GCOS
Reference
Upper-
Air
Network

GRUAN Technical Note 6

Brief Description of GruanToolRsLaunch (gtRsl)

Michael Sommer

Publisher

GRUAN Lead Centre

Number & Version

GRUAN-TN-6
Rev. 1.0 (2020-08-24)

Document info

Title:	Brief Description of GruanToolRsLaunch (gtRsl)
Topic:	Software Manual
Authors:	Michael Sommer
Publisher:	GRUAN Lead Centre, DWD
Document type:	Technical Note
Document number:	GRUAN-TN-6
Page count:	36
Version:	Rev. 1.0 (2020-08-24)

Abstract

The GRUAN software tool *gtRsl* is a Java-based command-line utility intended for automatic creation of GMD (GRUAN meta data) files for radiosonde launches (“RsLaunch”). The GMD files are mandatory to import any radiosonde launches into the GRUAN data archive. The *gtRsl* is a useful alternative to the manually operated and GUI-guided *GruanRsLaunchClient (RLC)* in following cases: Routine flights with a single radiosonde, stable launch setup with few operational changes over time, or auto-launcher systems. The current version of this brief description is related to version 0.5.x of *gtRsl*.

Contacting the GRUAN Lead Centre

Note: Please consult the GRUAN Lead Centre before use of this tool: gruan.lc@dwd.de.

Note: Please contact the author or the GRUAN Lead Centre (gruan.lc@dwd.de) in case file formats are not supported, functionalities are missed, or if bugs or errors occur.

Editor remarks

Note: Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by GRUAN.

Revision history

Version	Author / Editor	Description
1.0 (2020-08-24)	Michael Sommer	First published version as GRUAN-TN-6
0.9.2 DRAFT (2020-08-21)	Michael Sommer	Corrections and added descriptions as answer of the LC-internal review
0.9.1 DRAFT (2020-08-12)	Christoph von Rohden	LC-internal review
0.9.0 DRAFT (2020-07-03)	Michael Sommer	Final preparations for the review
0.5.3 DRAFT (2020-04-29)	Michael Sommer	Small corrections included
0.5.2 DRAFT (2019-10-29)	Michael Sommer	New appendix “Evaluation formulas” added
0.5.1 DRAFT (2019-10-25)	Michael Sommer	Small corrections included
0.5.0 DRAFT (2019-04-23)	Michael Sommer	First version which is related to v0.5.x of *gtRsl*
0.4.1 DRAFT (2019-03-29)	Michael Sommer	Small corrections included
0.4.0 DRAFT (2019-03-27)	Michael Sommer	First draft version as GRUAN-TN-x using LaTeX
0.3.41 DRAFT (2017-12-12)	Michael Sommer	Draft version as GRUAN-IN-6

Table of contents

Appendix

1 Installation

1.1 System requirements for running GruanToolRsLaunch

For using the *GruanToolRsLaunch (gtRsl)* following requirements should be satisfied:

- computer with any operating system (like Windows, Linux, Mac OS, . . .),
- installed Java version 8 (newer versions are not yet tested).

1.2 Download and install Java

An actual Java Runtime Environment (JRE) should run under the used operating system. In case the JRE version is outdated or not installed you can download the current version from:

- Oracle: www.java.com/en/download (last free update from January 2019), or
- AdoptOpenJDK: adoptopenjdk.net (security updates until at least September 2023)

and install on your operating system.

1.3 Download GruanToolRsLaunch

The current version of *gtRsl* is available at the GRUAN website (www.gruan.org/data/software/gtrsl). A user login is required for download. Please register at www.gruan.org/user/registration, if required.

1.4 Installing GruanToolRsLaunch

The few installation files are packed in a zip-file. The main program file is a Java executable archive file (“.jar”). Other files (“gtRsl.*”) are launching scripts for different operating systems, like “gtRsl.bat” for Windows.

Unpack all files into a folder of your choice. The program (or the folder) should then manually be announced to the system. There are two options:

- Edit the *PATH* system variable and add the directory of installation.
- Edit the dedicated script and correct the (absolute) path to the “*.jar”-file, than copy this script into a folder that is known to the system for executable programs (e.g. a “/bin” folder).

2 General usage

The usage is as usual for command line tools. The program is controlled in detail by a number of available options, some of which are mandatory.

```
gtRsl [OPTION]... [FILE]...
```

As usual with command line programs, any number of options can be specified after the command in any order. The complete list of all available options is available in the appendix A. More detailed explanations of each option can be found in Section 3.

There are three “main” options, one of which must be selected:

- *-r or --run* – run the tool for a list of files and/or directories
- *-h or --help* – print the help page with a short description of all options
- *-v or --version* – print information about the software version

Therefore, for normal use of this tool, the *-r* or *--run* option must always be specified (see Section 3.1). It is generally equivalent whether the short or long form of an option is used. In addition to the main options, there is a number of further options that can optionally be specified to adapt the operation of the actual command. However, if the *-r* option is selected, two additional options *-t* and *-g* are mandatory. With *-t* a template file is assigned, which is a template for the GRUAN meta data files to be generated (see Section 3.2). The option *-g* is used to assign a file which defines general meta data as well as documented changes of these meta data over time (see Section 3.3).

Following the options, at least one source file or directory must be specified. Instead of a single file/directory a space-separated sequence of files and/or directories can be passed to the actual command. The complete list of these files (or, if a directory is given, all files in this directory) is then processed.

