Frame Processing Framework tools for low level CCSDS processing, and projects for cooperative real-time global EO data collection

Presented at CSPP/IMAPP Users Group Meeting by

Alexander Shumilin
RBC Signals

27-29 June 2017, Madison, WI

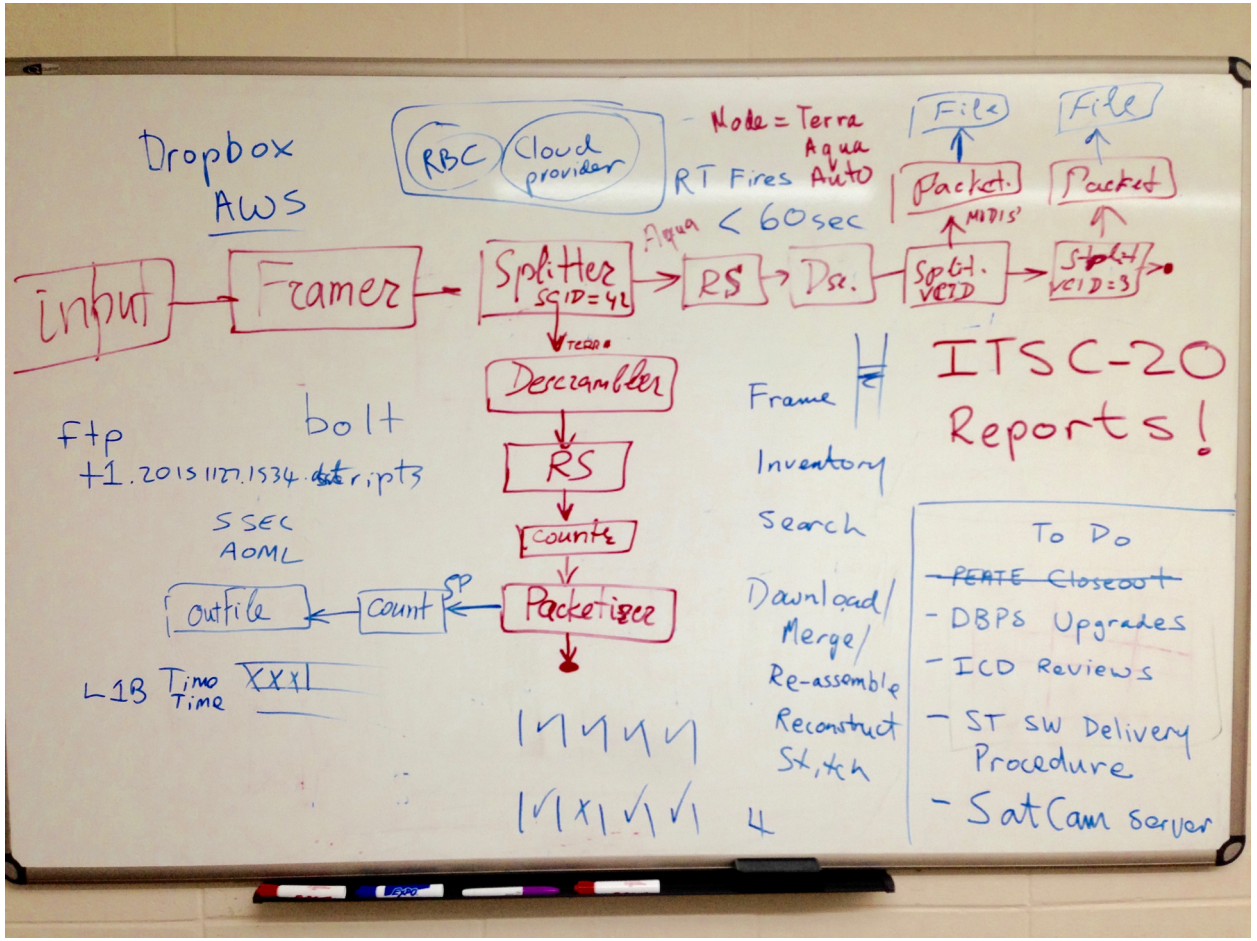
Motivation

- Need for a flexible tool to solve different processing and investigation tasks for different satellite streams.
- Powerful enough to get the task solved
- Extendable. Add new features and function as new challenges arrive – clean modular source structure
- KISS. Keep it simple not compromising above requirements
- Portable and easy to deploy,

- One of the widely used cases:DB streams from AQUA/TERRA/NPP/Metop
- Existing options: RT-STPS (385 *.java files), Metopizer, etc

(In the beginning was the Word...)

