MASS Software Reference Manual

V.Kornilov, S.Potanin, N.Shatsky, O.Voziakova, A.Zaitsev

Generated by Doxygen 1.2.14

February 28, 2002

Introduction

This document represents the description of the MASS controlling program - TURBINA from the point of view of programming and implemented algorithms. The TURBINA usage is given in a separate document "MASS Software User Guide" where the principles of the performing of measurements with MASS are described. The general presentation of the MASS project is given in "MASS Final Design Document" where one can find the scientific objectives and the MASS design presentation. The MASS maintenance separated into another document "MASS Engineer Guide".

Current Reference consists of two parts which are made almost independent from each other: Part I "Instrument Control" contains the description of the TURBINA program architecture, its basics, modules hierarchy and the modules related to the MASS device control. The "scientific" modules are described in the Part II "Data Processing" where the exact description of the used formulae, algorithms etc is given for calculation of the scintillation indices and atmospheric parameters and turbulence profile.

Part I. Instrument Control

Contents

1 MASS Software. Part I: Instrument Control 1

2 Part I. Namespace Index 7

3 Part I. Hierarchical Index 7

4 Part I. File Index 9

5 Part I. Namespace Documentation 11

6 Part I. File Documentation 24

1 MASS Software. Part I: Instrument Control

1.1 Introduction

The TURBINA program represents the GUI-interface for managing the measurements of stellar scintillation indices and atmospheric integral parameters and turbulence profiles. It is written in C++ language and is based on the Qt libraries which provide the means for creation of GUI components. In this part of the document, the principles of the program organization, its modules and their mutual relations and dependencies are described. The "scientific" modules which deal with scintillation and atmospheric calculations are described in a second part of MASS Software Reference "Data Processing".

TURBINA is a complex program executed in X-Windows environment which provides the interface between the User and the device or, more precisely, its driver inserted in the Linux kernel on the stage of the PC startup. Its actions and events are naturally asynchronous; the calculations are done in real-time, in parallel with interaction with user and device and I/O operations with disk. Apart from this, the CPU time cannot be fully consumed by TURBINA and must be left for other running system and user's applications. So, the program is organized in a complex way which demands an introduction of some basic notions. They are not specific to TURBINA only and used in most of similar applications, but it is useful to describe them in brief here.

© 1997-2002

1.2 Basic notions

In this section, we give a short explanation for "thread" and "mutex" notions on which the TURBINA operation is based.

- *Thread* is a separate sequence of CPU instructions which serve for one particular task of application and which can work in parallel with other threads. Technically, the thread owns its stack and CPU registers sets which are loaded in CPU when it is under execution. Meanwhile, all the threads started from one program (i.e. *process)* operate on the same memory segment which demands some means for synchronization of the data access between them (see below).

From the point of view of TURBINA software, the *threads* which are started explicitly are realized as infinite cycles. These are the

- the "Scenario" thread which implements the execution of modes or their sequences (*scenario)*;
- the "Data" thread which waits for data blocks from the driver FIFO and sorts the received data into four buffers associated with the MASS channels (A, B, C and D).

Two other threads which are started implicitly on the program startup are the "Main" GUI-thread which serves for the interaction with a user, and another ("internal") thread which serves the GUI system.

Since the threads within one program - i.e. TURBINA - share the same memory segment and need to access the same data structures for exchanging the information between them, some *synchronization* of this access from parallel running threads is needed. It is made by means of *Mutex*.

- *Mutex* is a structure in memory (Qt represents the class "QMutex") which serves for threads manipulation:
  - for their synchronization and for preventing the conflicts while occasionally simultaneously accessing the same data structures under modification.
  - for stopping/resuming the working of threads according to different circumstances occurring in the program execution.

  To implement these functions, the mutex has the ("atomic", i.e. non-interruptible) functions **lock()** - to stop execution of (some) thread, and **unlock()** - to resume execution of a thread. In TURBINA, the thread stops itself with the help of the mutex lock() and is resumed

© 1997-2002

from *another* thread by unlock()-ing of the same mutex. The following rules describe the behavior of the program (one of its threads) when its control reaches the mutex lock/unlock instruction:

- lock() on the mutex (variable) which is unlocked results in locking this mutex and the control goes to the next instruction after lock(), i.e. the thread continues to work;
- lock() on the mutex which is *already* locked() stops the current thread execution until some other thread unlocks this mutex. Then the previous rule takes place (locking and continuation).
- unlock() on mutex in any state unlocks it and the control goes further.

In what follows, we will explain the place of these notions in TURBINA organization.

1.3 Justification of the threads use and their interaction

The key notions presented above help to understand the overall scheme of working of TURBINA (see Figure 1 below) in execution of its basic functions - reaction to the user's actions and digesting the input data from the device.

After the program startup (optionally, Initial Scenario is performed), the control is reserved completely for Main thread. It waits for actions from user (clicking on buttons or menu items etc.) in reply to which it

- sends some commands directly to the device (i.e. driver). As example - "Open Door", "Turn HV On", etc., and waits until these commands are completed.

- starts the different modes or their sequences (scenarii) setting the mode switch to the needed mode or to the first mode of scenario. This switch (squares with slash in the figure) is passed by the "Scenario" thread control when the "Main" thread unlocks the respective mutex in it. Then the "Scenario" thread selects the mode in a switch and calls one of the respective mode procedures (implemented in **device** module). This algorithm (see next Figure 2) initializes itself, sends some commands to the device to start the accumulation of data and waits until these data are ready in the mutex lock point (black box in each mode algorithm rectangle).

© 1997-2002

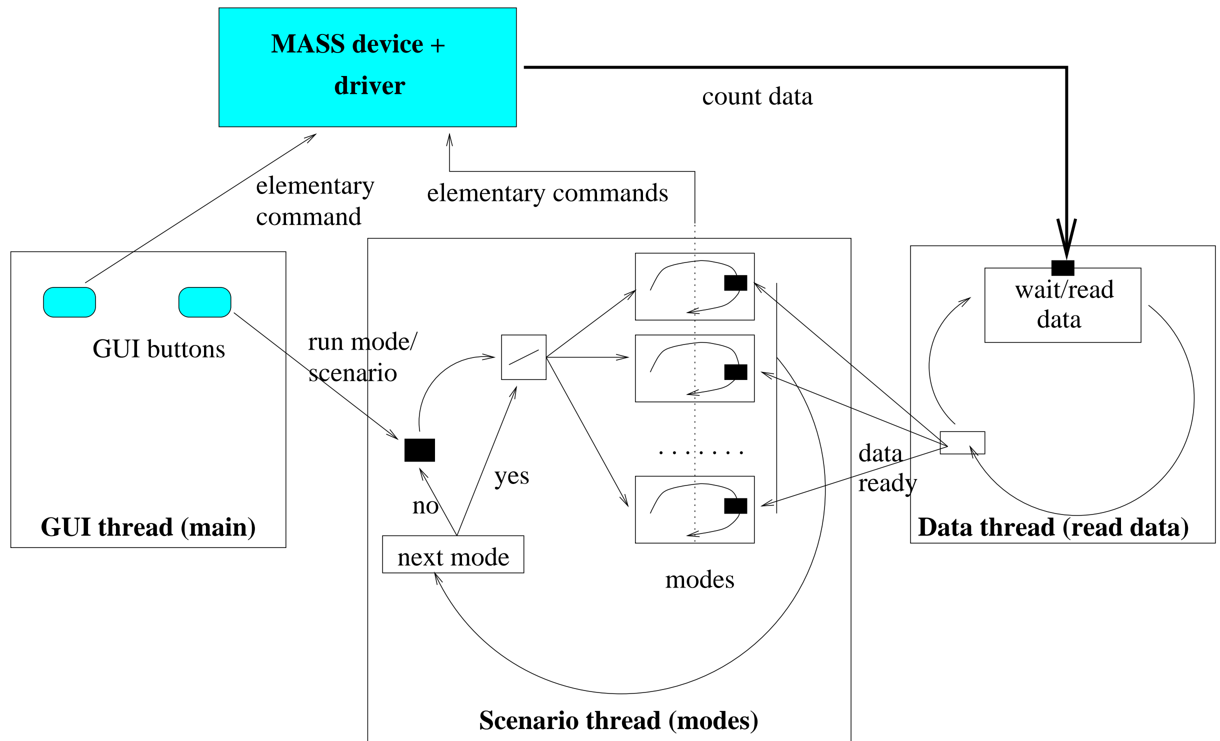

Figure 1: Interaction of the three TURBINA threads shown as three large marked rectangles. Black boxes represent mutexes which are unlocked by incoming arrows from other threads and locked by internal cycles of threads.

