Version

Nicolas A. Barriga, Carlos Quiroz, Arturo Núñez

27052011/01

Created: May 27, 2011

Modified: May 30, 2011



**Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Notes** |
| 1 | May 27, 2011 | Nicolas A. Barriga | * First Version |

Table of Contents

1 Introduction 1

1.1 Document Purpose 1

1.2 Intended Readership 1

1.3 Conventions 1

1.4 Acronyms 1

1.5 Reference Materials 1

2 Reasoning behind GDS 3

3 Overview of the Gemini Data Service 3

3.1 Data Values Composition 4

3.2 Observation Event driven data collection 4

3.3 Completion of FITS files 5

3.4 Sending FITS Headers from Seqexec to GDS 5

4 Data Label Generation 5

5 GUI 6

6 Implementation Details 10

6.1 Protocols 11

6.2 Scala 12

7 Operational aspects 12

7.1 Startup/Shutdown procedure 12

7.2 Logging 12

7.3 Troubleshooting 13

# Introduction

This document provides a description of the Gemini Data Service (GDS) Design.

## Document Purpose

The purpose of this document is to present the GDS design for review by interested parties.

This document does not rehash the critical information in references and . It is assumed that the reader has read these other documents and understands their content.

## Intended Readership

The intended audience for this document is *groups who are writing software, design review documents or providing operational support* *for Aspen instruments*.

## Conventions

The GDS is still under active development and things that are expected to undergo some changes are marked like this paragraph with a yellow exclamation point. There are not many of these situations in this document.

## Acronyms

DHS Data Handling System

GIAPI Gemini Instrument Application Programmer Interface

GMP Gemini Master Process

GSDN Gemini Data Storage Network

ICD Interface Control Document

JMS Java Message Service

GDS Gemini Data Service

ODB Observing DataBase

XMLRPC eXtensible Markup Language Remote Procedure Call

RMI Remote Method Invocation

EPICS Experimental Physics and Industrial Control System

FITS Flexible Image Transport System

GDSN Gemini Data Service Network

## Reference Materials

x

|  |  |
| --- | --- |
| [1] | Kim Gillies and Arturo Nunez, "Aspen GIAPI Design and Use," 2006. |
| [2] | Kim Gillies, "Guidelines for Designing Gemini Aspen Instrument Software," 2004. |
| [3] | Arturo Núñez and Kim Gillies, "ICD50: GIAPI C++ Language Glue API," Gemini Observatory, 2008. |
| [4] | Nicolas A. Barriga, Carlos Quiroz, and Arturo Núñez, "ICDXX: GDS to Gemini ICD," Gemini Observatory, 2011. |
| [5] | FITS Standard Specification. [Online]. http://archive stsci.edu/fits/fits\_standard/fits\_standard.html |
| [6] | XML-RPC Home Page. [Online]. <http://www.xmlrpc.com> |
| [7] | Norman Hill, Dayle Kotturi, Severin Gaudet, Steve Cockayne, and Jennifer Dunn, "ICD 3.2 -- The DHS Interface," Gemini Observatory, 1998. |
| [8] | Norman Hill and Severin Gaudet, "ICD 3 -- Bulk Data Transfer," Gemini Observatory, 1996. |
| [9] | Steven Beard, "ICD 1.9/3.2 Science Instrument to Data Handling System," 1997. |
| [10] | Experimental Physics and Industrial Control System. [Online]. <http://www.aps.anl.gov/epics/> |
| [11] | Java Message Service. [Online]. <http://en.wikipedia.org/wiki/Java_Message_Service> |
| [12] | OSGi Alliance. [Online]. <http://www.osgi.org> |
| [13] | Jini.org. [Online]. <http://www.jini.org> |
| [14] | The Scala Programming Language. [Online]. <http://www.scala-lang.org/> |
| [15] | Apache Foundation. (2011) Apache Felix. [Online]. <http://felix.apache.org/site/index.html> |
| [16] | Java Logging Overview. [Online]. <http://download.oracle.com/javase/1.4.2/docs/guide/util/logging/overview.html> |
| [17] | Pax-Logging. [Online]. <http://wiki.ops4j.org/display/paxlogging/Pax+Logging> |
| [18] | Apache Log4J. [Online]. <http://logging.apache.org/log4j/1.2/index.html> |
| [19] | Arturo Núñez, "GIAPI/OCS Plan," Gemini Observatory, 2005. |

x

# Reasoning behind GDS

* DHS too complex
* Difficult to maintain and modify.
* Compatibility with GIAPI
* Requirements of GIAPI-based Instruments