Core parts of the FPF were born in CIMSS/SSEC in late 2015.
Thanks to Liam Gumley for kind support and fruitful discussions



author gratefully acknowledges support of

* CIMSS/SSEC UW-Madison
* RBC Signals
* RDC ScanEx

Frame Processing Framework (FPF)

- A **Frame** – a unit of data stream
- All operations done frame-wise
- Frames = CCSDS CADU,
  Source Packet, etc.
- Same type but frame size is not fixed
- Handle of a frame contains
  pointer to the frame data and some attributes:
  - size,type
  - position in the input stream
  - stream timestamps

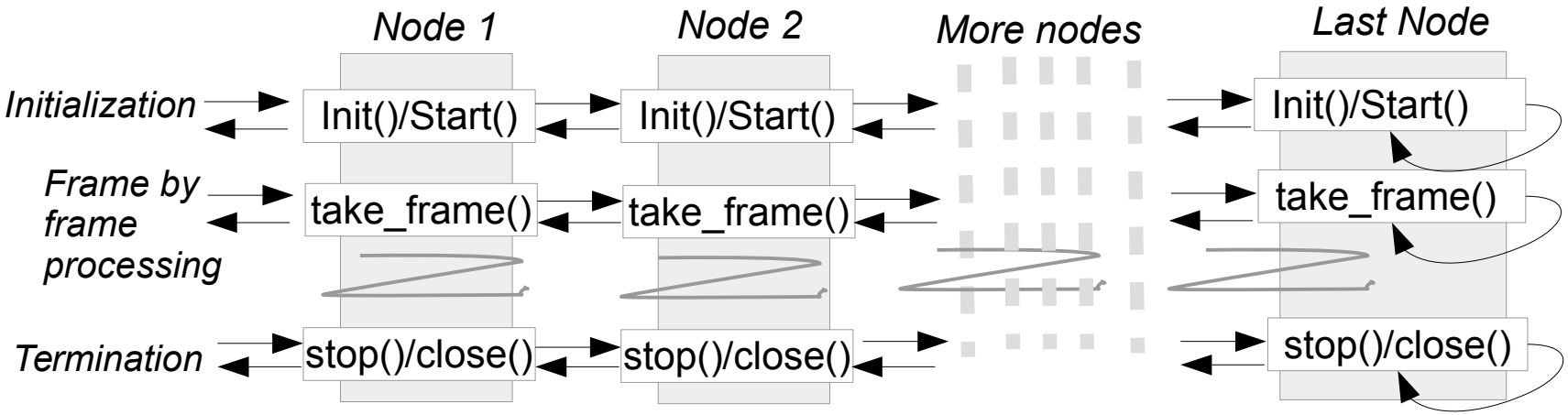
```
class CFrame
{

    public:
        CFrame();
        BYTE* pdata;        //pointer to the frame data buffer
        CStream* pstream;  //pointer to the Stream
        unsigned int frame_size; // frame size
        t_frame_type frame_type; // frame type
        t_stream_pos stream_pos; //position of the frame in the stream
           t_uint32  cframe;     // frame counter as issued by framer
                int  vcid;       // channel ID (VCID,APID,etc)
                int  scid;       // spacecraft or source ID
                int  crc_ok;     // frame passed error correction
                int  bit_errors; //bit errors estimation
             time_t  wctime;     //wall clock time, seconds, unix epoch
          long int   wctime_usec;
             time_t  actime;     //acquisition clock time, seconds, unix epoch
          long int   actime_usec;
             time_t  obtime;     //on-board clock time, seconds, unix epoch
          long int   obtime_usec;

};
```

- Frames are processed by C++ objects (**Nodes**)
  inheriting simple common interface
- Nodes are simple. One node – one elementary
  operation on one incoming fame per call
- A Node after reception of a new input frame can...
  - pass it through to the next object in the chain as is
  - block further processing of the frame
  - cache or buffer a few packets for own needs
  - add/update attributes in the handle
  - modify data (e.g. decode, descramble, add timestamps)
  - output frame content or any extracted information to files/socket
  - create a new kind of frames and initiate another chain of processing objects

```
class INode    // chain node interface class,
{
    public:
        INode();
        virtual ~INode();
        //
        string name;
        string id;
        bool is_initialized;
        INode* pnext_node;  //-> ref. to next Node
        //state control methods
        virtual bool init(t_ini& ini, string& name,
                          CChain* pchain);
        virtual void start(void);
        virtual void stop(void);
        virtual void close(void);
        //frame processing
        virtual void take_frame(CFrame* pf);
    protected:
        CChain* pchain;
};
```

Frames,Nodes and other beasts

- Common interface lets build a chain of objects/nodes
- No single supervisor/governor.
  Chains of nodes are “self-assembled” and “self-managed”



- Helper objects:
  - **Framers** – bake frames at the beginning of the chain, have no input frames, only output. Usually Framer is created first and then it initiate chain self-assembling
  - **Input Objects** – provide raw bit (pre-framed) stream as input to framers (input from files, socket, http)
  - **Node factory** – helps to instantiate node objects at run-time using class names and blocks of configuration parameters
- Main.cpp – the only executable which reads configuration, setup environment and initiate node chain assembling and execution.

