned out to be incompatible with WebGL builds and issued errors during the parsing stage. To ensure similar compatibility across all platforms, the JSON features have been disabled (**NOT removed**) for now.

To enable JSON Parsing and processing, the following changes need to be made:

1. Uncomment the following lines in Sequence.cs:

```
public void ParseFetchedData(ref FetchedDataHolder holder)
{
    state.Add(SequenceState.PARSING);
    ParseXML(holder.xmlDocument);
    //parserObject.AddComponent<SiteTopologyParser>().Parse(holder.topologyJSON);
    //parserObject.AddComponent<SitePledgeParser>().Parse(holder.pledgeJSON);
    //parserObject.AddComponent<SiteCapacityParser>().Parse(holder.capacityJSON);
}
```

2. Implement the new JSON parsing library (WebGL compatible) in ParseLogic() function of SiteTopologyParser.cs, SitePledgeParser.cs and SiteCapacityParser.cs
3. In SequenceCompletionEventAnnouncer.cs uncomment the conditions to include JSON parsing as a parameter for determining completion of a sequence.

```
        private void checkForAnnouncement()
        {
            if (dataTransferComplete && dataLinkComplete && siteComplete &&
productionJobComplete && activeSiteDataProcessingComplete) //&& siteTopologyComplete &&
sitePledgeComplete && siteCapacityComplete && rebusDataProcessingComplete)
                announce();
        }
```

2.4 Data Processing Stage

Data Processing stage involves processing of parsed data and making it available to different services so that it can be used by the visualization engine.

2.4.1 Simple Learning Algorithm to detect Tier-3 sites.

Information for Tier-3 sites is not available directly from any of the data sources.

Data Source	Information Contained
Kathy Noble's KML	All Tier-0 Sites, All Tier-1 Sites, All Tier-2 Sites
Google Earth KML	Active Tier-0 Sites, Active Tier-1 Sites, Active Tier 2 Sites, Active Tier-3 Sites

The only source of Tier-3 sites is Google Earth KML and it only gives information about ACTIVE Tier-3 Sites. So, by default, there is no way to detect inactive Tier-3 sites.

EGL API overcomes this limitation as follows: When EGL starts, it first adds the Active Tier-3 Sites to *WineCellar.allTier3Sites* from the current version of Google Earth KML. Then after parsing the next version of Google Earth KML, EGL API adds the NEW Active Tier-3 Sites to *WineCellar.allTier3Sites* and also keeps the Sites that became inactive in this version of Google Earth KML i.e. *WineCellar.allTier3Sites* List keeps Tier-3 Sites even after they become inactive.

Therefore, once an Active Tier-3 Site is discovered by EGL, it will remain in the system unlike other Tier-0, Tier-1, Tier-2 counterparts which are always present as they are already provided by Kathy Noble's KML.

This also means that, the longer EGL runs, the more Tier-3 sites it might display!

The source code implementing this logic has been documented in ActiveSitesDataProcessor.initMergeSites();

2.4.2 Redundancy checks after merging Google Earth KML and Kathy Noble's KML

We maintain a master list of Sites in *WineCellar* (static member variable names: *allTier0Sites*, *allTier1Sites*, *allTier2Sites* and *allTier3Sites*). These Lists always contain the **latest parsed and processed information** about the Sites from all the data sources. To achieve this, every time a sequence is executed, data from multiple sources is merged with the data contained in the lists. The **Site Name** is used as **the primary key** for detecting what entities are to be merged. The format for Site Names is derived from the Site Names used by Google Earth KML. Here are few examples: AGLT2, AM-04-YERPHI, CERN-PROD etc

So, if the site names for the same site do not match in different data sources, there will be two or more entries for the same site. Luckily, Google Earth KML and REBUS JSON's use the same format for Site Names as they are maintained by Edward. Kathy Noble's KML, however, uses a slightly different format. Here are few Site Names from Kathy Noble's KML: US | AGLT2, CH | CERN Data Centre, Tier-0 etc.

The *AllSitesKMLParser.cs* takes care of converting Kathy Noble's Site Names to standard Google Earth Site Names. Initially, we observed 2 site markers at CERN data center during the visualizations. This was because even after converting Kathy Noble's format to Google Earth KML Site Name format, CH | CERN Data Centre, Tier-0 could not be matched to CERN-PROD.