# Overview of the Gemini Data Service

The Gemini Data Service (GDS) is a system that will (partially) replace the Data Handling System (DHS) for GIAPI based instruments, such as GPI. The GDS is basically a software entity that collects a set of relevantdata items from different subsystems in the observatory, and use these items to supplement the FITS file generated by a GIAPI-based instrument. The GDS configuration will drive the data collection indicating which hata items need to be collected, the observatory system they come from and where they will be stored in the final FITS file. The GDS listens for Observation Events generated by the instrument (as specified in the GIAPI Specification[[1](#Kim1)]) and collects data items for each of these events.

Conceptually the GDS is very simple and its features have been kept to a minimum to provide an efficient and maintainable service. Nevertheless, It has been designed with extensibility in mind and can be extended, for example, if new subsystems need to be added/removed.

From the point of view of the observatory systems, the GDS is located between the instrument and the rest of the observatory and the resulting FITS files will go into the GDSN. This is depicted in the diagram below.

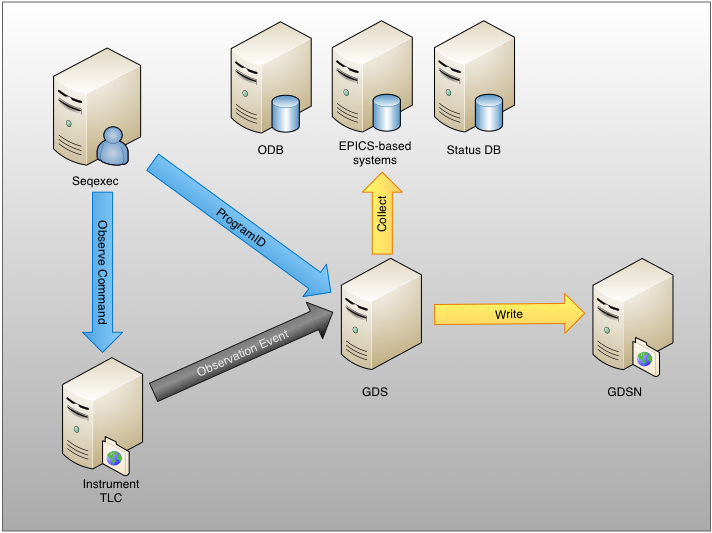


Figure : GDS System Diagram

GDS will perform the folloming duties:

* Listen for observation events from the instrument
* Gather data from observatory systems
* Accept external input from systems that cannot be queried externally like Seqexec
* Complete the FITS files with data collected during an observation

These duties are explained in the sections below

## Data Values Composition

The GDS is a component that listen for Observation Events sent from an instrument and that reacts to them by sampling the state of the Observatory, observation database, instrument, etc. The sampled information is then added to the data files produced by the instrument. This process is called value composition, where all sampled values upon observation events are composed into the data file.

The design uses the concept of actors or agents that are independent objects that can sample the information. This design makes it simple to keep track of the progress and make each of these actors, a single purpose, very simple to implement component.