Chain configuration sample (“Hello World”)

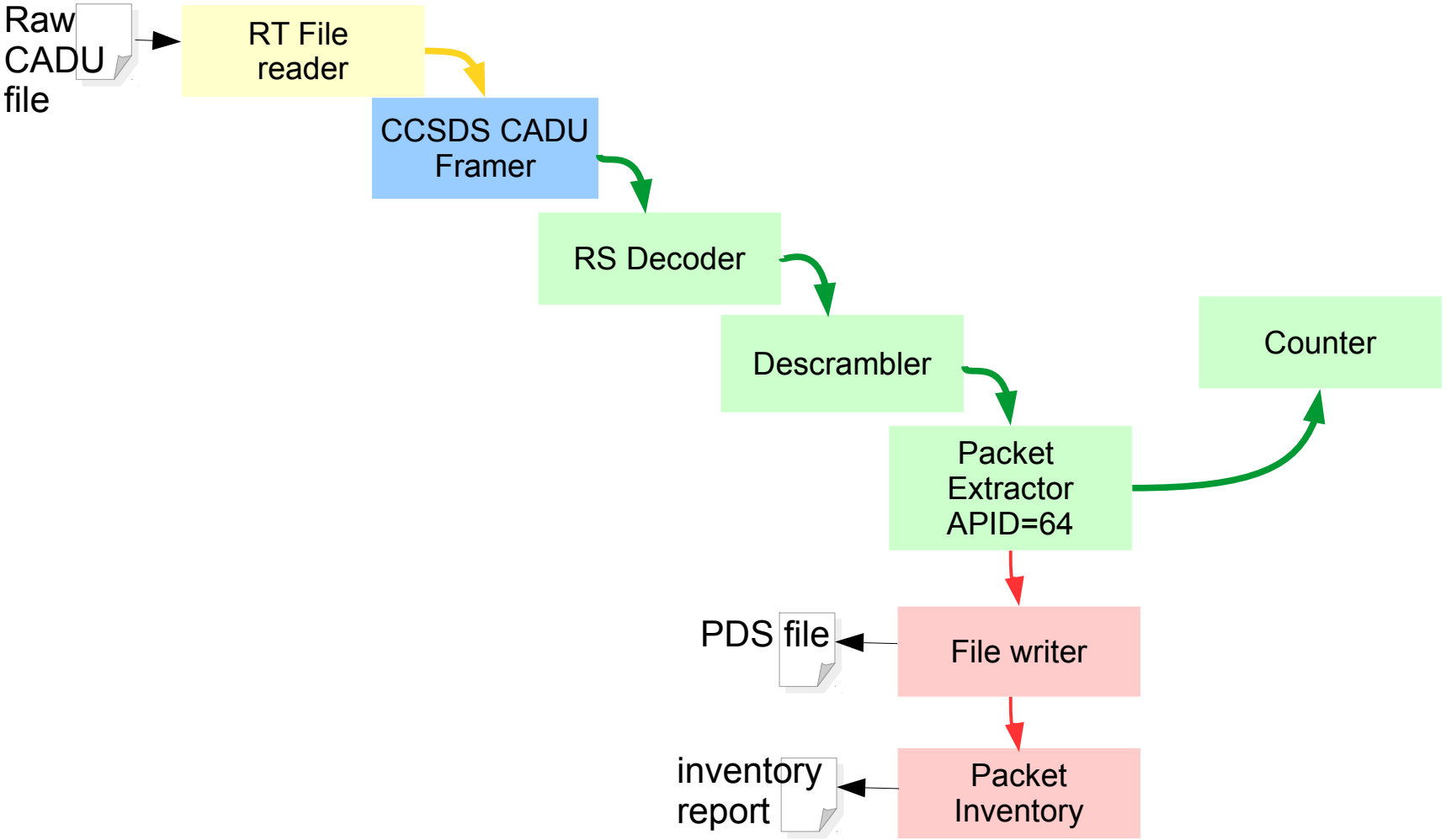
simple_sample.ini

```
# simple FPF chain.
# Read ~100 CADU 1040byte frames,
# dump a counter and frame headers
#
[_main_]
frame_source=CADU_Sync

[FileInput]
class=CInputStream_File
input_file =$INPUTFILE$
start_offset=100000
read_bytes=104000

[CADU_Sync]
class=CFrameSource_CADU
input=FileInput
frame_size=1040
trace_sync=yes
sync_marker=1ACFFC1D
fix_iq=no
discard_fill=no
next_node=Counter0

[Counter0]
class=CNode_Counter
trace_every=10
next_node=CADUDumper

[CADUDumper]
class=CNode_CADUdump
dump_every=1
format=line_long
#next_node=***
```

- Plain text INI-style chain configuration files
- One object – one INI section
- Common values (part or “The Rules”):
  - section name – is ID of the object
  - *class*= - C++ object class name
  - *next_node*= - ref. to ID of the next object in chain
- All other parameters are custom and interpreted by the object
- Parameter values may be substituted at runtime by values from
  - environment variables
  - command line arguments