© 1997-2002

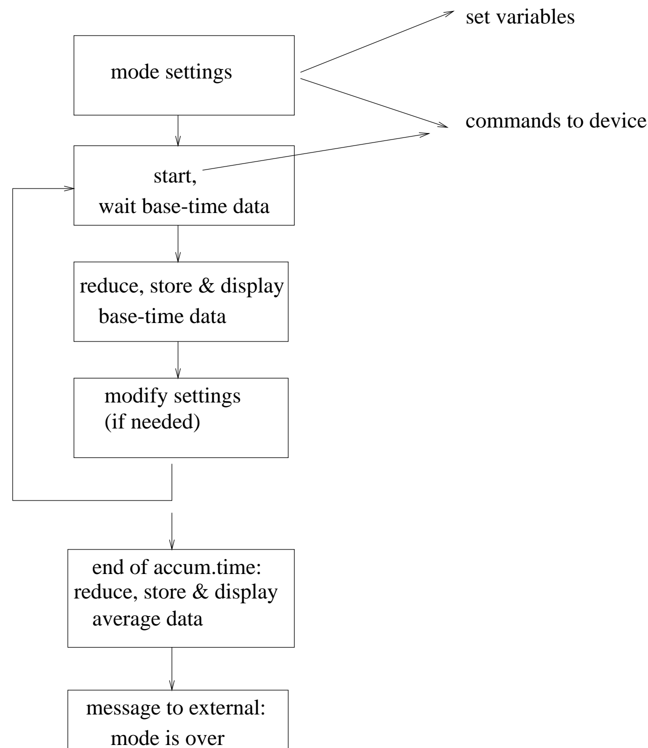

Figure 2: Block-scheme of a mode algorithm. The cycle is over the base-time and the loops are repeated approximately (accumulation time)/(base time) times. So, the scheme describes one full accumulation time actions.

When the acquired data arrive in the driver FIFO from the device modules (thick arrow from the device box to "Data" thread on the Fig.1), they are sorted by MASS channel buffers in "Data" thread and, when ready, this thread unlocks the mutex which has stopped the mode in "Scenario" thread. The data are reduced (see Fig.2), displayed, stored etc. and thus one **accumulation** time cycle is over.

Then the control goes to the point where the "next mode" is selected and set in a mode switch *if* the scenario was started. If it was a single mode call from the "Main" thread, the control stops at the mutex lock point until the next call from the "Main" thread.

1.4 The TURBINA modules and their dependencies

From the point of view of programming, the program consists of a number of modules (compiled independently and linked later in one executable file).

The scheme of their mutual relations is presented in Fig.3 where arrows show the "inclusion way": an arrow going from module 1 to module 2 means that module 1 is used by (or *#include-d* in) the module 2.

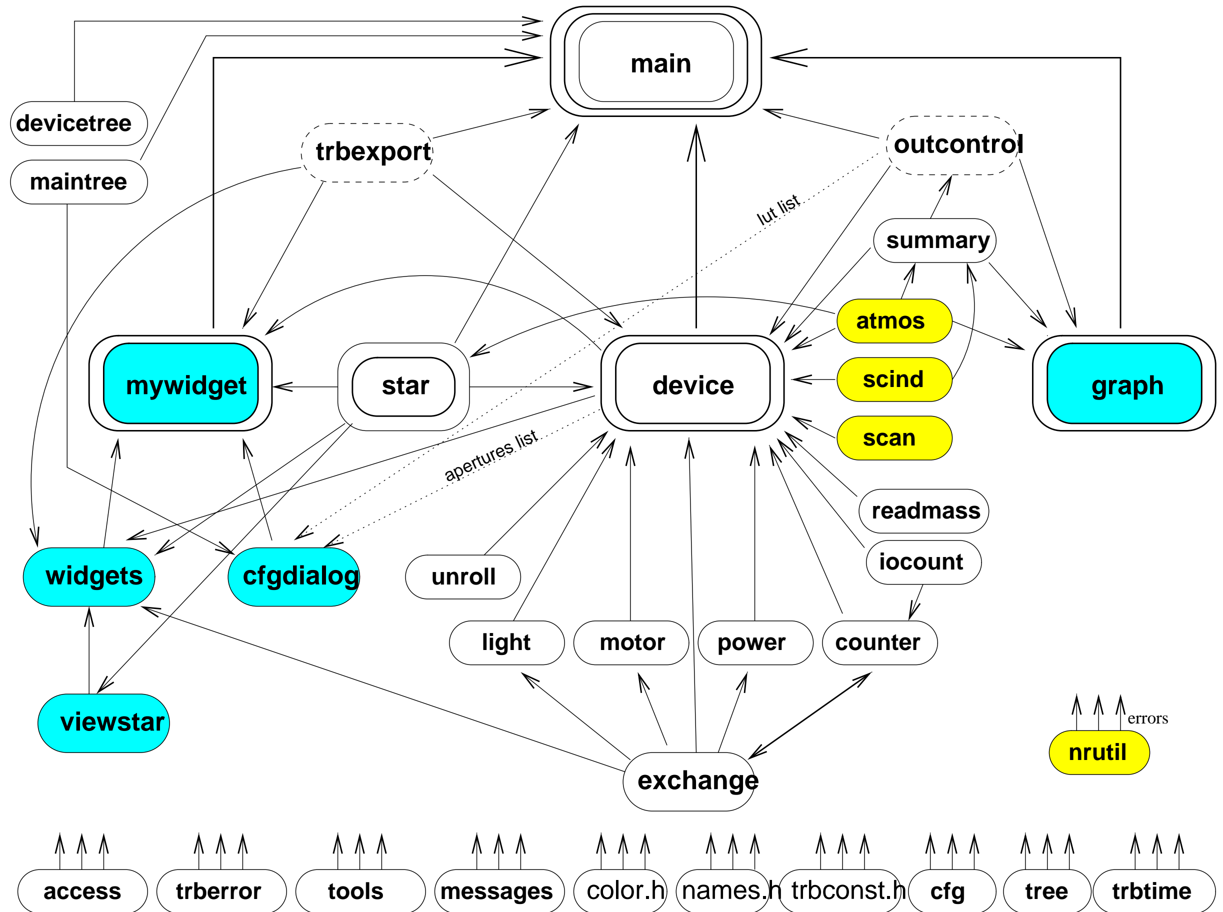

Figure 3: The tree of module dependencies in TURBINA. Light grey/yellow boxes - scientific modules, grey/cyan boxes - GUI modules.

References

1. UNIX System Programming using C++, T. Chan, Prentice Hall PTR, NJ, 1997

2. Linux device drivers. A. Rubini, O'Reilly, 1998

3. The C++ Programming Language. B. Stroustrup. Third Edition. Addison-Wesley, 2001

4. Programming with Qt. M.K.Dalheimer. O'Reilly, 1999 [obsolete edition

© 1997-2002

used for principles study only]

5. HTML reference manual pages for Qt programming at {\tt `http://doc.trolltech.com`}

2 Part I. Namespace Index

2.1 Part I. Namespace List

Here is a list of all documented namespaces with brief descriptions:

ioc (Declarations of the channel count type and counts I/O operations) **11**

rdm (Reading and parsing the mass-file lines) **13**

sum (Handling the scientific measurement results for the night-time summary and graphics) **18**

3 Part I. Hierarchical Index

3.1 Part I. Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AdjustDialog
BackCorrectionDialog
BlockCounter
BranchDialog
CenterCorrectionDialog
CenteringDialog
CFGBranch
CFGDialog
CFGLeaf
CFGTree
CommentDialog
Counter
CountThread
DataThread
DegreeValidator
DeviceState
DeviceTree
DevManager