It also makes it easy to extend the composition as the core components can discover new modules in charge of gathering these values

## Observation Event driven data collection

In order to perform a timely data collection, the instrument must keep the GDS updated on the status of the current observation. This is done through Observation Events. The following table summarizes them all:

|  |  |  |
| --- | --- | --- |
| **Event Time** | **GIAPI OEvt** | **Description** |
| T1 | OBS\_PREP | Observation Event sent as instrument starts preparation for starting acquisition of a dataset. |
| T2 | OBS\_START\_ACQ | Observation Event sent just before data acquisition starts. |
| T3 | OBS\_END\_ACQ | Observation Event sent when the requested acquisition has completed. |
| T4 | OBS\_START\_READOUT | Observation Event indicates that the data is being transferred from the detector or other activities needed to write data. |
| T5 | OBS\_END\_READOUT | Observation Event indicates readout or write preparations have completed. |
| T6 | OBS\_START\_DSET\_WRITE | Observation Event indicates that the instrument has started writing the dataset to the GDSN. |
| T7 | OBS\_END\_DSET\_WRITE | Observation Event indicates that the instrument has completed writing the dataset to GDSN. |

When receiving an Observation Event, the GDS will collect the appropriate FITS keyword data, according to its configuration.

## Completion of FITS files

GIAPI based instrument store their FITS files directly into the disk instead of through the DHS as legacy instruments do. However, since the Instrument has a limited access to the observatory subsystems, those FITS file are missing critical metadata in their FITS files.

Once the instrument completes the FITS files, GDS will store the extra metadata as collected during the observation. It will first create a copy of the original FITS file, add the new headers and store the resulting file in the GDSN.

The original file is never deleted and remains as a backup. Likewise, the GDS never touches the data or existing keywords and will only add now values collected during the observation.

## Sending FITS Headers from Seqexec to GDS

The FITS headers that the seqexec provides, must be passed to the GDS. There aren't many suitable remote communication alternatives supported both by tcl and java/scala. The most suitable seems to be XMLRPC, which has implementations in both languages, and is relatively simple, but much higher level than using plain http or tcp.

# Data Label Generation

Currently, the dataset names (or datalabels) generation is performed via a control command by the DHS, at the request of the seqexec. The seqexec later specifies who will contribute data to this dataset, at which point itself and the contributors can start sending data. A first study of the following documents and code was done:

* ICD 3: Bulk Data Transfer
* ICD 1.9/3.2: Science Instrument to Data Handling System
* dhs/dhs/dhsData/list.C
* dhs/dhs/dhsData/ctl.C

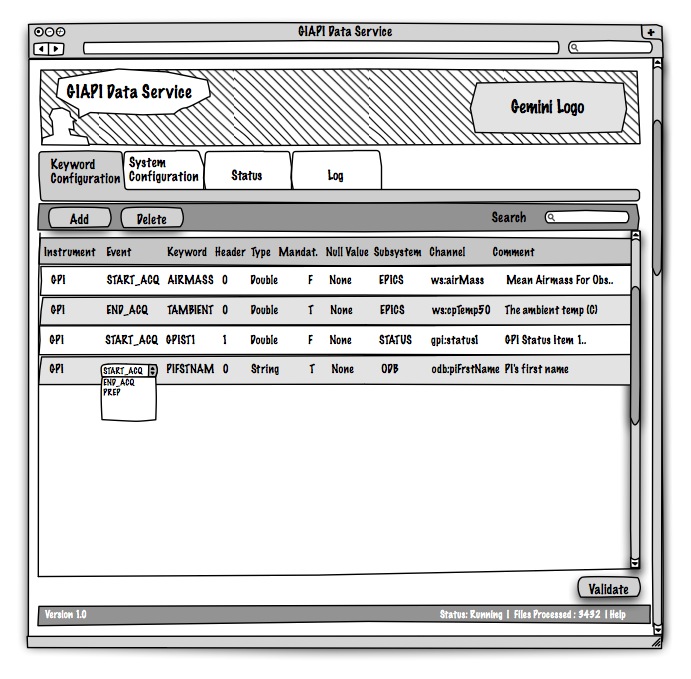
This investigation shows that the datalabel generation functionality is fairly independent from the data storing functionality, in a way that no files are created, and no internal state changes (except for a list of the last labels generated), when labels are generated. Furthermore, data can be sent with arbitrary datalabels not generated by the DHS.

