Argus

# **Table of contents**

# **1 Introduction**

# **2 Installation**

# **3 Use**

# **4 Etc directory**

# **5 Adding a CT**

# **6 Source code**

# **1 Introduction**

Argus is a full rewrite of Applogsum. Both tools are meant to process the application event logs of Toshiba Medical System Computed Tomography systems automatically in order to provide usable information in a few seconds. For now, Argus will only be provided as a command line instrument.

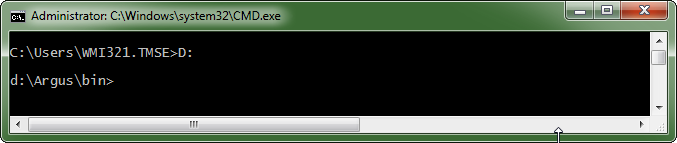
The software is written based upon the documentation contained in the training handouts and the service manuals, as well as feedback provided by a happy few and many hours of troubleshooting in the field.

# **2 Installation**

Unzip the file provided, by preference in a directory without white spaces in the name. E.g ‘D:\CT\_Tools’ and not ‘D:\CT Tools’.

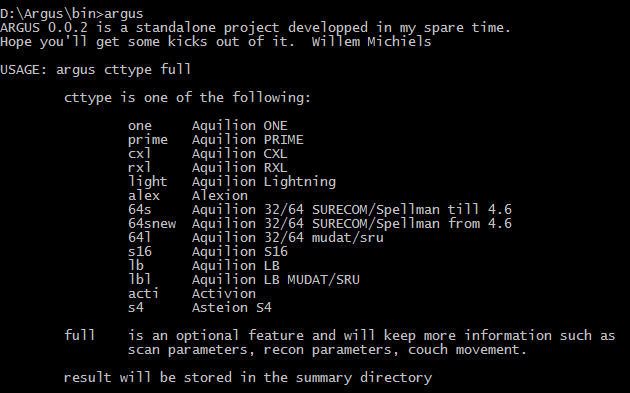
# **3 Use**

Open a windows shell and go to the directory where argus.exe is located



Store the uncompressed application event logs in the **evtlogs** directory.

In the command line run **argus** to get a small overview on how to use.

****

To process the application event logs stored in the evtlogs directory if they were provided from an Aquilion One enter **‘argus on’** or **‘argus one full’**. The difference between both is that full will generate a much bigger file due to the extra filters used to process the logs. All logs generated on the same day will be joined in a single output file. The output is located in the **summary** directory.

# **4 Configuration**

All files located in the **etc** folder influence the behavior of Argus.

Following files are related to a specific model of CT:

* + 64l.yaml
  + 64s.yaml
  + acti.yaml
  + alex.yaml
  + cxl.yaml
  + lb.yaml
  + lbl.yaml
  + light.yaml
  + one.yaml
  + prime.yaml
  + rxl.yaml
  + s4.yaml
  + s16.yaml

The content of these files can be edited with an asci editor such as notepad, scite or notepad ++.

All the data in these files comes from the service manuals where the gts logs are explained in detail.

* + ct\_type.yaml

This file contains the short names as used for the yaml files and a brief explanation about the system it refers to.

When en error is displayed on screen or in the logs, it will look like:

11:16:01.838:(A)[ErrDspMgr.] ErrDspMgr: errcode = b1e00000 device = 00010001 processNo.= 8000

**Device** is decoded by using:

* + device\_id.yaml
  + ss\_sub\_id.yaml
  + errmsg

This file is stored on all CT systems and can be overwritten by a newer one. The only condition is that the semantics of the file remain the same. Up till version 7 the construction of the file has remained unchanged. For version 8 or higher, if the error messages are no longer decoded, revert back to an older version.

When an error has information like:

11:16:01.840:(A)[ErrDspMgr.] ErrDspMgr: mode[0R], exec[MP], error[0x00000000]

If it is related to the GTS/GMS, the mode is decoded using

* + gts\_cmd.yaml

If the error is related to XC, the command and error are decoded using

* + xc\_cmd.yaml
  + xc\_errmsg.yaml
  + xc\_nack.yaml

For the decoding of the XC sequences, the following file are used:

* + xc.yaml
  + xc\_sru.yaml
  + xc2.yaml
  + xc3.yaml

SRU and first generation Spellman generators use the same type of encoding (3f3f3f….) for the XC feedback. To differentiate between both, the CT’s using an SRU are listed in

Second generation Spellman will be recognized automatically because the first digit is always 4.

Third generation Spellman uses either 3 or 4 as first digit, therefore the CT’s using this type of generator are listed in

* + ct\_with\_xc3.txt

The look and feel of the html files can be modified by editing

* + toshiba.css

# **5 Adding a CT**

To add the decoding data for a specific type of CT you need the following:

* the service manual
* type of generator

Copy one.yaml located in the etc directory and rename it. As example we will use **foo**.

Edit the contents of **foo.yaml** according the data found in the service manual.

If the generator is an sru, create a line in ct\_with\_sru.txt and put **foo** on that line.

If the generator is an XC3 type add **foo** to ct\_with\_xc3.txt.

Edit ct\_type.yaml and add **foo** followed by a proper description of the CT type.

# **6 Source code**

For those who are interested I’ll try to explain the code of Argus. Argus is written in Ruby (<http://www.ruby-lang.org/>) and compiled to an executable with ocra (<https://github.com/larsch/ocra>) . For more information about both, please consult the internet.

The way argus processes data is rather easy:

* initialize the program by using the data stored in the etc folder
* scan the evtlogs directory for all files with extension Evt or Evtx
* group the log files by data
* extract all readable lines from a log file and look at them in sequence
* if a sequence fits a given pattern, add information
* write the obtained information to an html file in the summary directory

The source file has a single **class** inside **Argus**.

The methods of the Argus class are:

* initialize
* ct\_type
* find\_eventlogs\_by\_date
* eventlogs\_to\_txt
* error\_meaning\_and\_dev\_id
* decode\_ddmotor
* decode\_gts\_mode\_exec\_error
* decode\_interlock
* decode\_free\_light\_out2
* decode\_censor
* decode\_gts\_detail
* decode\_lmudat
* decode\_xc\_mode
* decode\_xc
* convert\_base
* hex\_to\_bin

All decode methods are related to a specific item in the details provided by the GTS/GMS.

In order to know which data is seen as useful, check the eventlogs\_to\_txt method. All data is matched with a pattern (regular expression) or a fixed value. Some of the lines will only be used if the **full** switch is used to run argus.

The class variables are:

* eventlogs
* errmsgs
* readable\_info
* device\_id
* ss\_sub\_id
* xc\_cmd
* xc\_errmsg
* xc\_nack
* details
* ct\_type
* ct\_with\_xc3
* ct\_with\_sru

All these variables have fixed contents related to the etc directory, except for ct\_type and eventlogs which are definded by the command line options and the contents of the evtlogs directory.