© 1997-2002

- DigitalClock
- DisplayData
- ErrorBase
  - CFGError
  - DeviceError
  - LoadError
- ExportTable
- ExportText
- FloatValidator
- ForSlot
- GraphData
- GraphSet
- GraphWindow
- HelpView
- HVDialog
- IllumDialog
- InfoExport
- IntValidator
- JKQComboBox
- JKQLineEdit
- JKQListViewItem
- Light
- LightDialog
- Link
- MainTree
- Motor
- MyWidget
- OutControl
- sum::param_t
- PointDouble
- Power
- Progress
- QJKMainView
- QJKPushButton
- QJKTable
- Scale
- SelectStarDialog
- ShowSet
- ShowStarDialog
- Star
- StarInfo
- StateExport
- SubItem
- TalkDialog

© 1997-2002

Time
TimeValidator
ViewStarList

4 Part I. File Index

4.1 Part I. File List

Here is a list of all documented files with brief descriptions:

access.h (Deals with access rights to files, fields of CFG etc., determines the current user status: "mass-root" (expert) or not (simple user)) 24

cfg.h (Tools for accessing to CFG tree in memory: search, validation, I/O) 25

cfgdialog.h (Module for implemention of the GUI-dialogs of a user with a tree of CFG parameters) 25

cnt2asc.cpp (Reading the MASS binary count-file *.cnt and converting the count data into an ASCII table. A standalone program) 26

color.h (The list of the color names (character constants) reserved in Qt (17 colors)) 28

counter.h (Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with photometric modules) 28

decldevicetree.h (The "body" of tree from devicetree.h module: the list of CFG parameters itself) 28

declmaintree.h (The "body" of tree from maintree.h module: the list of CFG parameters itself) 29

device.h (High-level interaction with the device bearing the manager functions which implement the algorithms of modes) 29

devicetree.h (Keeps (statically) in memory the tree (parameter values) of the CFG stored in `device.cfg`) 30

© 1997-2002

exchange.h (Low-level exchange implementation of the program with the device driver) 30

graph.h (Supports the output of the measurement results in the graphic window) 31

iocount.h (Input/Output of the individual photon counts in a binary "count-file") 32

light.h (Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with buttons and control/illumination LEDs module) 34

main.cpp (Initiates the program execution) 34

maintree.h (Keeps (statically) in memory the tree (parameter values) of the CFG stored in `turbina.cfg`) 36

messages.h (GUI-output of the software messages to the screen) 36

motor.h (Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with stepper motor of the aperture wheel) 36

mywidget.h (GUI-implementation of the program main window: menu structure, status bars etc. linking together all the components provided by widgets.h) 37

names.h (Reserved string constants: file names, paths, extentions and the numeric formats for data display on the screen) 38

outcontrol.h (Manipulation of the output in the main results and graphic results windows) 38

power.h (Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with high voltage supply module) 39

readmass.h (Reading the content of the mass-file for Playback function and CFG parameters last-written values) 39

star.h (Reads the star list and interprets the star parameters when needed) 41

© 1997-2002

summary.h (Collection of scintillation indices and atmospheric parameters results from scientific modules for their graphic representation and the summary-file handling) **42**

tools.h (A variety of functions and tools needed for all other modules) **44**

trbconst.h (Commonly used constants (e.g. MM2CM, SEC_IN_DAY etc.)) **45**

trberror.h (The declarations of C++ exception classes which may be invoked from some points in star.h, device.h etc) **45**

trbexport.h (The class is presented with static fields which serve for exchange of data between the Scenario thread and Main-thread of the program) **46**

trbtime.h (Support of date/time-related tasks. Implements namespace trb_tm and site stellar time, coordinates, UT etc) **46**

tree.h (Class implementing the CFG tree organization: principles of its building, reading, searching info etc) **47**

unroll.h (Converts the scenario formula in the sequence of mode symbols) **48**

viewstar.h (Shows the list of stars with help of GUI and lets user to choose the star from this list) **49**

widgets.h (Implementation of the different components the main window of TURBINA collected together by mywidget.h) **50**

5 Part I. Namespace Documentation

5.1 ioc Namespace Reference

Declarations of the channel count type and counts I/O operations.

Functions

- long **write** (FILE *fcnt, **count_t** **buffer, int buflen, int nbuf)

© 1997-2002

Write the channel count series to disk.

- void **read** (FILE *fcnt, **count_t** **buffer, int *buflen, int *nbuf, long addr)

  Read the channel count series from disk.

5.1.1 Function Documentation

5.1.1.1 long ioc::write (FILE * *fcnt*, count_t ** *buffer*, int *buflen*, int *nbuf*)

Parameters:

fcnt structure of the opened binary file of counts

buffer array of *nbuf* pointers to channel count buffers

buflen length of the channel buffer in units of counts (not bytes!)

nbuf Number of buffers (e.g. number of active channels)

Returns:

Starting position in file (in bytes from beginning) where the counts were written or (-1) on error

The function uses the *fseek()* to set the file position to the end of file (where it is, normally, set already), gets this current position to be returned on success, and writes sequentially all *nbuf* channel buffers to disk. Before writing of each *i-th* channel buffer (*i=0..nbuf-1)*, the descriptor is written consisting of two numbers: {*nbuf-i*, *buflen* }.

After writing, the file pointer is retained at the next position after the last written data number.

5.1.1.2 void ioc::read (FILE * *fcnt*, count_t ** *buffer*, int * *buflen*, int * *nbuf*, long *addr*)

Parameters:

fcnt structure of the opened binary file of counts

buffer array of *nbuf* pointers to channel count buffers

buflen On enter: Maximal length of the channel buffer in units of counts (not bytes!) On exit: Actual length of channel buffers

nbuf On enter: Expected number of buffers (e.g. number of active channels) On exit: Encountered number of buffers (error if differs from *nbuf* on enter)

addr Starting position in file (in bytes from beginning) where the counts should be read from.

The address is checked to be only the multiple of the size of the **count_t** type.

The function uses the *fseek()* to set the file position to the given *addr*, and reads sequentially *nbuf* records *each* consisting of a two-number descriptor plus a count buffer read into the respective *buffer[i]* *(i=0..nbuf-1)*.

Each of descriptors is checked to be consistent with the parameters *nbuf* and *buflen*:

- the first number is checked to be equal to *nbuf-i*
- the second number must not exceed the parameter *buflen* value
- the second number must be equal for all *nbuf* records

If one of these conditions are not justified, the error with the code nr::EROFL is returned and the further reading stops. The error nr::ERFIO is returned when not all the counts are successfully read or *addr* is not fseek()-able.

The content of a first descriptor is returned in place-holders *nbuf* and *buflen* in any case, either on error or on success.

After reading, the file pointer is retained at the next position after the last data number read.

5.2 rdm Namespace Reference

Reading and parsing the mass-file lines.

Functions

- char **readline** (FILE *mf)

  Read the next line from the mass-file stream mf.

- char **gettype** ()

  Return the last line prefix.

- bool **isheader** ()

  Disentangle the comment- and header-type lines.

- long **getstartpos** ()

  Return starting position of last-read record.

- double **getut** (int *UTh, int *UTm=0, int *UTs=0)

  Return the parsed UT moment as hours, minutes and seconds.

- string **getut** ()

  Return the string-type UT moment (unparsed).

- string **getname** ()

  Return the parameter name or started mode name or star catalogue ID.

- string **getvalue** (const string pname="")

  Return the value of parameter or star parameters or line content.

- long **getaddr** ()

  Return the count-file address for the last read base-time results line ('i','j'-type).

- int **add2map** (const string pname="", const string pvalue="")

  Add the parameter and its value to the map filled with **grabparams**() (p. 17).

- int **grabparams** (FILE *mf)

  Read the whole mass-file to collect the parameter values from the preamble-type lines.

Variables

- const long **NOADDR** = -1

5.2.1 Function Documentation

5.2.1.1 char rdm::readline (FILE * *mf*)

Parameters:

mf the stream pointer of an opened mass-file

Returns:

line prefix or \0 for failed reading/parsing or if EOF

© 1997-2002

Reading the line and parsing it if possible: for line types **'P','M','i','j','O'** . No action is performed if null-file *mf* is supplied (segm.fault-protected).

Before parsing the line, the carriage-return character is removed from the line.

The first character of line determines the line *type* and thus the returned value (see also **gettype**() (p. 15)).

For any line but that of header-type the UT is extracted from the second field and can be obtained with **getut**() (p. 16).