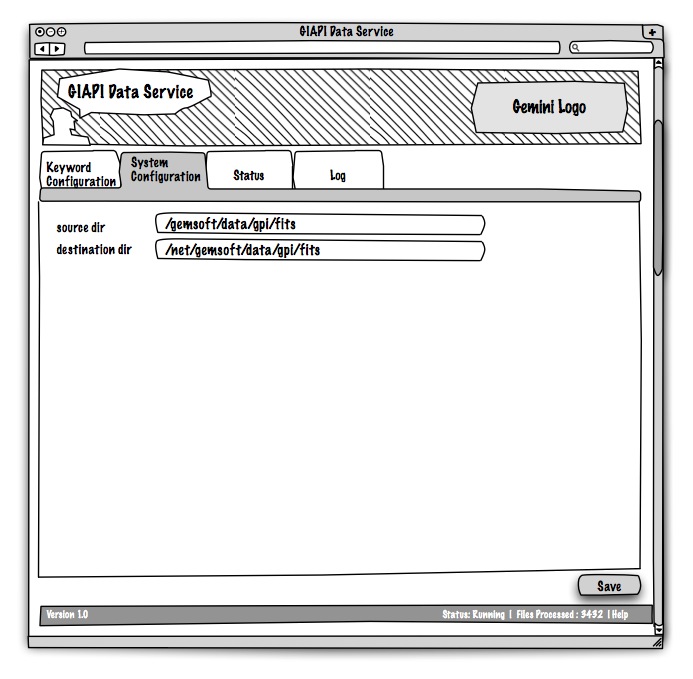
So, there are three main alternatives for instruments using the GDS:

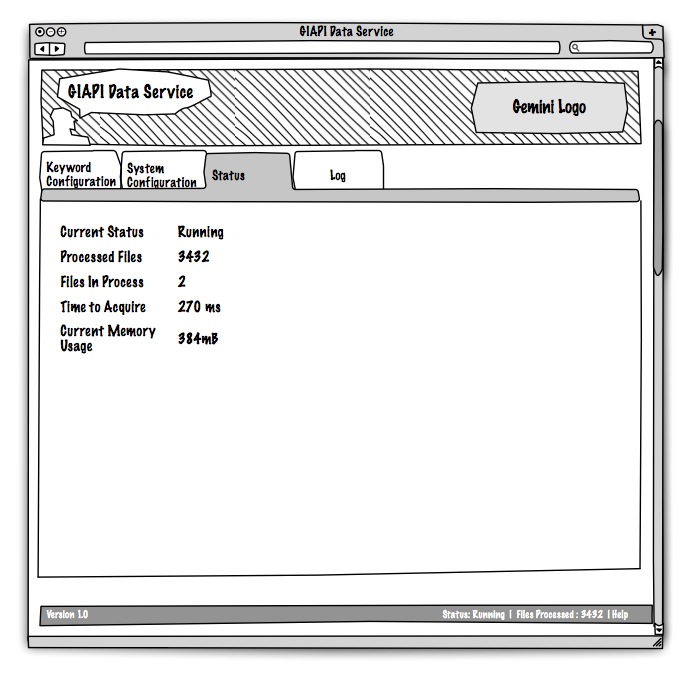
1. Continue with the seqexec requesting datalabels to the DHS, and then not sending any data to the DHS, but instead sending it to the GDS. The major disadvantage is a dependence on the DHS for GIAPI based instruments.
2. Extract the datalabel generation from the DHS to an external service that the DHS can query for most instruments, and the seqexec can query for GIAPI based instruments. The major disadvantage is the risk of modifying a complex piece of software like the DHS. For consistency and simplicity, if this option is taken, the communication with the new external service should use the same protocol as the one used in Sending FITS Headers from Seqexec to GDS.
3. Generate datalabels independently in GDS and DHS, and ensure no collisions will happen by changing the naming convention.