To give an idea of how to use the tool, a simple example is shown and discussed here:

```
gtRsl -r -t "test-template.ftl" -g "test-changes.txt" -u -o "upload" "test-data"
```

The *gtRsl* command is followed by an unordered sequence of options, with some working as switches (without passing values), such as *-r*, and others used to define names, values, or other keywords, such as *-t “temp-template.ftl”*.

The first option in the example *-r* switches the program into normal processing mode. This requires two more mandatory options. With *-t “test-template.ftl”* the template file to be used is assigned, and *-g “test-changes.txt”* transfers the desired list of meta data including possible temporal changes. The next option *-u* allows the program to independently determine the time range covered by the soundings selected for processing (see Section 3.4).

The last option *-o “upload”* defines a path to a directory where the created GRUAN meta data (GMD) files are to be stored. If the target directory is not given with an absolute path, then it is treated as relative to the working directory. The specification of the file or directory which contains the original sounding files (see Section 3.7) is given with the very last parameter (“test-data” in the example).

Note that before executing the command, the template file (see Section 3.2) as well as the file containing the list of metadata (Section 3.3) have to be properly adapted or created.

Note: The creation of the template and metadata list files should be done in consultation with the GRUAN Lead Centre (gruan.lc@dwd.de).

In addition to this first example some more examples can be found in Section 4.

3 Command options

3.1 Option -r, --run

The option *-r* is the key option for a normal processing of the input file, i.e. it is mandatory for the creation of GMD (GRUAN meta data) files. When using this option, two further options are required: *-t* (Section 3.2), and *-g* (Section 3.3).

But how does the processing of input data work? In a first step, all directories and files to be used as input are analysed. The processing of a number of different file formats is currently built-in (see Section 3.3.4). An attempt is made to extract a number of metadata, e.g. start date, radiosonde serial number, etc. Files to be excluded from processing can be filtered out (blocked, see Section 3.5). Afterwards a time grid is created according to the defined scheduling (see Section 3.4) and the time steps are run through one after the other. This procedure checks for input files matching a time step. If matching input files are found, the appropriate GMD file is created and saved. Depending on the selected options, the GMD files are stored together with the input data in a structured way (see Section 3.7) and optionally uploaded to the GRUAN LC (see Section 3.8).

3.2 Option -t, --template <FILE>

The template option is required for a run, because a template file is necessary to create the GMD files.

Note: Please contact the GRUAN Lead Centre (gruan.lc@dwd.de) for assistance in adapting or creating a template file.

The appointed template file must be a valid FTL file. The full definition of FTL files is not given here, but some key properties and examples. FTL files should be based on GMD files. There are some example FTL files in the sub-directory “/templates”. A FTL file can include specific (or user-defined) place holders. These place holders are used to ensure that the appropriate metadata can later be written in the right places in the GMD files to be generated. A place holder has the following syntax:

```
${place_holder_name}
```

There are some pre-defined place holders which are automatically filled with valid values, according to the following list:

- *creation_timestamp* – date and time of processing
- *creation_comment* – short comment for automatic creation
- *sonde_sn* – serial number of primary sonde (only available, if it can be extracted automatically)
- *launch_number* – number of launch of a schedule date [default: 1]. If a double launch is detected, the second launch is assigned the launch number 2.

In addition to the rather simple pre-defined place holders, there are some constructs for managing data file lists.

The following syntax can be used in the template file for including optional text in the result GMD file. The variable *required_data* is true, if all required data files exist exactly once (see also Section 3.3.3 to get an idea about required or optional data files).

```
<#if !required_data> ... text to include ... </#if>
```

The following syntax can be used for including a list of data files. The text lines in this construct will be included n times (with n the number of data files).

```
<#list data_files as file>
... text to include ...
<#/list>
```

Several place holders are pre-defined for use in the file list block:

- *file.fileName* – name of data file (with extension), e.g. "data_file.tab"
- *file.filePath* – local absolute path of data file
- *file.fileLength* – length of data file [byte]
- *file.fileChecksum* – checksum of data file [CRC32]

Note: Please find example template files in Appendix B.

3.3 Option -g, --change-list <FILE>

The change-list option is required for a run, because a change-list file is necessary to create the GMD files. Such a file contains all relevant general meta-data (related to the site, instruments, equipment, etc.) and furthermore allows to define change events of these meta-data over time.

Note: Please contact the GRUAN Lead Centre (gruan.lc@dwd.de) if first time a new or adapted change-list file should be created.

A change-list file should follow a specific format. Each line with a change definition should consist of three parts:

```
<key>;<timestamp>;<value>
```

The three parts are separated by a semicolon ";". They have to follow the scheme:

- *key* – unique key of a pre-defined or user-defined place holder
- *timestamp* – start date&time from which the given value applies
- *value* – value to be used (number or string)

Note: Example change-list files can be found in appendix B.

3.3.1 Comments

It is possible or even desired to include comment lines for additional information. Comment lines have to start with a hash mark "#", e.g.

```
# ------------------------------------
# Balloon type and filling
# ------------------------------------
#
# TODO Please add here additional lines to display any change
# (always 'balloon_type' and 'balloon_filling')
# like above.
#
# ------------------------------------
```

3.3.2 Timestamps of change events

According to the number of changes of a sounding parameter or setting over time, each place holder can be redefined as often as required, e.g. balloon_type and balloon_filling:

```
# set default balloon type and filling
balloon_type;;TA350
balloon_filling;;1100

# change to TX600 on 2013-01-04 12UTC
balloon_type;2013-01-04T12;TX600
balloon_filling;2013-01-04T12;1250

# change to TA350 on 2013-04-24 00UTC
balloon_type;2013-04-24T00;TA350
balloon_filling;2013-04-24T12;1100

# change to TX600 on 2013-11-15 00UTC
balloon_type;2013-11-15T00;TX600
balloon_filling;2013-11-15T00;1250

# change to TX800 on 2013-11-30 12UTC
balloon_type;2013-11-30T12;TX800
balloon_filling;2013-11-30T12;1600
```

This example shows that the balloon type is TA350 before 2013-01-04T12, then changing to TX600, back to TA350, TX600, and finally to TX800 on 2013-11-30T12.