The *name* (returned by **getname**() (p. 16)) is extracted for 'P'-, 'M'-lines and 'O'-lines starting *after* the *last* space character after UT record:

- 'P' : name is ended *before* the (first) '=' character (*after* this '=' the *value* of the parameter starts) and turned into **upper-case**;
- 'M' : name of mode is ended at the end of line and turned into **upper-case**;
- 'O' : name (HR-number) lasts till the end of line. Note that the length of the HR-record depends on the numeric value of HR (e.g. 3 chars for HR=999).

The *value* (returned by **getvalue**() (p. 16)) is extracted for 'P'-lines (after the first '='-character) and for non-parsed and comment-lines.

The *address* (see **getaddr**() (p. 14)) is converted from "@..." substrings of the mass-file line or set to **rdm::NOADDR** (p. 18) if not found.

5.2.1.2 char rdm::gettype ()

Returns:
prefix of the last successively read/parsed line or \0 if no lines were read

5.2.1.3 bool rdm::isheader ()

Returns:
Header-line flag: **true** for lines started as '#*' and **false** for any other line type (comment or non-comment)

5.2.1.4 long rdm::getstartpos ()

Returns:
file position before reading last line

© 1997-2002

File position is saved before each **readline**() (p. 14); thus, with help of this function, one line may be read twice:

```
rdm::readline(f) ; // some line read
fseek(f, rdm::getstartpos(), SEEK_SET);
rdm::readline(f) ; // the same line again
```

;

5.2.1.5 double rdm::getut (int * *UTh*, int * *UTm* = 0, int * *UTs* = 0)

Parameters:

UTh Resulted UT hour (may be set zero if no UTh needed)

UTm Resulted UT minute (may be left zero)

UTs Resulted UT second (may be left zero)

Returns:

Fractional UT hour or -1 on error of UT parsing

5.2.1.6 string rdm::getut ()

Returns:

UT-string

5.2.1.7 string rdm::getname ()

Returns:

Name of parameter as Section[/SubSection[/SubSubSection]] or star's catalogue ID (first field after UT in 'O'-lines, normally HR) or mode name

5.2.1.8 string rdm::getvalue (const string *pname* = "")

Parameters:

pname parameter name in **upper-case** for accessing the map of parameter name/values created by **grabparams**() (p. 17). Empty by default for getting the "value" last read with **readline**() (p. 14).

© 1997-2002

Returns:

The rest of line to the right of the first '=' for 'P'-lines or acquired parameter name *pname*; of the catalogue ID for 'O'-lines or the full line for header-type lines or the line content to the right of UT for other non-parsed lines

This function is of the dual use. First, it returns the *value* for a mass-file line which was just read with **readline**() (p. 14) (see **Returns** description). Second, it returns the value associated with the provided parameter name *pname* taken from the parameters map created by **grabparams**() (p. 17).

5.2.1.9 int rdm::add2map (const string *pname* = "", const string *pvalue* = "")

Parameters:

pname parameter name in **upper-case** for adding in the map of parameter name/values created by **grabparams**() (p. 17). Empty by default for simply getting the size of the map.

pvalue parameter *pname* value. May be empty if, say, no value is associated with *pname*.

Returns:

The size of the parameters map after adding the parameter *pname*.

This function simply assigns the value *pvalue* (if not empty/defaulted) to the map *key pname* (i.e., creates a new entry *pname* if not existent before and associates it with the (new) value *pvalue)*. Returned is the size() of the parameters map.

5.2.1.10 int rdm::grabparams (FILE * *mf*)

Parameters:

mf the stream pointer of an opened mass-file

Returns:

Number of preamble-type lines encountered (**not** of parameters stored in map - it is less if the same parameter was encountered, say, twice).

The utility first sets the file position at its origin and then collects all the preamble-type line contents in an associative array (map). The values of parameters, if encountered repetitively are thus overwritten and upon the end of reading represent the last-written values.

© 1997-2002

After the function finishes its job, the file pointer is left at the end of file. The parameter values may be acquired with **rdm::getvalue**() (p. 16) with an exact parameter name in upper-case as an argument.

If null *mf* is supplied, the accumulated map of parameter-value pairs is only emptied.

Note:

grabparams() (p. 17) uses the function **readline**() (p. 14) for reading the mass-file. So, the content of the variables returned by **gettype**() (p. 15), **getvalue**() (p. 16) etc. is modified compared to pre-**grabparams**() (p. 17) state.

5.2.2 Variable Documentation

5.2.2.1 const long rdm::NOADDR = -1

Invalid count-file address, returned when not found or failed to convert

5.3 sum Namespace Reference

Handling the scientific measurement results for the night-time summary and graphics.

Compounds

- struct **sum::param_t**

Enumerations

- enum **partype**
- enum **reftype** { **FIRSTMEAN**, **LASTMEAN**, **SEASONMEAN** }

Functions

- const char * **partype_name** (**partype** t)

  Get "standard" name for a parameter type.

- int **createset** (int nchan)

  Create the set of series with properly assigned parameters.

- void **init** (const char *sumfile, long date, int nchan, const char *dumpfile=0)

  Read the summary file, make references and initialize the series storages.

- void **add** (double UTh, double mag, double airmass, bool isgen)

  Add the results of the last accumulation time in the series tails.

- double **getref** (const string &pname, **sum::reftype** rtype)

  Get the reference value for a given parameter.

- const double * **getdata** (const string &pname, int nres=-1)

  Get the pointer to the series storage for a given parameter.

- int **getn** ()

  Get the number of results accumulated in series storages.

- void **done** ()

  Median average the result series and update/create the summary-file and the dump-file.

- double **massmag** (double vmag, double bv, double coleq=**sum::MASSCE**)

  Convert the Johnson V-magnitude into MASS magnitude.

Variables

- const double **NODATA** = 0
- const double **MASSCE** = 0.5905

5.3.1 Enumeration Type Documentation

5.3.1.1 enum sum::partype

Types of data stored continuously during the measurements; not saved in a summary-file are the last items starting from NCN2

5.3.1.2 enum sum::reftype

Types of the reference information available:

Enumeration values:

FIRSTMEAN The value of the parameter taken from the *first* record in the summary-file (e.g. manually written site averages)

LASTMEAN The value of the parameter taken from the *last* record in the summary-file

SEASONMEAN The *mean* value of the parameter through all the lines in the summary-file (zero-date lines, e.g. with a word instead of date, are averaged)

5.3.2 Function Documentation

5.3.2.1 int sum::createset (int *nchan*) `[inline]`

Parameters:

nchan number of MASS channels

Returns:

0 on success, nr::erget() on error (nr::nrerror() is set)

5.3.2.2 void sum::init (const char * *sumfile*, long *date*, int *nchan*, const char * *dumpfile* = 0)

Parameters:

sumfile MASS summary file name

date Current date represented as DDMMYY

nchan MASS channels number

dumpfile binary file name to restore the series content from

The *sumfile* is read (if exists) with checking the compatibility of its header line with the data set expected. If not readable, the message is returned with error code nr::ERFIO. If the header is not compatible, the file content is rejected and no reference is made for parameters; error message is set with code nr::ERNOD.

If file is Ok, all the result lines are read and the mean value of all columns is computed as a *Season mean* values. The first and last line contents are saved separately. These kinds of the reference data are available with **getref**() (p. 21). First results line does not participate in averaging if it contains the invalid (zero) date. Thus, the manual provision of a first line for the **sum::FIRSTMEAN** (p. 20) reference type (e.g. the current or comparison

© 1997-2002

site averages) must be done with such a zero date to avoid biasing of a real season statistics.

The necessary space is allocated for the result series to be accumulated with **add**() (p. 21). If a *dumpfile* is supplied, the attempt to read this binary file is made. If successful and the date specified in the file descriptor coincides with the *date*, the series storages are filled from the file and will thus contain the data from the previous runs of TURBINA in the same night, to be graphically displayed using **getdata**() (p. 22). If not, the series are empty on startup; their starting length (returned by **getn**() (p. 23)) is zero.

5.3.2.3 void sum::add (double *UTh*, double *mag*, double *airmass*, bool *isgen*)

Parameters:

UTh Fractional UT hour

mag MASS-band magnitude of a target star (for the flux conversion, see **massmag**() (p. 24))

airmass airmass of a target star (for summary writing only)

isgen Generalized mode flag (availability of the shifted pupil results)

The function adds all relevant parameter values to the tails of the result series using sc::getavgflux(), sc::getavgidx(), atm::getval(), atm::getzcn2() and atm::getcn2(). Non-computed parameters are saved as **sum::NODATA** (p. 24) (e.g. shifted indices in the Normal mode).