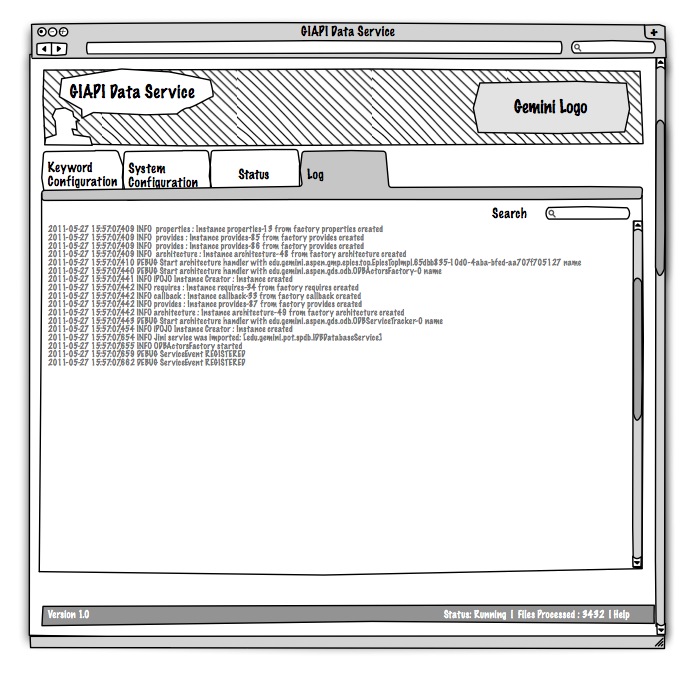
# GUI

These are mockups of the proposed GDS user interface.