To resolve this, we have an exceptional case for Tier-0 CERN Site. In case there are more exceptional cases, please make an addendum to the code in *AllSitesKMLParser.cs* to reflect it.

To easily check for redundancies in Site Data, *EGL.Tests.EGLWebAndDataStatusTest* contains a function *List<Site> siteRedundancyTest()*. This function checks for redundant Site data and prints the redundant sites to the Unity Debug Console. To enable this test, set *Config.RUN_TESTS = true.*

```
/// <summary>
/// If true, <see cref="Service.EGLWebAndDataStructureService"/> will execute all tests
specified in the <see cref="Service.EGLWebAndDataStructureService.initTests()"/>
functions.
/// </summary>
public static bool RUN_TESTS = true;
```

2.5 Services

Services in EGL API abstract the Sequence Pipeline and Event System and provide a clean interface to interact with the EGL API. It is the preferred way to communicate with EGL API when you do now want to get involved in too much technical details about implementation of Sequence Pipeline and Event System.

Please read up Doxygen documentation and the source code for more details on how you can use individual services. They are contained in the *EGL.Service* namespace.

Some of the mail services are described in brief here:

2.5.1 EGLWebAndDataStatus

EGLWebAndDataStatus service provides an alternative to Event System to enquire about the state of things in EGL. While EGL Event System's design is based more on a push notification mechanism i.e. subscribers automatically get notifications when an event occurs, *EGLWebAndDataStatus's* design is inspired from a request-response or polling mechanism. *EGLWebAndDataStatus* can be pinged to check if an event has occurred yet or not at any point of time during program's execution.

EGLWebAndDataStatus contains static Boolean variables that correspond to each event described by the Event System. If the value of any variable is true, it means that the corresponding event has occurred. The usage and the different fields have been documented in Doxygen.

This service also provides the **parsing status for Google Earth KML** and Kathy Noble's KML in terms of percentages. The parsing status is exposed through the following functions:

Return Type	Function Name	Default Denominator	Data Source
Float	getActiveSitesParsedPercentage()	Config. DEFAULT_ACTIVE_SITES_NUMBER	Google Earth KML
Float	getDataTransfersParsedPercentage()	Config. DATA_TRANSFER_DEFAULT_NUMBER	Google Earth KML
Float	getDataLinksParsedPercentage()	Config. DATA_LINK_DEFAULT_NUMBER	Google Earth KML
Float	getProductionJobsParsedPercentage()	Config. PRODUCTION_JOB_DEFAULT_NUMBER	Google Earth KML
Float	getAllSitesParsedPercentage()	Config.DEFAULT_TOTAL_SITES_NUMBER	Kathy Noble’s KML

2.5.1.1 Estimation of Parsing Percentages

Parsing percentage from different sources is calculated as follows:

$$Parsing\ Percentage = \frac{Number\ of\ records\ parsed}{Total\ Number\ of\ Records} * 100$$

The denominator i.e. *Total Number of Records* cannot be determined before actually parsing all the data. Therefore, the parsing percentage is approximated by using the *Total Number of Records* from the previous parsing operation. This is done on the assumption that *Total Number of Records* for any of the data sources will not vastly change in the 10-minutes after which a new data source becomes available. When EGL first starts, the values to be used in the denominators are set *by static int variables* defined in the *Config* class (see *Default Denominator* column in the above table). In order to account for any discrepancies that may occur, the percentages automatically become 100% at the end of the parsing operation.

2.5.2 DaVinciPallette

DaVinciPalette provides convenience functions that are used by DaVinci to plot data. In lay man terms, it is the glue linking EGL API to the visualization engine. The idea is that *DaVinciPallete* should abstract the entire functionality of EGL API from the visualization engine. It should just provide functions that return the data needed by *DaVinci* or the visualization engine.

In the current implementation, *DaVinciPallete* contains a field *wineCellar*. **This WineCellar object is auto populated with all the latest processed data.** Therefore, if you want to just use the processed data anywhere during visualizations, you can create a new function in DaVinciPallete that returns the data from wineCellar (which is a private member variable by default, so it can’t be directly accessed).

2.5.3 EGLWebAndDataStructureService

This service is the main initializer for EGL API. It sets everything up ranging from configuring EGL API, starting Event System and Services etc. It is very important to initialize EGL through this Service before using any other components of EGL API. More details on this service and it’s usage are documented in the Doxygen docs.