The magnitude *mag* is used to recompute the stellar flux for the zero- magnitude star. This magnitude is assumed to be converted into the MASS instrumental magnitude using the stellar B-V color index and the color equation of the MASS photometric system. It can be done with **massmag**() (p. 24). The aim is to trace the atmospheric transparency of the site (having also the median *airmass* in the same summary-file) in parallel with the turbulence study.

The shifted-pupil stellar fluxes are not accumulated and not saved in a summary file.

5.3.2.4 double sum::getref (const string & *pname*, sum::reftype *rtype*)

Parameters:

pname The parameter name

rtype Type of a reference: **sum::FIRSTMEAN** (p. 20), **sum::LASTMEAN** (p. 20), **sum::SEASONMEAN** (p. 20)

© 1997-2002

Returns:

Reference value, **or sum::NODATA** (p. 24) if no reference is available

The name *pname*, if supplied as is for the first time, is parsed to get know which of scintillation or atmopsheric parameters it refers to. If successfully parsed, the *reference* value of a specified type *rtype* is returned and the result of parsing (address of a reference storage) is stored in an associative array.

The reserved names are

- `F_A..F_D` - fluxes in channels A..D converted to zero magnitude (`FLUX_*` are also understood)
- `SI_A..SI_D,SI_AB..SI_CD` - scintillation indices in channels and combinations (differential indices). Alternative names like `DSI_AB` may also be used
- `DESI_A..DESI_D` - differential exposure indices
- `SI_AS,SI_BS,(D)SI_ABS,DESI_AS` - the selected indices set obtained with a pupil shift in a Generalized mode
- `SEE,FSEE,HEFF,FHEFF,ISOPL,M0,M2,TAU` - integral atmospheric parameters (equivalent of atm::ordint enumeration). The names are all in upper case and **case-sensitive**.

The reference value is not available for the airmass, UT, and for the Cn2 profile data (Cn2 is not stored in a summary-file).

On error (non-identified parameter), the error nr::ERNOD is set and the value **sum::NODATA** (p. 24) is returned.

5.3.2.5 const double * sum::getdata (const string & *pname*, int *nres* = -1)

Parameters:

pname The parameter name

nres The result index for *pname*== `ZCN2` and `CN2`; latest result is returned by default

Returns:

pointer to the result series or to *nres-th (z)cn2[]* array for *pname*== `ZCN2` and `CN2`

The name of the parameter is parsed as described for **getref**() (p. 21) and, if successfully done, the pointer to respective series storage is returned. In addition to the list of parameter names in **getref**() (p. 21), the following names are allowed:

- `UT` - the fractional UT for the graphics abscissa values
- `NCN2` - the number of restored turbulence profile layers
- `ZCN2` - the turbulence layer altitudes for the result #*nres*.
- `CN2` - the turbulence layer strengths for the result #*nres*.

Note, that unlike other parameters, for pname== \c[Z]CN2 the layer altitudes and profile strengths arrays are returned, for the single (last by default) result, NOT the array of all the results obtained during the night.

On error (non-identified parameter), the error nr::ERNOD is set and the pointer to the UT array is returned.

5.3.2.6 int sum::getn ()

Returns:

Current length of series (equal to all, taken from UT series)

5.3.2.7 void sum::done ()

If the summary file *sumfile*, specified in **init**() (p. 20) call, does not exist, create it with a header-line with names of parameters and leading comment-char '#'. Then, if some scientific measurements were performed, compute the night-time median parameters from the series and add a new line in a summary-file. This line is written after the last line in a file, or, if the line with the same date is found, on the place of this obsolete record.

If a "dump-file" name *dumpfile* was supplied to **init**() (p. 20), the series are (over)written in some binary format in this file. Before quitting, the series storages are released.

The format of a dump-file is following. This binary file consists of a series of records of double-type numbers, except for the first record. The first record is a descriptor; it consists of four **long integer** numbers

```
<date DDMMYY> <Number_of_series> <length_of_each_series> <maximal N_layers>
```

Series include the one for fractional UT, one for airmass etc. Then follow <`Number` of series> records -

```
<parameter_(start_UT)> ... <parameter_(end_UT)>
```


© 1997-2002

i.e. the contents of the series storages (arrays), each of the length <`length_of_each_series`>.

Thus, all the internally stored information of the module is written in the dump-file. This format is not fixed (not "official") since the dump-file is a "black-box" devoted for internal use in the SUMMARY module only.

5.3.2.8 double massmag (double *vmag*, double *bv*, double *coleq* = sum::MASSCE) `[inline]`

Parameters:

- ***vmag*** V-band magnitude
- ***bv*** B-V color
- ***coleq*** Linear color equation coefficient

This function converts the standard V-magnitude of a star into the MASS-band magnitude using the linear color equation with a coefficient *coleq*. By default, this coefficient is taken from the module constant **sum::MASSCE** (p. 24). For a star having (B-V)==0 (A0), the MASS magnitude is assumed to be equal to V-magnitude.

5.3.3 Variable Documentation

5.3.3.1 const double sum::NODATA = 0

The value reserved for the unknown parameter value. Returned on failure; don't enter the averaging if encountered in a set.

5.3.3.2 const double sum::MASSCE = 0.5905

MASS color equation coefficient from linear regression of a synthetic relation Mag_{mass}-V vs (B-V)_0: Mag_{mass} = V + MASSCE * (B-V)

6 Part I. File Documentation

6.1 access.h File Reference

Deals with access rights to files, fields of CFG etc., determines the current user status: "mass-root" (expert) or not (simple user).

© 1997-2002

6.1.1 Detailed Description

MASS project: TURBINA module file header file for access.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.2 cfg.h File Reference

Tools for accessing to CFG tree in memory: search, validation, I/O.

`#include <stdio.h>`

`#include <qstring.h>`

`#include "names.h"`

`#include "tree.h"`

6.2.1 Detailed Description

MASS project: TURBINA module file header file for cfg.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.3 cfgdialog.h File Reference

Module for implemention of the GUI-dialogs of a user with a tree of CFG parameters.

`#include <qdialog.h>`

`#include <qpushbutton.h>`

`#include <qpopupmenu.h>`

`#include <qlabel.h>`

`#include <qlineedit.h>`

`#include <qstring.h>`

```
#include <qlayout.h>
#include <qarray.h>
#include <qvalidator.h>
#include <qcombobox.h>
#include "maintree.h"
```

Compounds

- class **FloatValidator**
- class **IntValidator**
- class **TimeValidator**
- class **DegreeValidator**
- class **SubItem**
- class **JKQLineEdit**
- class **JKQComboBox**
- class **BranchDialog**
- class **CFGDialog**

6.3.1 Detailed Description

MASS project: TURBINA module file header file for cfgdialog.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.4 cnt2asc.cpp File Reference

Reading the MASS binary count-file *.cnt and converting the count data into an ASCII table. A standalone program.

```
#include "iocount.h"
#include "nrutil.h"
#include <iostream>
```

6.4.1 Detailed Description

Usage

```
./cnt2asc <MASS_count_file_name> [address [Nbasetime]]
or
./cnt2asc [-h] to get a short help
```

The program opens the *MASS_count_file_name* and seeks the position *address*. By default, the beginning of file is sought. Then the *Nbasetime* records (1 by default) are read and printed in standard output as an ASCII table with the number of columns equal to the number of channel buffers (active MASS channels when writing the count-file) and row number equal to the number of counts received in each channel during *Nbasetime* base times.

The *address* is accepted either in decimal or in hexadecimal format (preceded with "0x" in latter case). It is normally given in the last field of the base-time scintillation index record (format of this address is "@<hex_offset>"). This address is converted into the long integer number and used as an *address* for **ioc::read**() (p. 12) command to read data from the count-file.

It is assumed that the count-file ∗.cnt is written with the same or compatible version of iocount.cpp module as used here.

The number of count buffers and the length of buffers (checked to be the same for all buffers of the same base-time record) are read from the descriptor to which the *addr* points. The constraint for several base-time records read in one turn (*Nbasetime>1)* is that they all have the same buffers' number and the length of any buffer must not exceed the one of the first buffer. If either of these conditions are violated, the reading is stopped with a message to stderr.