The timestamp should be given in ISO 8601 format. It is possible to give abbreviated versions. If so, the time stamp is automatically completed with default values according to the scheme "xxxx-01-01T00:00:00.000Z".

```
2013-11-30T12:00:00.000Z
2013-11-30T12:00
2013-11-30T12
```

The specification of a timestamp is optional. If no timestamp is given, the defined value applies to the entire period (from the beginning).

```
balloon_type;;TA350
balloon_filling;;1100
```

3.3.3 Definitions of data file types

The definition of relevant data file types is a bit more complex, see e.g. the following section of a change file:

```
# -------------------------------------
# Definitions of data file types
# -------------------------------------
# e.g. LITAuto1_20110101_113008.dc3db
#   1) Tenerife_20080101_111504.dc3db
#   2) Tenerife_20080101_111504_Add.dc3db
file_types;;DC3DB|DC3DB-ADD
file_codes;;F01|F02
file_masks;;*_??????.dc3db|*_Add.dc3db
file_date_formats;;yyyyMMdd'_'Hhmmss|yyyyMMdd'_'Hhmmss
file_date_starts;;9|9
file_required;;true|false
file_sv_use;;false|false
file_ignore_pe;;false|false
# -------------------------------------
```

Note: Please contact the GRUAN Lead Centre (gruan.lc@dwd.de) for assistance in creating or adapting data file type sections for the first time.

It is mandatory to define at least one file type, multiple files at once is possible. The definitions should be separated by a vertical line "|".

The following keys are required to define file types in a change-file:

- *file_types* – a unique key of a known file type (see section 3.3.4)
- *file_codes* – a unique key, e.g. F01, F02, ...
- *file_masks* – a file mask to identify files, e.g. "*.dc3db"
- *file_date_formats* – a date format to extract date and time of launch from the file name
- *file_date_starts* – an index indicating the beginning of the date string
- *file_required* – boolean value (true/false) if a file is required or not
- *file_sv_use* – boolean value (true/false) to activate extracting additional special values from files (see section 3.3.5)
- *file_ignore_pe* – boolean value (true/false): Should parsing errors be ignored for this file type?

Following information will be extracted from each file:

- *launch_date* – [required] full time stamp of launch
- *sonde_sn* – [required] serial number of main sonde
- *list of special values* – [optional] a pre-defined list of additional special values, e.g. balloon type, sonde type, frequency, operator, ... (see section 3.3.5).

In the actual *gtRsl* version, changes of file types over time is not implemented yet. That is, only one file type section can be defined in a change-file, and timestamps for "file_" keys will be ignored when given. As a work-around, a new change-list file must be created in case file type changes are to be defined.

3.3.4 Processable sounding file types

Sounding files are in most cases raw data files with a specific format. To account for this, corresponding specific reader modules for a number of file types were developed and integrated to enable the *gtRsl* tool to process the different file types equivalently. The following file types can be processed by the actual version of gtRsl:

- *MWX* – Vaisala radiosonde archive file of the MW41 system
- *DC3DB* – Vaisala DigiCoraIII radiosonde archive file
- *DC3DB-ADD* – very similar to *DC3DB*
- *IGNC-RAW* – French IPSL GRUAN NetCDF file of radiosonde raw data (L1A)
- *GSFZ* – Graw Sounding File (packed)
- *ARM-PTU* – ARM radiosonde PTU file
- *ARM-RAW* – ARM radiosonde RAW file
- *ARM-PARSED* – ARM radiosonde parsed file.

Note: In case of the need for additional file types, please contact the author or the GRUAN Lead Centre (gruan.lc@dwd.de). Other file types can be integrated on request if they are used operationally on at least one GRUAN site.

3.3.5 Special values in change files

Often many additional or sonde-specific meta-data are stored in the manufacturer data files. The usage of the tool's "specific values" functionality gives access to this information.

The following keys can be defined in a change file if "specific values" are to be used:

- *file_sv_<file_code>_names* – list of names of "specific values" to be extracted from the file
- *file_sv_<file_code>_<sv_name>_target* – target name (place holder) for use in the template file
- *file_sv_<file_code>_<sv_name>_default* – default value to be assigned in case the value of a "specific" variable cannot be extracted from the file.

In the change file it should look like this:

```
# define files
file_types;;MWX|DC3DB|TXT
file_codes;;F01|F02|F03
...
# define specific values
file_sv_use;;true|true|false
file_sv_F01_names;;SondeFamily,SpecialSensor
```

```
file_sv_F01_SondeFamily_target;;sonde_family
file_sv_F01_SondeFamily_default;;0-UNKNOWN-SONDE
file_sv_F01_SpecialSensor_target;;with_ozone
file_sv_F01_SpecialSensor_default;;0
file_sv_F02_names;;comment
file_sv_F02_comment_target;;operator
file_sv_F02_comment_default;;TS1-DUMMY
```

It is possible to define special values for each data file type. To do so a specific file "<special FileType name>SpecificValues.properties" can be stored in the folder "config" (e.g. "MwxFile-SpecificValues.properties"). Such properties files must start with the following lines:

```
# Specific values of MWX meta-data
type=java.util.Map
converter=org.osjava.sj.loader.convert.MapConverter

# Here you can define all specific values (additional meta-data).
```

The definition of a "specific value" can be done with one or two lines. A line defining the mapping (or renaming) from a meta-data key to the name of the specific value is essential. In addition an optional evaluation formula can be applied to change, filter, or calculate the final result:

- *sv.<Name>.def* – Main definition line to map (rename) a meta-data key to a specific value name,
- *sv.<Name>.jeval* – Optional line to define an evaluation formula (see appendix D).