# Implementation Details

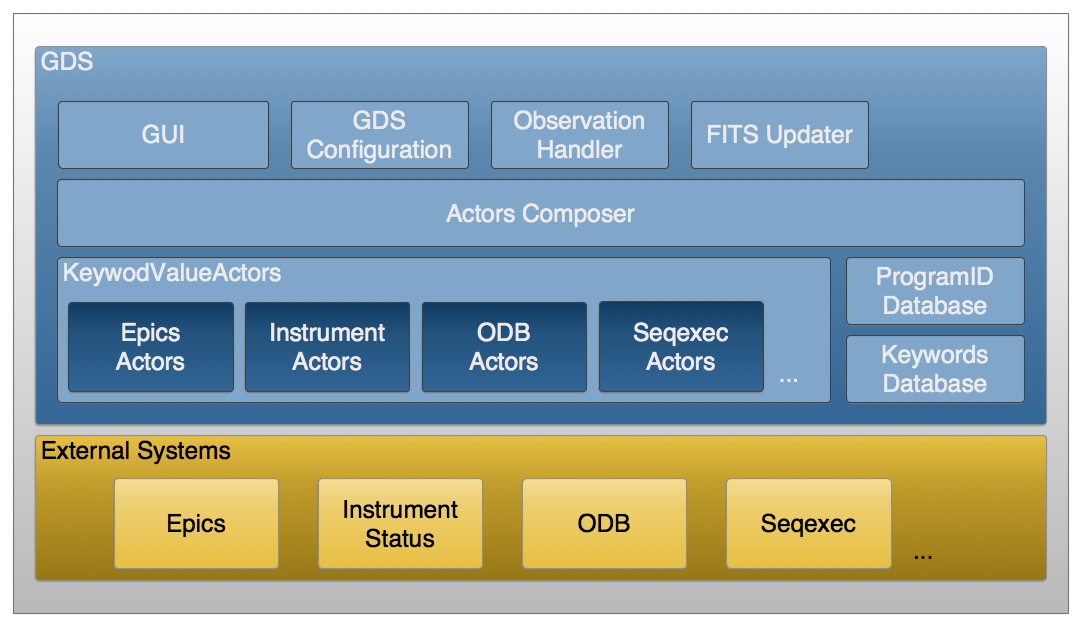
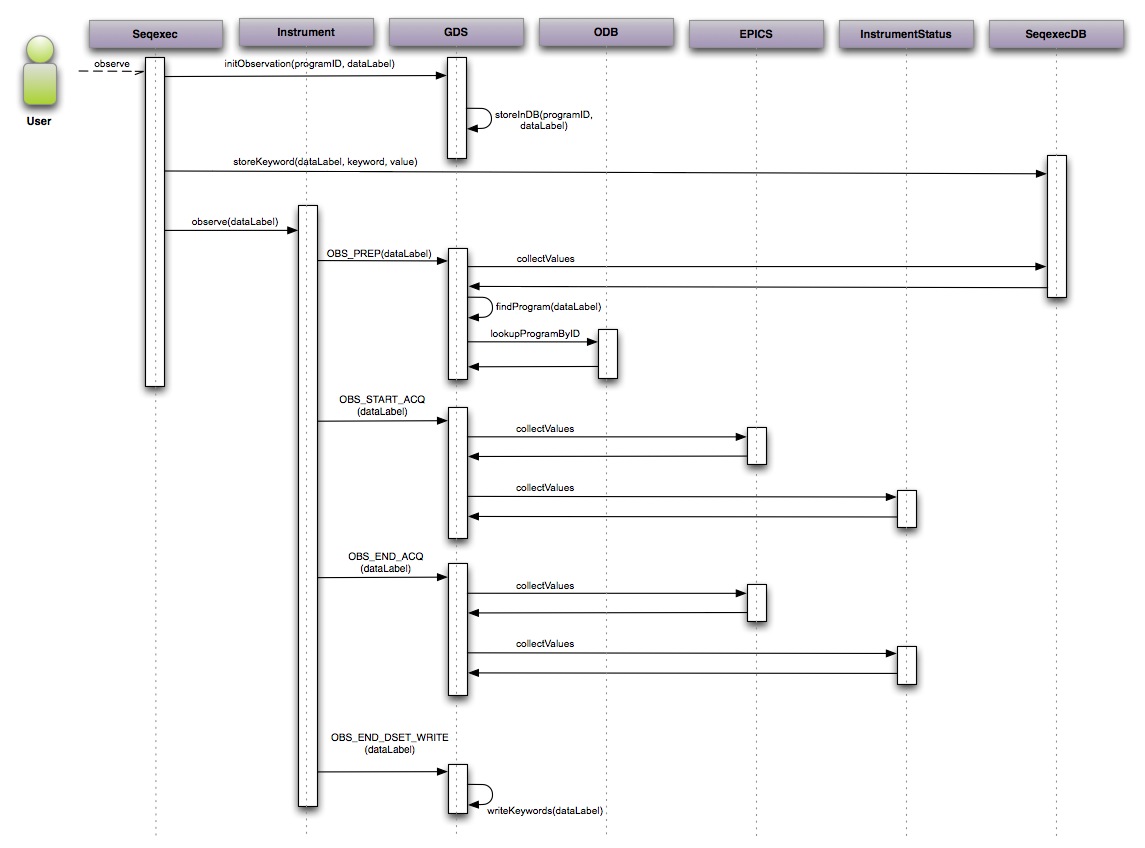