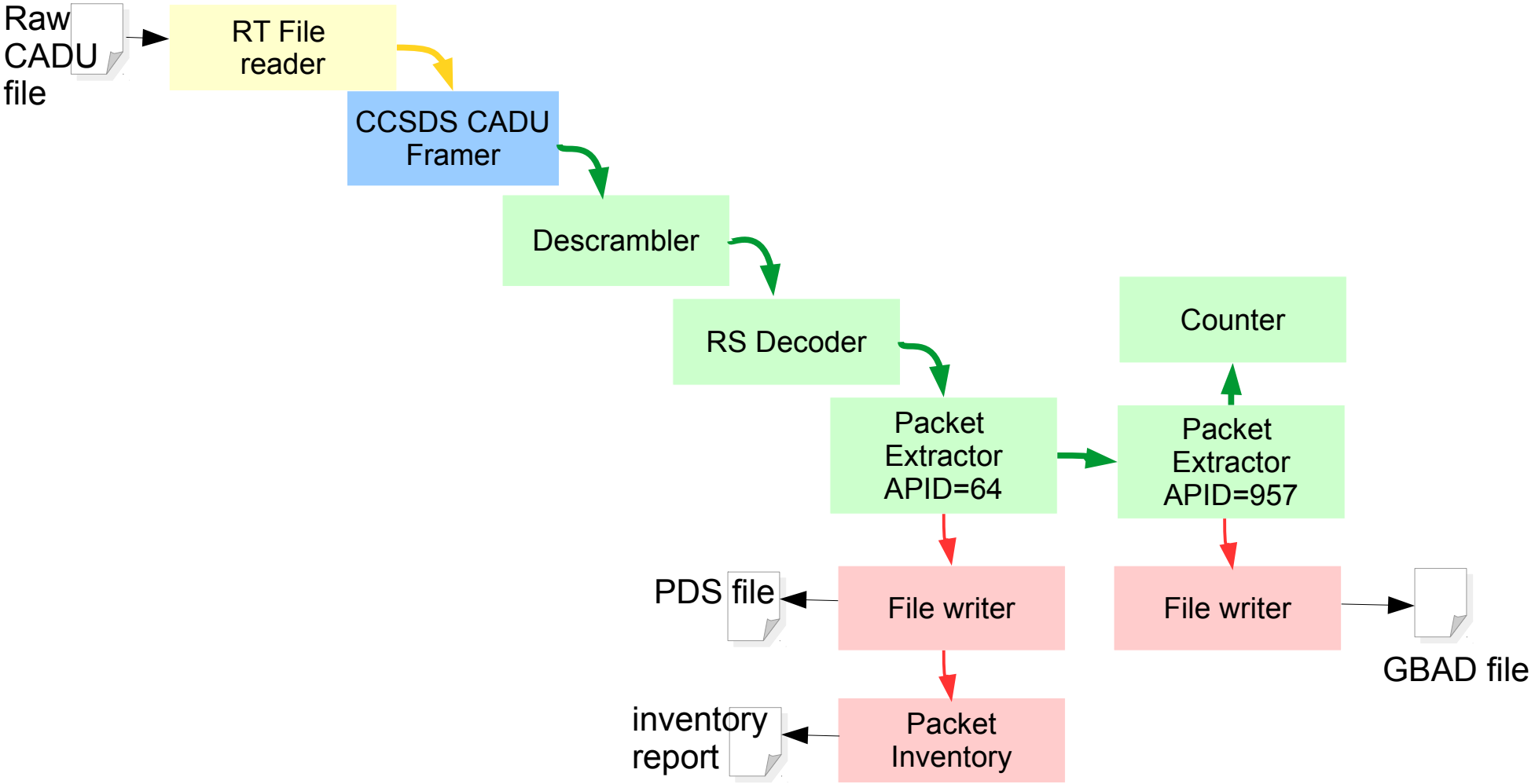
Real world processing sample. 1

Sample processing case: **TERRA** MODIS PDS extraction from raw CCSDS CADU stream.



Real world processing sample. 2

Sample processing case: **AQUA** MODIS PDS/GBAD extraction from raw CCSDS CADU stream.