Following example shows the definition of two "specific values" (one with and one without usage of an evaluation formula):

```
# ---------------------------------------
# Example definitions
# ---------------------------------------

# SondeFamily
# * RS92, RS41
sv.SondeFamily.def=Radiosondes.SondeFamily
# sv.SondeFamily.jeval=

# SpecialSensor
# * Special sensor type: 0 = None, 1 = Generic, 2 = Ozone
sv.SpecialSensor.def=Soundings.SpecialSensorType
sv.SpecialSensorType.jeval=S:ifs(\#{value} == 0, 'None', ifs(\#{value} ==
    1, 'Generic', ifs(\#{value} == 2, 'Ozone', 'Unknown')))
```

Note: Please contact the author or the GRUAN Lead Centre (gruan.lc@dwd.de) for assistance related to the "specific value" functionality.

3.4 Scheduling options

There is a number of options to manage scheduling purposes. In the GRUAN data flow, a *scheduled date* is required for each measurement, that means a planned launch date and time in case of a radiosounding. Especially sounding sites that are operated by weather services normally launch on a regular basis, e.g. at daily times 00 UTC or 12 UTC. In practice the actual time of a site-specific launch may vary within one hour or so from day to day around the scheduled nominal times.

In a first step, *gtRsl* creates a time schedule, and in a second step the files will be assigned to these times.

Using the following group of options a range of dates can be defined. Start and end date can be defined in an automatic way (*-u, --auto-date*) or manually (*-s, --start-date* and *-e, --end-date*). The date *<DATE>* format should follow the ISO standard: yyyy-MM-dd (yyyy – year, e.g. 2017, MM – month, e.g. 12, dd – day of month, e.g. 04). The scheduling starts by default with time "00:00:00.000Z". If a different start time is to be used, it can be given with the option *-m,--start-time*. The time *<TIME>* format should follow ISO as well: HH:mm:ss.SSS (HH – hour, e.g. 23, mm – minute, e.g. 59, ss – second, e.g. 59, SSS – millisecond, e.g. 000). It is not required to give a full time, e.g. "00" or "00:00" is allowed.

The schedule requires the definition of a measurement interval. The default is "PT12H" (what means 12 hours), if nothing else is specified using *-p, --period*. The period *<PERIOD>* format should follow ISO: e.g. "PT12H" (12 hours) or "PT1H30M" (1 hour and 30 minutes).

There are two options (*-a, --after-slot* and *-b, --before-slot*) to handle possible deviations between scheduled and actual launch date and time. With these options a time interval can be defined so that the concerned launches are assigned to the same nominal schedule time, if they are within that interval covering the schedule time minus "before slot" to schedule time plus "after slot". The default settings are "PT2H" for option *-b* and "PT1H" for option *-a*.

Two or more launches can be assigned to the same schedule date, e.g. in case of a relaunch after a failed or aborted sounding. With the option *-w, --launch-gap* a time period can be defined (default is "PT10M") which separates between these launches.

In addition to the above scheduling options, a pre-defined time difference between ground checks and launch can be defined by using the option *-k, --check-gap* (default is "PT20M"). This creates the variables *check1_date* to *check5_date* and is useful if one or more ground checks are defined in the template. For each ground check performed, a realistic time (relatively short before the launch) must be specified in the final GRUAN meta data file. Since this is often not known, these symbolic times are generated, which can be used in template files to store a symbolic date and time for ground checks 1 to 5.

3.5 Blocking of files

It may occasionally happen that files available in source directories are not "compatible" with the pre-defined setup (see change file), e.g. because of:

- change of the radiosonde type, e.g. from RS92 to RS41,
- an ozone sonde is attached to the radiosonde, but a "routine" sounding is expected.

In such cases it is necessary to block these files and to remove them from the list of selected data files. There are two options for blocking:

- *--block-formula* – Define an evaluation formula or condition (see appendix D). If the evaluation of this expression for a specific file will result in a “1” (true), the file will be blocked.
- *-x, --move-blocked-files* – Move blocked source files to the quarantine directory (see Section 3.7).

3.6 Detect double files

Sometimes two or more files with different names but identical or very similar content may be found:

- same CRC, which indicates identical file content
- same file type, sonde serial number, and launch date and time; that means files from the same sounding but with small deviations in content (e.g. a reprocessed or simulated sounding).

In both cases it is necessary to remove the duplicate or multiple files from the list of data files before processing.

Following options specify how to handle duplicates:

- *-q, --move-double* – Move all detected double source files to the quarantine directory (see Section 3.7).

3.7 Create, copy, delete, and move files

The *gtRsl* tool is capable to perform some file operations using options to specify these in detail. Before doing these file operations, the relevant directories must be defined:

- *[FILE]* – Define a user list of source files or directories. If not specified, the working directory is used.
- *-o,--out-dir* – Define a user output directory. The result files will be created there, and the copies of the used/related source data files will be saved there. A sub-folder structure will be created there automatically:
  - *<site>* – GRUAN site name, e.g. “Lindenberg”, “Cabauw”,
  - *<type of measurement system>* – GRUAN name of measurement system, e.g. “Radiosonde”, “Lidar”
- *--quarantine-dir* – In case of blocked or double files (Sections 3.5 and 3.6), this directory is used to store such sorted out files.