Returns:

On success, the long integer address of the *next* base-time record is returned; on error -1 is returned and the message (normally, from the IOCOUNT module) is printed in standard error output. The change of buffer's number or length is not accounted as error; the address of this differing record is returned.

Compilation:

```
g++ -Wall -o cnt2asc cnt2asc.cpp nrutil.o iocount.o
```

Author:

N.Shatsky (SAI,2002)

Version:
1.0

6.5 color.h File Reference

The list of the color names (character constants) reserved in Qt (17 colors).

6.5.1 Detailed Description

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.6 counter.h File Reference

Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with photometric modules.

`#include <qarray.h>`

`#include "iocount.h"`

Compounds

- class **BlockCounter**
- class **Counter**

6.6.1 Detailed Description

MASS project: TURBINA module file header file for counter.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.7 decldevicetree.h File Reference

The "body" of tree from **devicetree.h** module: the list of CFG parameters itself.

6.7.1 Detailed Description

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.8 declmaintree.h File Reference

The "body" of tree from **maintree.h** module: the list of CFG parameters itself.

6.8.1 Detailed Description

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.9 device.h File Reference

High-level interaction with the device bearing the manager functions which implement the algorithms of modes.

```
#include <qobject.h>
#include "exchange.h"
#include "counter.h"
#include "light.h"
#include "motor.h"
#include "power.h"
#include "scan.hxx"
#include "trbtime.h"
```

Compounds

- class **PointDouble**
- class **DevManager**

6.9.1 Detailed Description

MASS project: TURBINA module file header file for device.cpp

The algorithms represent the sequences of actions which implement the one accumulation-time cycle of measurements by means provided by **counter.h**, **motor.h**, **power.h** and **light.h**.

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.10 devicetree.h File Reference

Keeps (statically) in memory the tree (parameter values) of the CFG stored in `device.cfg`.

`#include "tree.h"`

`#include "decldevicetree.h"`

Compounds

- class **DeviceTree**

6.10.1 Detailed Description

MASS project: TURBINA module file header file for devicetree.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.11 exchange.h File Reference

Low-level exchange implementation of the program with the device driver.

`#include "qstring.h"`

`#include "counter.h"`

`#include "trbconst.h"`

Namespaces

- namespace **exch**

6.11.1 Detailed Description

MASS project: TURBINA module file header file for exchange.cpp

Implements the commands sending, data and signals receiving from the device via the driver inserted into OS kernel.

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.12 graph.h File Reference

Supports the output of the measurement results in the graphic window.

```
#include <qpainter.h>
#include <qpixmap.h>
#include <qwidget.h>
#include "outcontrol.h"
```

Compounds

- struct **Scale**
- class **GraphWindow**

6.12.1 Detailed Description

MASS project: TURBINA module file header file for graph.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.13 iocount.h File Reference

Input/Output of the individual photon counts in a binary "count-file".

```
#include <stdint.h>
#include <stdio.h>
```

Namespaces

- namespace **ioc**

Typedefs

- typedef uint16_t **count_t**

Variables

- const **count_t** **max_count_t** = (**count_t**)(-1)

6.13.1 Detailed Description

MASS project: header file for iocount.cpp

This module implements the Input/Output operations with disk for saving and restoring the counts in count buffers of MASS detectors. The aim is to allow for saving the night-time count series in a binary file and for the *off-line* playback the recorded series of input channel counts to repeat their processing and display the output parameters. Alternatively, the counts may be read and processed externally.

The counts from all channels of the device are written in a binary format by the utility **ioc::write**() (p. 12) which returns the *start-position address* of the written series. This address is saved somewhere else (e.g. in main MASS output file). Afterwards, for reading the *count-file*, the address is fed to **ioc::read**() (p. 12) which restores the counts in the channel buffers. After this, the processing may go on as if the count data were just received from counters.

The binary count file should be opened with an attribute "a+b" and closed upon completion of reading/writing externally. If problem occurs (bad input address for **ioc::read**() (p. 12) or low-level I/O problem), the error is set by these utilities to be detected by nr::erget() and nr::ermessage().

The buffers are made of numbers of the type **count_t** which is defined in this module. All the bytes of count_t numbers are written to the disk. Thus,

© 1997-2002

the two-byte unsigned or signed integers are seemed to be best suited for MASS, for which we do in this module:

```
#include <stdint.h> // or <sys/types.h>
    ...
typedef count_t uint16_t ;  // or int16_t
```

Normally, the buffers are written by **ioc::write**() (p. 12) or read by **ioc::read**() (p. 12) simultaneously if they contain the signal of the same time interval of acquisition (*base-time).* Thus, for 4-channel MASS device, *four* buffers are written/read in one time. Each these count buffers is preceded by a *descriptor* consisting of two numbers of the same type **count_t**.

The first number is a number of count buffers which are written starting from the current position and which belong to the **same** base-time. This allows to check that the buffers are synchronized in acquisition time while reading the count file.

The second number is the length of the following buffer in counts (one should multiply this number by the sizeof(count_t) to get length in bytes). Currently, MASS channels are forced to fill their count buffers always with equal amount of counts.

The example of the record of one base-time of MASS with 4 channels each having 1000 counts (binary count_t-type numbers are denoted as <..>) :

```
<4> <1000> <channel_1 count_1> <channel_1 count_2> ... <channel_1 count_1000>
<3> <1000> <channel_2 count_1> <channel_2 count_2> ... <channel_2 count_1000>
<2> <1000> <channel_3 count_1> <channel_3 count_2> ... <channel_3 count_1000>
<1> <1000> <channel_4 count_1> <channel_4 count_2> ... <channel_4 count_1000>
```

Thus, reading the first two bytes of the file or of the base-time record allows one to get know the number of buffers written further for this base-time and the size of (each) buffer.

Author:

N.Shatsky (kolja@sai.msu.ru)

Version:

1.1 Descriptor added for each record

6.13.2 Typedef Documentation

6.13.2.1 typedef uint16_t count_t

The count type will be unsigned 16-bit integer type

6.13.3 Variable Documentation

6.13.3.1 const count_t max_count_t = (count_t)(-1)

Maximal count_t (implementation works for unsigned integral types)

6.14 light.h File Reference

Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with buttons and control/illumination LEDs module.

`#include <qstring.h>`

Compounds

- class **Light**

6.14.1 Detailed Description

MASS project: TURBINA module file header file for light.cpp

Author:

O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:

1.0

6.15 main.cpp File Reference

Initiates the program execution.

`#include <qthread.h>`

`#include <qapplication.h>`

`#include <qfont.h>`

`#include <qdatetime.h>`

`#include "exchange.h"`

`#include "names.h"`

`#include "device.h"`

`#include "cfg.h"`

```
#include "trbtime.h"
#include "mywidget.h"
#include "maintree.h"
#include "devicetree.h"
#include "graph.h"
#include "trbexport.h"
#include "access.h"
#include "outcontrol.h"
```

Compounds

- class **DataThread**
- class **CountThread**

6.15.1 Detailed Description

MASS project: TURBINA main file

This is a top-level module of TURBINA program, a starting point of a program where the following functions are executed.

- Reservation of static variables for **trbexport.h**,
- start of CFG files reading into the trees in memory,
- start the "Scenario" and "Data" threads,
- initialize the device (if `turbina.cfg` contains "Yes" in `Operations/Common/WithDevice`),
- create the main and graphic windows, load the module constants,
- If Initscenario is activated in CFG, start this scenario in the "Scenario" thread (all the other similar starts are made via GUI).

Since many of the class constructors (from other modules) are (implicitly) started here, on the stage of the program startup all the checks of consistency are done (e.g. existence and validity of weight function file for all stars in the list, etc.).

Author:

O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:

1.0

6.16 maintree.h File Reference

Keeps (statically) in memory the tree (parameter values) of the CFG stored in `turbina.cfg`.

`#include "tree.h"`

`#include "declmaintree.h"`

Compounds

- class **MainTree**

6.16.1 Detailed Description

MASS project: TURBINA module file header file for maintree.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.17 messages.h File Reference

GUI-output of the software messages to the screen.

`#include <qstring.h>`

6.17.1 Detailed Description

MASS project: TURBINA module file header file for messages.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.18 motor.h File Reference

Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with stepper motor of the aperture wheel.