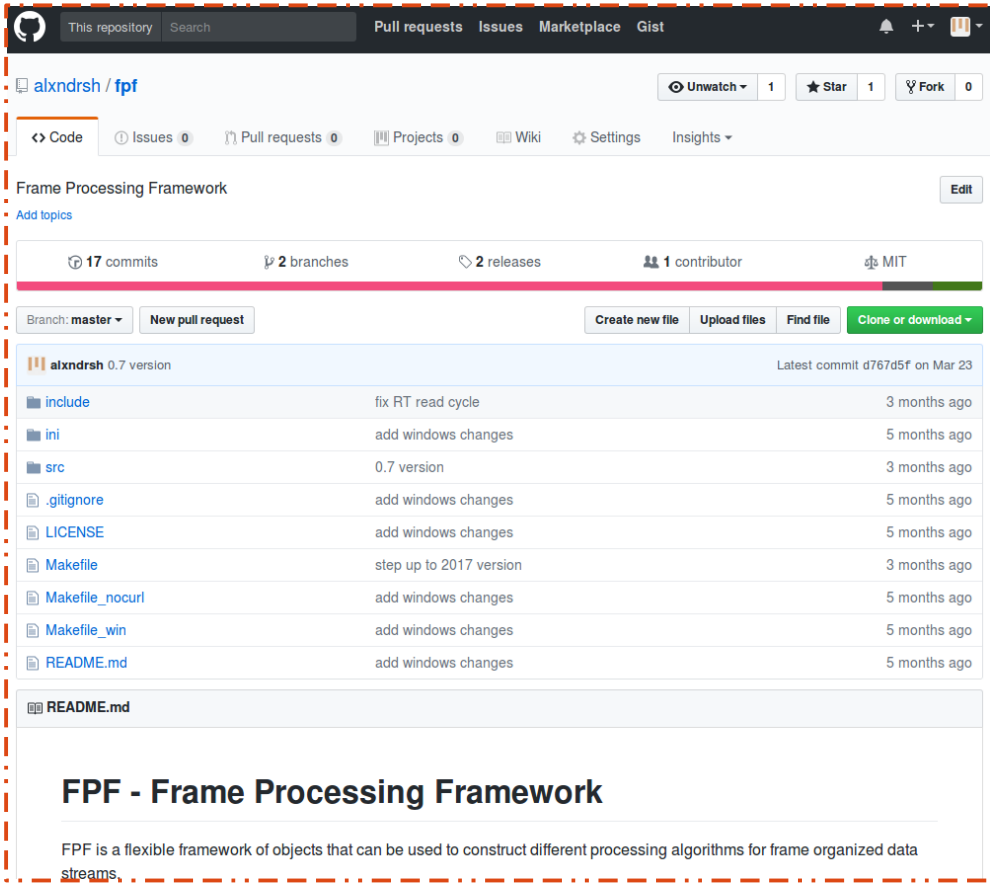


Distribution (where to take it)

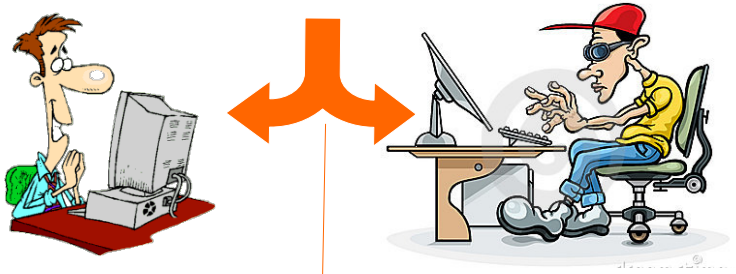
- FPF is maintained as Free Open Source project.
- License: MIT Style (no restrictions)
- Source and binary distributions are hosted at GitHub
  https://github.com/alxndrsh/fpf

- Source code is C++ 11.
- Compile environments:
  - GCC/make on Linux
  - GCC/Linux + mingw cross-compile on Linux for Windows
  - GCC/Mingw on Windows for Windows

- No build dependencies for the framework core functions and basic objects
- Optional dependencies for some objects and optional features:
  - cURL (optional, required)
  - TCP socket API



Deployment (how to get it running)



- Prebuilt binaries are provided for
1) Linux 64 bit
2) Windows 64/32 bit

- Read the manual (“Fast start”)
- Download the archived package https://github.com/alxndrsh/fpf/releases
- Unpack it in any folder.

- ~~Read the manual~~
- Grab the source
  git clone https://github.com/alxndrsh/fpf.git

- Make
  make

Name	Size
LICENSE	1.1 kB
fpf.exe	1.1 MB
README.md	20.5 kB
raw2pds_terra_aqua.ini	4.1 kB
aqua_pds_inventory.ini	1.1 kB
terra_pds_inventory.ini	1.1 kB

- Decide what do you need it for, if you need it at all
- Provide input data, select suitable INI file sample, modify or write your own
- Run the FPF engine as (e.g.)
  fpf.exe -i c:\fpf\inifiles\terra.ini TERRA_RAW_FILE.dat

- Errors? Read the manual
- Errors? Contact author (alex@rbcsignals.com)
- Do you really need it ?

Applications (let us do something useful)

As soon as we have a tool working with frame stream at a ground station.
Let us do...

- Read input stream in real-time (using socket stream or reading hot files right in process of cooking)
- process from CADU frames to sensor packets and export L0
- While passing packets conduct quality/content/continuity analysis and generate a metadata report about small data chunks.
- As soon as we have inventory information about what a station is now receiving, let us **pass this information to a common metadata inventory**. This gives possibility to collect global information what station are collecting now and what they have got in the past.

CSAIS = Common Satellite Acquisitions Inventory Service

- **Collects only metadata (~10-50kB/pass)**
- **Stream type agnostic (generic metadata – timestamp, content, location, completeness, quality indicators) , may be extended to different sensor streams**
- **Extreme Real Time (a few seconds latency)**
- **Ingest inventories from manuy stations. Global coverage**

How to connect a station to CSAIS

- In the provided INI sample for processing AQUA/TERRA MODIS (starting either from raw CADU or from PDS level) there is an Inventory node, which be default writes an inventry report into local file.
- Report may be posted to CSAIS once after completion or in Real Time after inventory of short stream slices (1-2 sec)
- Provide information about your station (do distinguish metadata coming from different stations) (replace XXX placeholders in *report_header*=….)
- Uncomment either *post_to*= or *post_to_nrt*= paramenter giving a server endpoint