The following further options can be used to manage files:

- *-c, --copy-all* – Copy the complete set of source files to the output directory. Without using this option, only GMD files will be created there.

- *-d, --delete-all* – Delete all related source files. If this option is used, option *-c* is automatically applied. That means, deletion without copying is not possible as a safety precaution.
- *-q, --move-double* – Move the detected double source files to the quarantine directory.
- *-x, --move-blocked-files* – Move the blocked source files to the quarantine directory.

3.8 FTP uploading options

Since version 0.4.0 the functionality to upload files directly to the GRUAN Lead Centre using FTP is built in.

- *-f, --ftp-conn <NAME>* – name of the FTP connection for uploading files to GRUAN. If NAME is not specified, then 'GruanIncomingRawFTP' is used as default.

With the option *-f*, the files in the user output directory (*--out-dir*) will be uploaded to GRUAN. This is done after data file processing and creation and moving/storing the associated GMD files to the output directory. After successful upload of the files to GRUAN they are deleted in the local output directory.

The tool searches for a file named "<NAME>.properties" in the directory "config" (e.g. "GruanIncomingRawFTP.properties"). Two default FTP connections are included in the tool: GruanIncomingRawFTP and GruanIncomingTestRawFTP. For such a configuration file, the content of "GruanIncomingTestRawFTP.properties" is given as an example:

```
type=org.gruan.config.FtpConfig
converter=org.osjava.sj.loader.convert.BeanConverter

# definitions of FTP connection
protocol=ftp
provider=Hetzner
host=filetransfer.gruan.info
user=site_all
passwd=(not shown here)
dir=/test/raw
```

FTP configuration files must always start with exactly the indicated two lines (type, converter).

The following specifications are then expected for the definition of the access:

- *protocol* – data transfer protocol to use; only 'ftp' is currently implemented
- *provider* – text identifier of the provider; optional
- *host* – host server name (or IP address)
- *user* – user name to login
- *passwd* – user password to login
- *dir* – path to the base directory where the data should be stored/uploaded.

3.9 General options

There are additional, more general options:

- *-l, --logging* – Sets the logging level to another than the default (INFO), i.e. SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST.
- *-n, --no-abort* – Increase the error tolerance of the tool.

Especially in the case of errors it can be useful to make the output of the tool more verbose during processing. The logging level "INFO" is selected as the default. Already with the levels "CONFIG" or "FINE" the output can be extended considerably.

gtRsl is often used in automated data flows. Errors can occur during processing for various reasons, e.g. if an input file is corrupt and cannot be read correctly. Normally, the tool aborts the processing if any "real" error occurs. However, this results in nothing being processed. With the option *-n* this very strict behaviour can be made a bit more tolerant, so that errors that are not considered critical for the entire processing will not cause the processing to be aborted.

4 Examples

Note: To increase the readability of the following examples, the commands have been distributed over several lines. Normally, however, there is exactly one line. However, depending on the operating system and the selected shell, there are ways to spread the commands over several lines. If this is required, please consult the relevant documentation for the shell you are using.

Note: Depending on the operating system, the directory levels within paths are separated by different characters, e.g. '/' for Linux and '\' for Windows. Please follow the system specific notation.

4.1 Example 1 – use of a time range

As an exercise you may run the processing for a defined time range (January 2012 – exactly one month) using the given template ("templates/test-template.ftl"), change list ("changes/text-change.txt"), and data files in the directory "~/test-data". Some default values are used, because they are not exactly specified, e.g. the period PT12H (12 hours), "before slot" PT2H (2 hours) and "after slot" PT1H (1 hour). See Section 3.4 to find more information about the scheduling options.

```
gtRsl -r -s "2012-01-01" -e "2012-02-01" -t "templates/test-template.ftl"
      -g "changes/test-changes.txt" "~/test-data"
```

4.2 Example 2 – use of start date only

Same as example 1 except using a time range from start date (2012-07-01) to now.

```
gtRsl -r -s "2012-07-01" -t "templates/test-template.ftl"
      -g "changes/test-changes.txt" "~/test-data"
```

4.3 Example 3 – use of automatic time range detection

Data files in the directory "new_files" will be parsed, i.e. the time range will automatically be detected from files. The period between two launches is 3 hours (starting at 00:00:00Z) with a minimum of 10 minutes for an afterstart. The relevant files will be copied into the folder "upload" and deleted in the source folder. The amount of status information given by the tool during processing is enhanced by using the log-level "CONFIG".

```
gtRsl -r -c -d -n -u -m "00:00:00" -p "PT3H" -w "PT10M" -l "CONFIG"
      -t "RS_ROUTINE_v4.ftl"
      -g "changeFile_LAU-RS-02_2016.txt"
      -o "upload"
      "new_files"
```

4.4 Example 4 – use of data file blocking

A more complex example of a sounding system with an "unknown date" of sonde type change from RS92 to RS41 is shown. Here data files for RS92 should be blocked and moved to the directory "quarantine". The files in the directory "input_data" will be parsed and the time range is automatically detected within 30 min intervals starting at 00:00:00Z.

```
gtRsl -r -n -l "INFO"
      -u -m "00:00:00" -p "PT30M" -w "PT10M" -b "PT15M" -a "PT15M"
      -t "templates/RS_ROUTINE_temp6.ftl"
      -g "changes/changeFile_SGP-S01_SGP-RS-01_2017.txt"
      -q --quarantine-dir="quarantine"
      -o "upload"
      --block-formula="S:equals(#{sonde_family}, 'RS92')"
      "input_data"
```