`#include <qstring.h>`

Compounds

- class **Motor**

6.18.1 Detailed Description

MASS project: TURBINA module file header file for motor.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.19 mywidget.h File Reference

GUI-implementation of the program main window: menu structure, status bars etc. linking together all the components provided by **widgets.h**.

```
#include <qmainwindow.h>
#include <qtoolbar.h>
#include <qbuttongroup.h>
#include <qpopupmenu.h>
#include <qstring.h>
#include <qlabel.h>
#include <qtoolbutton.h>
#include <qiconset.h>
#include <qprogressbar.h>
#include <qarray.h>
#include "widgets.h"
#include "cfgdialog.h"
```

Compounds

- class **MyWidget**

6.19.1 Detailed Description

MASS project: TURBINA module file header file for mywidget.cpp

Author:

O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:

1.0

6.20 names.h File Reference

Reserved string constants: file names, paths, extentions and the numeric formats for data display on the screen.

6.20.1 Detailed Description

Note that the formats of data which are stored in the mass-file are defined in scientific modules of TURBINA.

Author:

O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:

1.0

6.21 outcontrol.h File Reference

Manipulation of the output in the main results and graphic results windows.

```
#include <qobject.h>
#include <qstring.h>
#include <qcolor.h>
#include "summary.h"
```

Compounds

- class **GraphSet**
- class **GraphData**
- class **ShowSet**
- class **DisplayData**
- class **OutControl**

6.21.1 Detailed Description

MASS project: TURBINA module file header file for outcontrol.cpp

The module "tells" the other modules which data should be displayed and which should be hidden depending on the (current) state of CFG parameters which are listed in the `Display` section of `turbina.cfg`.

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.22 power.h File Reference

Implementation of interaction (command calls and replies from modules, their current data and status keeping, etc.) with high voltage supply module.

`#include <qstring.h>`

Compounds

- class **Power**

6.22.1 Detailed Description

MASS project: TURBINA module file header file for power.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.23 readmass.h File Reference

Reading the content of the mass-file for Playback function and CFG parameters last-written values.

`#include <stdio.h>`

`#include <string>`

Namespaces

- namespace **rdm**

6.23.1 Detailed Description

This module implements reading and parsing of lines of a *mass-file*. This is done in course of following tasks:

- comparison of the configuration file (CFG) parameters written previously in a current mass-file with that read from the CFG (repetitive start of TURBINA in the same night). Deals with the so-called *preamble-type* lines ('P'-lines).
- the "playback" of an old mass-file (normally named as `YYMMDD.mass.in`) for re-reducing of the previous nights data. Here, apart from the 'P'-lines, we take care mostly of the *base-time* originated lines (as that with prefixes 'i','j') and new modes declarations ('M'-lines). Also, the object info lines ('O') are parsed.

Usage

Reading the single line:

The function **rdm::readline**() (p. 14) reads and (possibly) parses the (next) line from the opened stream of the mass-file. It returns the line prefix which denotes the type of line and parsing results. Some line type are parsed and the rest are simply provided "as is" by **rdm::getvalue**() (p. 16) (see below).

The parsed data are available with following functions:

- **rdm::gettype**() (p. 15) returns the line prefix just as **rdm::readline**() (p. 14) does, for **all** types, naturally;
- **rdm::isheader**() (p. 15) tells for the comment-type lines (prefix '#') whether the line is a header-line (second character is a prefix also).
- **rdm::getut**() (p. 16) returns the UT moment reference, for **all** types except for header-lines (starting with '#*' where '*' is a prefix character);
- **rdm::getname**() (p. 16) returns the name of parameter (**'P'-type)** or started mode (**'M'-type)** or catalogue (HR) number of a star (**'O'-type)** in **upper-case**;
- **rdm::getvalue**() (p. 16) [no arguments] returns the parameter value for **'P'-type** line, or the content of the line to the right of the star number for **'O'-type** (name, coordinates etc.) or to the right of UT reference for *non-parsed* lines;

- **rdm::getaddr**() (p. 14) returns the *count-file* address, for **'i','j'-types**;

The parsed parameters are stored in global (private) variables, and are modified solely by overwriting. This means that, for example, the count-file address returned by **rdm::getaddr**() (p. 14) will stay unchanged while the lines without one are read.

Reading the whole mass-file for the parameter values set:

The function **rdm::grabparams**() (p. 17) should be called on the opened mass-file stream to collect the preamble-type information. The pairs of parameter names and their associated latest assigned values are stored in a *map*. Then, these latest written parameter values are available with **rdm::getvalue**() (p. 16) with an argument **exactly equal** to the parameter name as a *key* of this map (in upper-case, as returned by **rdm::getname**() (p. 16)).

Note:

No error is set by this simple module to be checked as nr::erget(). The reading failures are to be traced by the return values (see e.g. **rdm::getut**() (p. 16))

Author:

N.Shatsky

Version:

1.0 Creation

1.1 Bugs (minor) correction, overloaded getut()

6.24 star.h File Reference

Reads the star list and interprets the star parameters when needed.

```
#include <qobject.h>
#include <qstring.h>
#include <qarray.h>
#include <qdatetime.h>
#include "trbconst.h"
```

Compounds

- class **StarInfo**

© 1997-2002

- class **Star**

6.24.1 Detailed Description

MASS project: TURBINA module file header file for star.cpp

Can read the list of target stars, to search for the file of the weight functions corresponding to the given star SED, keeps the current star parameters and position on sky.

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.25 summary.h File Reference

Collection of scintillation indices and atmospheric parameters results from scientific modules for their graphic representation and the summary-file handling.

`#include <string>`

Namespaces

- namespace **sum**

Variables

- const char **SUMDLM** [] = " "
  delimiter of a summary file.

6.25.1 Detailed Description

The module is aimed to collect the output results (averages over *Accumulation* Time) of scintillation measurements and atmospheric calculations during the night and update the so-called *summary-file* of MASS with their night-time median parameters. In this file, the set of these averages is given as a single line per program start (thus, normally, per night) with some reference and statistic supplementary information (the evening date, UT time

span covered, number of obtained and medianed results). The format of a summary-file is described in MASS User Guide.

An access to the result storages and references with **sum::getdata**() (p. 22) and **sum::getref**() (p. 21) and while parsing the header of a summary-file in **sum::init**() (p. 20) is done via their character names (e.g. "SI_A" for an A-channel scintillation index). The relation between the names (both given in the head-line of a summary-file and supplied as a parameter in getdata() etc.) and the respective result structures is installed via an internal *associative map*.

In addition, the results of previous nights are read from this summary-file to have *reference* while graphically presenting the series of the current night results. The continuously accumulated series of results serve both for the end-of-night median averaging in order to update the summary file and for providing the arrays of points for graphic representation during the night.

In the hierarchy of TURBINA modules, this module thus stays above the scientific modules ATMOS and SCIND. It "knows" which scintillation and atmospheric parameters to access and how to write them in file or read from file (the same formats are used as in the *mass-file)*.

Usage

Before starting the measurements, the module has to be initialized by **sum::init**() (p. 20) with a name of a summary-file and a number of the MASS device channels. The summary-file is read (if exists) and the *reference* value for each parameter is stored - the ones from the first and last summary-file lines ("First mean" and the "Last mean") and an average through all records in a file ("Season mean"). These reference values are accessible wit **sum::getref**() (p. 21). Also, the "dump-file" name of accumulated series may be supplied to restore the content of the series if TURBINA was restarted in the same night.

Upon completion of each Accumulation Time of scientific measurements (Normal or Generalized mode), the function **sum::add**() (p. 21) has to be invoked to add the parameters in tails of their series. After this, the series graphics may be updated using **sum::getdata**() (p. 22); their length are taken from **sum::getn**() (p. 23).

Before exit of TURBINA, the function **sum::done**() (p. 23) must be called. If some scientific measurements were performed, it computes the night-time median parameters from the series. The summary file is created with a header line if it did not exist; the name is taken as that supplied to **sum::init**() (p. 20). If this is the first start in the current line, a new line of results is added in a summary-file; if the record with the current date is already present in the file, it is overwritten.

Attention:

Since the summary-file is overwritten in case of a repetitive TURBINA run, a loss of information is possible (cause is, e.g., overfill of a numeric format leading to discrepant sizes of the old and new records). So, the summary-file is better to save somewhere in the beginning of a new night.

If a "dump-file" name was supplied to **sum::init**() (p. 20), the series are written in a binary format in this file. Before quitting, the series storages are released.