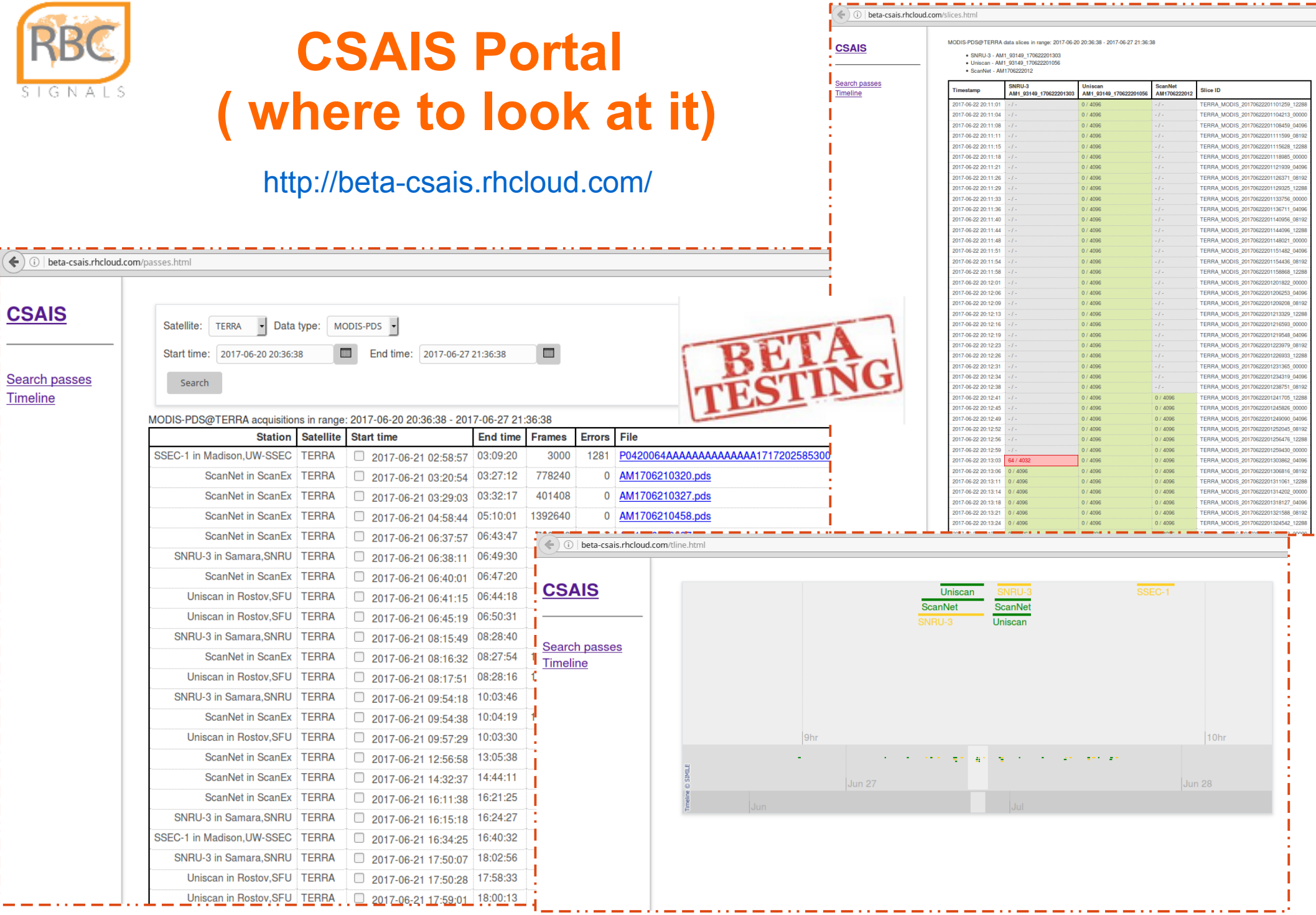
```
[AM_Inventory]
class=CNode_EOSinv
apid=64
lazy_create=yes
save_to=$INPUTFILE$.pds.inv
slice_prefix=TERRA_MODIS_
data_type=MODIS-PDS
satellite=TERRA_
#--WARNING: fill the header fields with your own identification strings
report_header=ReceiverID: XXXX, Provider: XXX, Note: saved from RAW
#-- this is URL of the folder where you hosts the PDS files
data_url=http://xxx.xx.xx/

##-- if you want to post your metadata to CSAIS service,
##-- uncomment one of the following lines:
##-a) either next one - post whole file at the end of processing
#post_to=http://beta-csais.rhcloud.com/inventory

##-b) or the next one - posts records in ERT as soon as they are collected
#post_to_nrt=http://beta-csais.rhcloud.com/inventory
```