Note: This example uses the "specific value" functionality, which is briefly described in Section 3.3.5. *Please contact the GRUAN Lead Centre (gruan.lc@dwd.de) for assistance.*

4.5 Example 5 – use of FTP upload

Same as example 3 except using in addition the pre-defined FTP connection "GruanIncomingRawFTP" to upload data directly to the GRUAN LC (see Section 3.8).

```
gtRsl -r -c -d -n -u -m "00:00:00" -p "PT3H" -w "PT10M" -l "CONFIG"
      -t "RS_ROUTINE_v4.ftl"
      -g "changeFile_LAU-RS-02_2016.txt"
```

```
       -o "upload"
       -f "GruanIncomingRawFTP"
       "new_files"
```

Appendix

A List of options

Full list of options for processing customisation. If main option -r is choose options -t, and -g are mandatory.

```
List of options:
 -a,--after-slot <PERIOD>       Period defines second part of time slot
                                after schedule date.
 -b,--before-slot <PERIOD>      Period defines first part of time slot
                                before schedule date.
    --block-formula <FORMULA>   A formula used to block files, if
                                the evaluation results in ‘true’.
 -c,--copy-all                  Copy referenced files to the specified
                                output directory (see --out-dir).
 -d,--delete-all                Delete referenced files after copying
                                to the specified output directory (see
                                --copy-all and --out-dir). Option
                                --copy-all is automatically set if not
                                done before.
 -e,--end-date <DATETIME>       End date defines last schedule date (not
                                included!) for creation of GMD files.
 -f,--ftp-conn <NAME>           The name of FTP connection to use for
                                uploading files to GRUAN. If NAME is
                                not given, then ‘GruanIncomingRawFTP’ is
                                used as default.
 -g,--change-list <FILE>        List of changes for creation of GMD files.
 -h,--help                      Prints the help information and exit.
 -k,--check-gap <PERIOD>        Period defines the gap between a check and
                                the launch or between two checks.
 -l,--logging <LOG-LEVEL>       Sets the logging level; default is INFO. The
                                setting options are: SEVERE, WARNING, INFO,
                                CONFIG, FINE, FINER, FINEST.
 -m,--start-time <TIME>         Defines the time of first schedule date
                                (included) for creation of GMD files.
 -n,--no-abort                  Continues the processing
                                after an error occurred?
    --no-create-if-corrupt      Do not create GMD files in case of missing
                                required data files.
 -o,--out-dir <DIR>             Output directory for created and
                                exported files. If not given the working
                                directory is used as default.
 -p,--period <PERIOD>           Period defining the time interval between
                                two schedule dates for creation of GMD files.
 -q,--move-double               Detect (CRC or SN) and move double files to
                                a quarantine folder.
    --quarantine-dir <DIR>      Quarantine directory for duplicate or
                                blocked files. If not specified the option
                                --move-double only produces messages and
                                the option --move-blocked-files has no effect.
 -r,--run                       Runs an automatic creation of GMD files.
 -s,--start-date <DATETIME>     Start date defines first schedule date
```

```
                               (included) for creation of GMD files.
-t,--template <FILE>           Template for creation of GMD files.
   --tmp-dir <DIR>             Directory for temporary files. If not specified,
                               the system temporary directory is used.
-u,--auto-date                 Automatic definition of start and end date.
-v,--version                   Prints the version information and exit.
-w,--launch-gap <PERIOD>       Period defines the minimum time gap between
                               two launches at one time step (e.g. afterstart).
-x,--move-blocked-files        Move blocked files to the quarantine folder.
```

B Example template files

The *gtRsl* is a helper tool for semi-automatic use. It can help to create necessary GMD files for each measurement event (e.g. radiosonde launch). As base a template file is required (see Section 3.2).

B.1 Template file for a routine launch with auto-launcher

The template is created based on the following sounding equipment and specific properties:

- Data processing system (DPS) with related data files
- Ground check tool 1
- Balloon with specific information about gas, filling volume and included parachute
- Parachute
- Unwinder
- Rig (in case of routine launch always *SOLO*)
- Radiosonde with link to DPS and specific check (performed with the check tool 1).