Figure : GDS Module Diagram

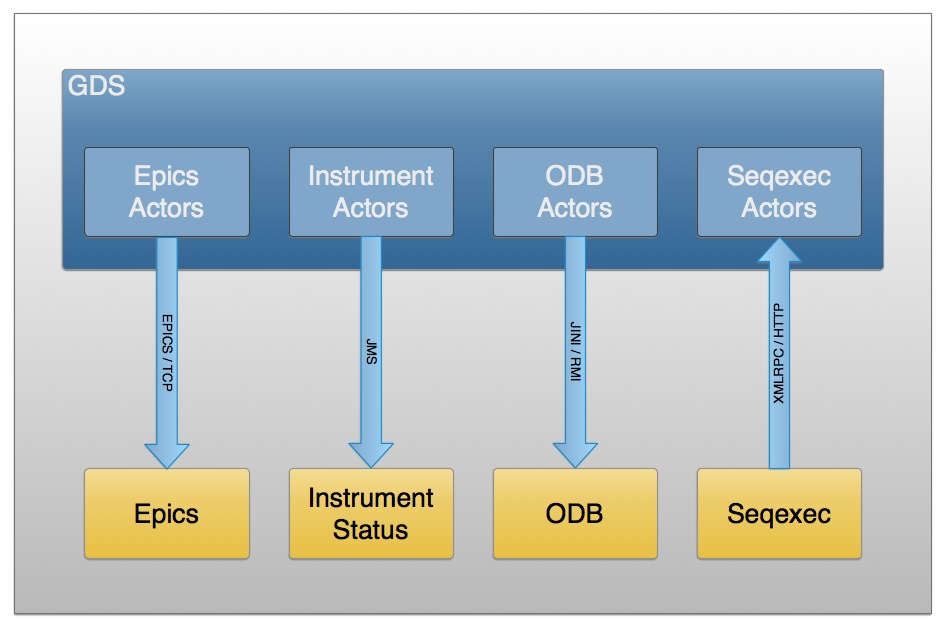
The following sections are of interest only to software engineers who need to maintain the GDS code base.



## Protocols

The GDS uses several protocols to communicate to different external systems. Most of these were chosen because current or legacy applications use them.

* EPICS: Currently used by most instruments and the OCS. The GDS needs to collect information such as weather data, which is accessible via EPICS.
* JMS: GIAPI based instruments publish status information via JMS. This information is received by the Gemini Master Process (GMP), and is accessible by the GDS via OSGi.
* Jini/RMI: Used by GDS to query the ODB for static information of programs, such as principal investigator’s names.
* XMLRPC: Used by the seqexec to communicate keywords to the GDS.



## Scala

The GDS is mostly written in Scala . Scala was chosen because of several reasons:

* Compiles to bytecode, runs on a standard JVM and transparently interoperates with Java.
* Statically typed combined with a type inference mechanism.
* Combination of functional and object-oriented paradigms.
* The actor concurrency model provides a safe way to build concurrent applications that scale easily.

# Operational aspects

## Startup/Shutdown procedure

The GDS runs inside the Felix OSGi container, in the same container that the GMP server is running, though it can also run in a separate instance. Scripts are provided to start and stop GDS under the bin directory of the distribution.

The start.sh script will start Felix with the set of OSGi bundles required for the GDS operation. No other parameters are required at startup. The startup procedure takes a few seconds to assemble all the components and then it becomes ready to accept observation events.

The stop.sh script will request a graceful shutdown of Felix. You need to take into account that GDS has no persistence and so if there are pending data labels in process, they may not be written to disk upon shutdown. No other parameters are required at shutdown.

## Logging

GDS uses the standard Java Util Logging API , but some of the dependencies use other APIs like Log4J, etc. To get a unified view of all the logging, the Pax-Logging OSGi service is used. This service converts all the log entries from any of the supported libraries and puts them in a single log. That log follows the Log4J  conventions and is configured through a Log4J-style configuration file at conf/services/org.ops4j.pax.logging.cfg.

The default configuration will produce a log file rotated daily and stored under the log/ directory. The configuration can be changed to produce different log types, file names and detail. To understand the Log4J configuration file, refer to .

A user interface to the Log is being planned that will allow to look at the contents of the most recent log in the GDS console.

## Troubleshooting

Currently, the main troubleshooting tool is Logging (5.2). There is also a heartbeat published via JMS, that Java clients can subscribe to. If necessary, this heartbeat can also be published via EPICS.

Planned features include a Health Publisher module, in charge of publishing a general status of the GDS, an Observation Error Publisher, that informs of specific errors that might occur when gathering data for an observation, an Administration Service that present this information to the user, plus the ability to change the configuration and to clear error conditions.

The Health Publisher module will reflect general system wide issues, such as low memory, low disk space, network outage, etc.

The Observation Error Publisher focuses on more specific issues, such as not being able to retrieve a given keyword, or missing observation events.