LATUV-1320_20170602...00139_681752_raw.txt

```
#ReceiverID: Modis,EOScan Provider: Valladolid,LATUV  Note: saved from PDS
#data type: MODIS-PDS
#satellite: TERRA
#data file: AM1706011024.pds
#data url:
#ref.APID: 64
#packets per slice: 4096
#created at: 2017-06-02T10:01:38Z
#fields: slice_id,ref_time,file_pos,slice_size,num_frames,num_errors
#
TERRA_MODIS_20170601102417259_00000 2017-06-01T10:24:17.259 0   29532   46  4050
TERRA_MODIS_20170601102417259_04096 2017-06-01T10:24:17.259 29532   2629632 4096    0
TERRA_MODIS_20170601102420214_08192 2017-06-01T10:24:20.214 2659164 2629632 4096    0
TERRA_MODIS_20170601102421691_12288 2017-06-01T10:24:21.691 5288796 2629632 4096    0
TERRA_MODIS_20170601102423168_00000 2017-06-01T10:24:23.168 7918428 2629632 4096    0
TERRA_MODIS_20170601102426122_04096 2017-06-01T10:24:26.122 10548060    2629632 4096    0
TERRA_MODIS_20170601102427599_08192 2017-06-01T10:24:27.599 13177692    2629632 4096    0
TERRA_MODIS_20170601102429076_12288 2017-06-01T10:24:29.076 15807324    2629632 4096    0
TERRA_MODIS_20170601102432031_00000 2017-06-01T10:24:32.031 18436956    2629632 4096    0
TERRA_MODIS_20170601102433508_04096 2017-06-01T10:24:33.508 21066588    2629632 4096    0
TERRA_MODIS_20170601102436152_08192 2017-06-01T10:24:36.152 23696220    2629632 4096    0
TERRA_MODIS_20170601102437939_12288 2017-06-01T10:24:37.939 26325852    2629632 4096    0
TERRA_MODIS_20170601102439416_00000 2017-06-01T10:24:39.416 28955484    2629632 4096    0
TERRA_MODIS_20170601102442184_04096 2017-06-01T10:24:42.184 31585116    2629632 4096    0
TERRA_MODIS_20170601102443848_08192 2017-06-01T10:24:43.848 34214748    2629632 4096    0
TERRA_MODIS_20170601102445325_12288 2017-06-01T10:24:45.325 36844380    2629632 4096    0
TERRA_MODIS_20170601102448279_00000 2017-06-01T10:24:48.279 39474012    2629632 4096    0
TERRA_MODIS_20170601102449756_04096 2017-06-01T10:24:49.756 42103644    2629632 4096    0
TERRA_MODIS_20170601102451233_08192 2017-06-01T10:24:51.233 44733276    2629632 4096    0
```

CSAIS Portal
(where to look at it)

http://beta-csais.rhcloud.com/



and a step further….

- So metadata inventory let us know what stations are receiving at the moment
- Let us **collect the data from many stations in extreme real time** to build a joined swath

- FPF at the stations extracts and prepares packet slices (~ 1 sec.) and posts them to the server.
- Push mode. Station do not need to maintain publishing of data (only client access to Internet is required)
- Server accepts packet slices from many stations and inventories all the data instances
- Server continuously runs a worker which relying on inventory of arrived data tries to select the best copy of available packets and reconstruct a single continuous data swath.
- So data at this stage is available with **few seconds delay** after taking image
- To maintain compatibility with the standard processing tools and procedures,the swath is then cut into fixed granules and stored as standard L0 product.
- L0 PDS granules for TERRA/AQUA MODIS. 20 – 60 sec.
- As soon as a L0 granule collected and a file is available it can be processed to higher level products.
- Total delay from imaging to images – 1-2 min, independent of position in the pass.