This example template can be used for launches performed with an auto-launcher system. The template variables *${variable_name}* have to be pre-filled using the "change file".

```
<#setting number_format="computer">
<?xml version="1.0" encoding="UTF-8"?>
<gmdFile xmlns="http://www.gruan.org/GruanMetaData/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gruan.org/GruanMetaData/1.0
  GruanMetaData-1.0.xsd">
  <#escape x as x?xml>
  <version>1.2</version>
  <creatorTool>${creation_tool}</creatorTool>
  <timestamp>${creation_timestamp}</timestamp>
  <purpose>RsLaunch</purpose>
  <RsLaunch measurementNumber="${launch_number}"
    measurementSystem="${measuring_system}" operator="${operator}"
    setup="${measuring}" standardDate="${launch_date}" station="${site_name}"
    version="${launch_version}">
    <comment>${creation_comment}</comment>
    <rsPart code="${dps_type}" group="GROUND" permanentCode="${dps_pcode}"
      specificCode="${dps_scode}"
      <#if !required_data>corrupt="true"</#if> sn="${dps_sn}" type="DPS">
      <#list data_files as file>
      <file crc="${file.fileChecksum}" fileType="${file.fileType}"
        filename="${file.fileName}"
        localPath="${file.filePath}"
        localType="LOCAL" size="${file.fileLength}" />
      </#list>
    </rsPart>
    <rsPart code="${checktool1_type}" group="GROUND"
      permanentCode="${checktool1_pcode}" specificCode="${checktool1_scode}"
      sn="${checktool1_sn}" type="CheckTool"/>
    <rsPart code="${balloon_type}" group="LAUNCH" sn="none" type="Balloon">
      <property name="Gas" relatedTo="event" type="ENUMERATION"
        value="${balloon_gas}"/>
      <property name="FillingVolumn" relatedTo="event" type="FLOAT"
        value="${balloon_filling}"/>
      <property name="WithInsideParachute" relatedTo="event" type="BOOLEAN"
        value="${parachute_inside}"/>
    </rsPart>
    <rsPart code="${parachute_type}" group="LAUNCH" sn="none" type="Parachute"/>
    <rsPart code="${unwinder_type}" group="LAUNCH" sn="none" type="Unwinder"/>
    <rsPart code="${rig_type}" group="LAUNCH" sn="none" type="Rig"/>
    <rsPart code="${sonde_type}" group="LAUNCH" sn="${sonde_sn}" type="Sonde">
      <check code="${check1}" date="${check1_date}">
        <tool code="${checktool1_type}"  permanentCode="${checktool1_pcode}"
          specificCode="${checktool1_scode}" sn="${checktool1_sn}"/>
      </check>
      <dataFlow code="${dps_type}" permanentCode="${dps_pcode}"
        specificCode="${dps_scode}" sn="${dps_sn}"/>
    </rsPart>
  </RsLaunch>
  </#escape>
</gmdFile>
```

B.2 Template file for a manual ozone sounding

The following equipment and specific properties are considered in this template:

- Data processing system (DPS) with related data files
- Ground check tools 1, 2, and 3
- Balloon with specific information about gas type, filling volume and integrated parachute
- Parachute
- Unwinder
- Rig (in case of ozone launch often *BLOCK*)
- Radiosonde with link to DPS and two specific checks (performed with the check tools 1 and 2)
- Ozone sonde with link to DPS and specific check (preparation performed with the check tool 3)

This example template can be used for typical manual ozone launches. The template variables *${variable_name}* have to be pre-filled using the “change file”.

```
<#setting number_format="computer">
<?xml version="1.0" encoding="UTF-8"?>
<gmdFile xmlns="http://www.gruan.org/GruanMetaData/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gruan.org/GruanMetaData/1.0
  GruanMetaData-1.0.xsd">
  <#escape x as x?xml>
  <version>1.2</version>
  <creatorTool>${creation_tool}</creatorTool>
  <timestamp>${creation_timestamp}</timestamp>
  <purpose>RsLaunch</purpose>
  <RsLaunch measurementNumber="${launch_number}"
    measurementSystem="${measuring_system}" operator="${operator}"
    setup="${measuring}" standardDate="${launch_date}" station="${site_name}"
    version="${launch_version}">
    <comment>${creation_comment}</comment>
    <rsPart code="${dps_type}" group="GROUND" permanentCode="${dps_pcode}"
      sn="${dps_sn}" <#if !required_data>corrupt="true"</#if>
      specificCode="${dps_scode}" type="DPS">
      <#list data_files as file>
      <file crc="${file.fileChecksum}" fileType="${file.fileType}"
        filename="${file.fileName}"
        localPath="${file.filePath}"
        localType="LOCAL" size="${file.fileLength}" />
      </#list>
    </rsPart>
    <rsPart code="${checktool1_type}" group="GROUND"
      permanentCode="${checktool1_pcode}" specificCode="${checktool1_scode}"
      sn="${checktool1_sn}" type="CheckTool"/>
    <rsPart code="${checktool2_type}" group="GROUND"
      permanentCode="${checktool2_pcode}" specificCode="${checktool2_scode}"
      sn="${checktool2_sn}" type="CheckTool"/>
    <rsPart code="${checktool3_type}" group="GROUND"
      permanentCode="${checktool3_pcode}" specificCode="${checktool3_scode}"
      sn="${checktool3_sn}" type="CheckTool"/>
    <rsPart code="${balloon_type}" group="LAUNCH" sn="none" type="Balloon">
      <property name="Gas" relatedTo="event" type="ENUMERATION"
        value="${balloon_gas}"/>
      <property name="FillingWeight" relatedTo="event" type="FLOAT"
        value="${balloon_filling}"/>
      <property name="Pretreatment" relatedTo="event" type="FLOAT"
        value="${balloon_pretreat}"/>
    </rsPart>
    <rsPart code="${parachute_type}" group="LAUNCH" sn="none" type="Parachute"/>
    <rsPart code="${unwinder_type}" group="LAUNCH" sn="none" type="Unwinder"/>
    <rsPart code="${rig_type}" group="LAUNCH" sn="none" type="Rig"/>
    <rsPart code="${sonde_type}" group="LAUNCH" sn="${sonde_sn}" type="Sonde">
      <check code="${check1}" date="${check1_date}">
        <tool code="${checktool1_type}"  permanentCode="${checktool1_pcode}"
          specificCode="${checktool1_scode}" sn="${checktool1_sn}"/>
      </check>
      <check code="${check2}" date="${check2_date}">
        <tool code="${checktool2_type}"  permanentCode="${checktool2_pcode}"
          specificCode="${checktool2_scode}" sn="${checktool2_sn}"/>
        <property name="Pot100.RefRelativeHumidity" type="FLOAT" value="100.0"/>
      </check>
      <dataFlow code="${dps_type}" permanentCode="${dps_pcode}" sn="${dps_sn}"/>
    </rsPart>
    <rsPart code="${sonde2_type}" group="LAUNCH" sn="${sonde2_sn}" type="Sonde">
      <check code="${check3}" date="${check3_date}">
        <tool code="${checktool3_type}"  permanentCode="${checktool3_pcode}"
          specificCode="${checktool3_scode}" sn="${checktool3_sn}"/>
      </check>
      <dataFlow code="${dps_type}" permanentCode="${dps_pcode}" sn="${dps_sn}"/>
    </rsPart>
  </RsLaunch>
  </#escape>
</gmdFile>
```