Author:

N.Shatsky (`kolja@sai.msu.ru`)

Version:

1.0 Creation

6.26 tools.h File Reference

A variety of functions and tools needed for all other modules.

`#include "trbtime.h"`

Functions

- const char * **getStamp** (const char *prefix, Time time, const char *ext=NULL)

  Return the concatenation of a character argument and QString.

6.26.1 Detailed Description

MASS project: TURBINA module file header file for tools.cpp

Author:

O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:

1.0

© 1997-2002

6.26.2 Function Documentation

6.26.2.1 const char* getStamp (const char * *prefix*, Time *time*, const char * *ext* = NULL)

Parameters:

char leading string (normally, a single character

string string to append

Returns:

C-string as char.arg+Qstring (normally, the stamp for writing in mass-file

6.27 trbconst.h File Reference

Commonly used constants (e.g. MM2CM, SEC_IN_DAY etc.).

6.27.1 Detailed Description

Author:

O.Voziakova `ovoz@sai.msu.ru`

Version:

1.0

6.28 trberror.h File Reference

The declarations of C++ exception classes which may be invoked from some points in **star.h**, **device.h** etc.

```
#include <qstring.h>
```

Namespaces

- namespace **trberr**

Compounds

- class **ErrorBase**
- class **LoadError**
- class **CFGError**
- class **DeviceError**

6.28.1 Detailed Description

MASS project: TURBINA module file header file for trberror.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.29 trbexport.h File Reference

The class is presented with static fields which serve for exchange of data between the Scenario thread and Main-thread of the program.

`#include <qstring.h>`

Compounds

- struct **DeviceState**
- struct **Progress**
- class **ExportText**
- class **ExportTable**
- class **InfoExport**
- class **StateExport**

6.29.1 Detailed Description

MASS project: TURBINA module file header file for trbexport.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.30 trbtime.h File Reference

Support of date/time-related tasks. Implements namespace trb_tm and site stellar time, coordinates, UT etc.

`#include <qobject.h>`

`#include <qdatetime.h>`

Namespaces

- namespace **trb_tm**

Compounds

- class **ForSlot**
- class **Time**

6.30.1 Detailed Description

MASS project: TURBINA module file header file for trbtime.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.31 tree.h File Reference

Class implementing the CFG tree organization: principles of its building, reading, searching info etc.

`#include <qobject.h>`

`#include <qstring.h>`

Compounds

- class **Link**
- class **CFGLeaf**
- class **CFGBranch**
- class **CFGTree**

6.31.1 Detailed Description

MASS project: TURBINA module file header file for tree.cpp

Author:
O. Voziakova, Sternberg Institute (`ovoz@sai.msu.ru`)

Version:
1.0

6.32 unroll.h File Reference

Converts the scenario formula in the sequence of mode symbols.

`#include <string>`

Functions

- const string & **unroll** (const char *rec, const char *dict)

  Unroll the scenario record in a string of mode symbols.

6.32.1 Detailed Description

MASS project: header file for unroll.cpp

The module implements the text parser for "unrolling" the MASS scenario records. The record is the string where the different modes follow after each other as "A+B" and different sequences of modes of an arbitrary length are repeated a certain amount of times as "20*(A+B)". The idea of implementation of this module comes from the "calculator" program by Bjarne Stroustrup (see "Jazyk Programmirovanija C++", 3rd edition, Moscow, Binom, 2001, p. 147).

6.32.2 Function Documentation

6.32.2.1 const string& unroll (const char * *rec*, const char * *dict*)

Parameters:

rec scenario record

dict "dictionary": a list of allowed mode symbols to reject the erroneous input. Case sensitive.

Returns:

string which is an exact sequence in which the planned modes will be started

The allowed operators are

- "+" addition for adding the modes or the sequences of modes. Addition of numbers or numbers and modes is not allowed;

© 1997-2002

- "*" multiplication - for repeating the sequences or modes many times. Defined only for integer numbers or modes/sequences and integer numbers, in arbitrary order.
- "(...)" for grouping the modes in sequences, with arbitrary degree of enclosure.

Only the integer numbers are allowed for multiplication. Spaces are allowed in a record.

Example:

```
string scenario = unroll("2*(A+3*B+C)","ABC") ;
```

returns "ABBBCABBBC".

6.33 viewstar.h File Reference

Shows the list of stars with help of GUI and lets user to choose the star from this list.

`#include <qstring.h>`

`#include <qlistview.h>`

`#include <qarray.h>`

`#include "star.h"`

Compounds

- class **JKQListViewItem**
- class **ViewStarList**

6.33.1 Detailed Description

MASS project: TURBINA module file header file for viewstar.cpp

Author:
O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:
1.0

6.34 widgets.h File Reference

Implementation of the different components the main window of TURBINA collected together by **mywidget.h**.

```
#include <qthread.h>
#include <qapplication.h>
#include <qpushbutton.h>
#include <qdialog.h>
#include <qlayout.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qmultilinedit.h>
#include <qcombobox.h>
#include <qtableview.h>
#include <qtextview.h>
#include <qlcdnumber.h>
#include "viewstar.h"
#include "trbexport.h"
#include "device.h"
```

Compounds

- class **QJKPushButton**
- class **QJKTable**
- class **HelpView**
- class **QJKMainView**
- class **TalkDialog**
- class **AdjustDialog**
- class **CenteringDialog**
- class **CommentDialog**
- class **ShowStarDialog**
- class **SelectStarDialog**
- class **IllumDialog**
- class **LightDialog**
- class **HVDialog**
- class **BackCorrectionDialog**

- class **CenterCorrectionDialog**
- class **DigitalClock**

6.34.1 Detailed Description

MASS project: TURBINA module file header file for widgets.cpp

Author:

O. Voziakova, Sternberg Institute (ovoz@sai.msu.ru)

Version:

1.0

© 1997-2002

Index

access.h, 24
add
    sum, 21
add2map
    rdm, 17

cfg.h, 25
cfgdialog.h, 25
cnt2asc.cpp, 26
color.h, 28
count_t
    iocount.h, 33
counter.h, 28
createset
    sum, 20

decldevicetree.h, 28
declmaintree.h, 29
device.h, 29
devicetree.h, 30
done
    sum, 23

exchange.h, 30

FIRSTMEAN
    sum, 20

getaddr
    rdm, 14
getdata
    sum, 22
getn
    sum, 23
getname
    rdm, 16
getref
    sum, 21
getStamp
    tools.h, 45
getstartpos
    rdm, 15
gettype
    rdm, 15
getut
    rdm, 16
getvalue
    rdm, 16
grabparams
    rdm, 17
graph.h, 31

init
    sum, 20
ioc, 11
    read, 12
    write, 12
iocount.h, 32
    count_t, 33
    max_count_t, 34
isheader
    rdm, 15

LASTMEAN
    sum, 20
light.h, 34

main.cpp, 34
maintree.h, 36
MASSCE
    sum, 24
massmag
    sum, 24
max_count_t
    iocount.h, 34
messages.h, 36
motor.h, 36
mywidget.h, 37

names.h, 38
NOADDR
    rdm, 18
NODATA
    sum, 24

outcontrol.h, 38

partype
    sum, 19
partype_name
    sum, 18
power.h, 39

rdm, 13
    add2map, 17
    getaddr, 14
    getname, 16
    getstartpos, 15
    gettype, 15
    getut, 16
    getvalue, 16
    grabparams, 17
    isheader, 15
    NOADDR, 18
    readline, 14
read
    ioc, 12
readline
    rdm, 14
readmass.h, 39
reftype
    sum, 19

SEASONMEAN
    sum, 20
star.h, 41
sum, 18
    add, 21
    createset, 20
    done, 23
    FIRSTMEAN, 20
    getdata, 22
    getn, 23
    getref, 21
    init, 20
    LASTMEAN, 20
    MASSCE, 24
    massmag, 24
    NODATA, 24
    partype, 19
    partype_name, 18
    reftype, 19
    SEASONMEAN, 20
SUMDLM
    summary.h, 42
summary.h, 42
    SUMDLM, 42

tools.h, 44
    getStamp, 45
trbconst.h, 45
trberror.h, 45
trbexport.h, 46
trbtime.h, 46
tree.h, 47

unroll
    unroll.h, 48
unroll.h, 48
    unroll, 48

viewstar.h, 49

widgets.h, 50
write
    ioc, 12