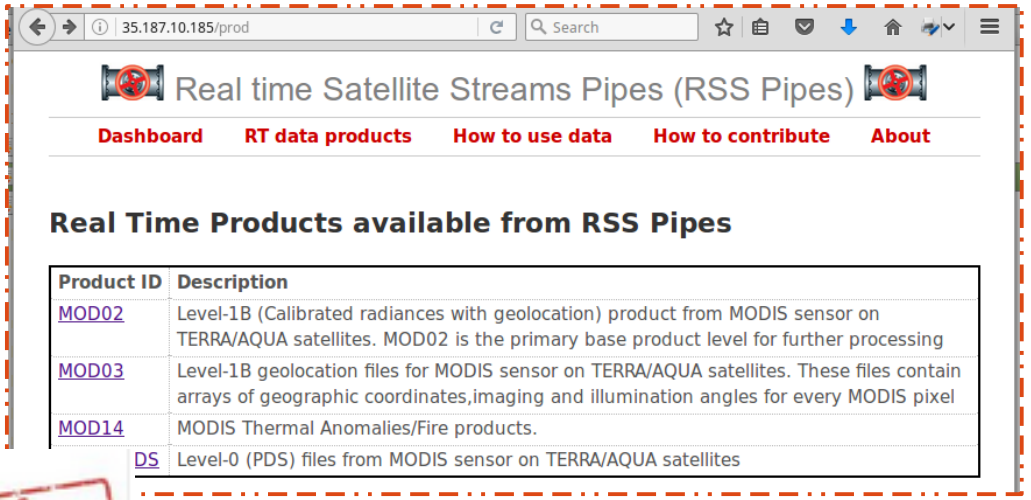
Real time Satellite Stream Pipes

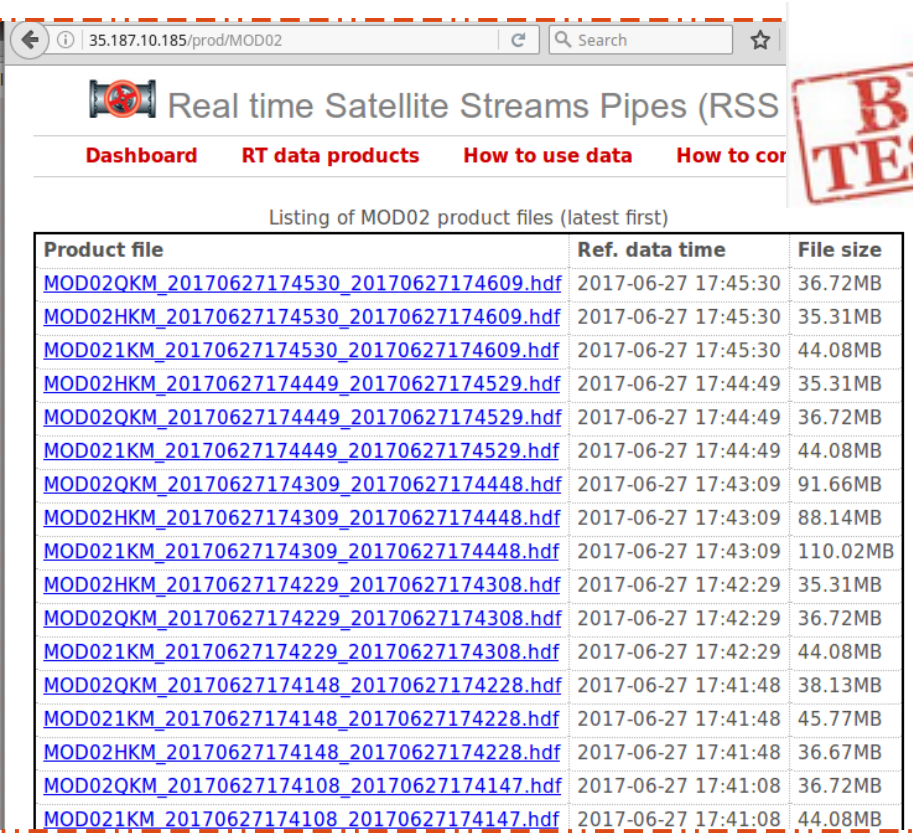
Very initial RSS Pipes system,

- data collection
- swath stitching worker
- distribution web portal

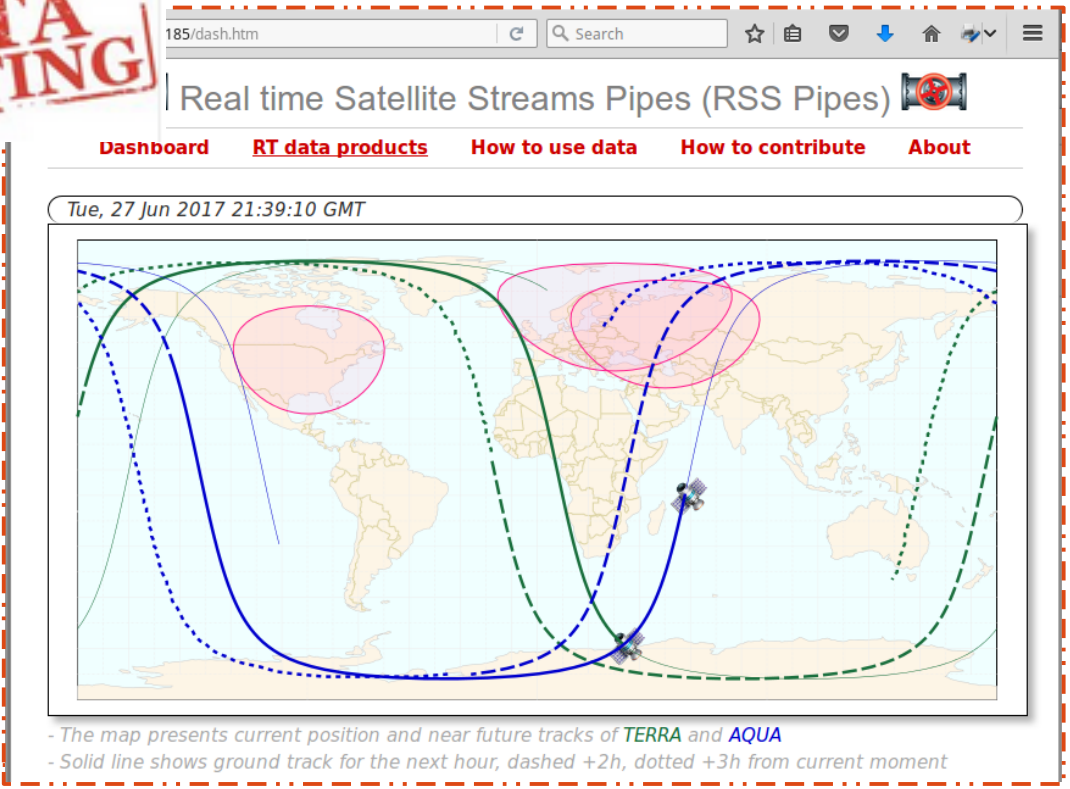
is running at

http://rsspipes.rbcsignals.com



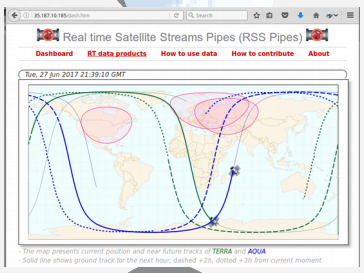


BETA TESTING



Invitation for cooperation

Ground station operators



Real Time data users

Ground stations and data users are welcome to join us in development, operation and use

Contact us: csais@rbcsignals.com
or alex@rbcsignals.com