C Example change-list files

The *gtRsl* is a helper tool for semi-automatic use. It can help to create necessary GMD files for each measurement event (e.g. radiosonde launch). A base configuration and all configuration changes over time should be defined in a change-list file (see section 3.3).

C.1 Example change-list file from a site with few changes over time

The following example of a change-list file contains a full base configuration of a routine sounding measurement program with only few configuration changes over time.

```
# -------------------------------
# Change file for MCM (McMurdo)
# -------------------------------
# Syntax: key;start date;value
# -------------------------------
# CASE: MWX file with RS41 in 2019 (since 2019-01-01, 00:00)
# -------------------------------
#
# ---------------
# Main properties
# ---------------
#
site;;MCM
site_name;;McMurdo
measuring_system;;MCM-RS-01
measuring;2019-01-01;ROUTINE
launch_number;;1
launch_version;;1
#
# ------------------------------
# Definitions of data file types
# ------------------------------
# >> NZCM_20190125_103934.mwx
```

```
file_types;;MWX
file_codes;;F01
file_masks;;NZCM_????????_??????.mwx
file_date_formats;;yyyyMMdd'_'HHmmss
file_date_starts;;5
file_required;;true
file_ignore_pe;;false
file_sv_use;;true
file_sv_F01_names;;SondeFamily
file_sv_F01_SondeFamily_target;;sonde_family
file_sv_F01_SondeFamily_default;;0-UNKNOWN-SONDE
#
# ---------------------
# Basis types and codes
# ---------------------
#
operator;;DUMMY
# balloon
balloon_type;;TA350
balloon_gas;;Helium
balloon_filling;;1300
# setup components
unwinder_type;;UW-V55-4
parachute_type;;0-NONE
rig_type;;SOLO
sonde_type;;RS41-SGP
# data processing system
dps_type;;DC-MW41
dps_pcode;;MW41-SYSTEMS
dps_scode;;MW41-1_001
dps_sn;;mcm-mw41-1
# ground check with used tool
check;;GC-RI41
checktool_type;;DC-RI41
checktool_pcode;;RI41-UNITS
checktool_scode;;RI41-1_001
checktool_sn;;mcm-ri41-1
#
# ----------------
# Specific changes
# ----------------
#
# change of balloon (from TA350 to TA500) at 2019-05-01T12
balloon_type;2019-05-01T12;TA500
balloon_filling;2019-05-01T12;1800
#
# change of balloon back (from TA500 to TA350) at 2019-10-01T00
balloon_type;2019-05-01T12;TA350
balloon_filling;2019-05-01T12;1300
#
```

D Evaluation formulas

For different purposes it is useful to have the option to define and evaluate simple mathematical, string, or conditional expressions. Therefore the feature of "evaluation formulas" has been integrated.

D.1 General usage

Simple mathematical and string operations can be applied to the given variables, or e.g. conditional expressions can be evaluated. Special values can be passed to the expression *#{value}*. Instead of "value", other named variables can be used in certain situations, e.g. *#{sonde_family}*.

The contents of variables can be interpreted as "string" (text value) or as "number" (floating point value). It is necessary to choose between these two. This is done by starting each expression or formula with either *s:* (string) or *n:* (number), see the following example:

```
# Round a number
N:rounding(#{value}, 4)
# Get one part of a string without any whitespace around
S:trim(part(#{value}, '_', 0))
# Give an empty string in case of 'None'
S:ifs(#{value} == 'None', '', #{value})
```

gtRsl includes the following contexts where "evaluation formulas" can be used:

- The option *--block-formula* can handle such formulas (see section 3.5).
- The definition of special values of specific file types (e.g. MWX) can handle such formulas in the definition line *sv.<Name>.jeval* of "SpecificValues.properties" files (see section 3.3.5).

Note: In case of ".properties" files it is necessary to mask the special character "#", because it would be detected as marker to start a comment. Please use "\#" instead.

Same example formulas in case of ".properties" files:

```
# Round a number
N:rounding(\#{value}, 4)
# Get one part of a string without any whitespace around
S:trim(part(\#{value}, '_', 0))
# Give an empty string in case of 'None'
S:ifs(\#{value} == 'None', '', \#{value})
```

Note: When using the option *--block-formula*, don't use white-space characters because they are treated as special characters at the command line.

Same example formulas in case of command line argument:

```
# Round a number
N:rounding(#{value},4)
```

```
# Get one part of a string without any whitespace around
S:trim(part(#{value},'_',0))
# Give an empty string in case of 'None'
S:ifs(#{value}=='None','',#{value})
```

Note: A parsing error is reported if any error occurs when applying formulas. Unfortunately, it may be difficult to find and solve the error. Please contact the author or the GRUAN Lead Centre (gruan.lc@dwd.de) in such cases.

D.2 Operators

The evaluation formulas – like normal formulas – can be nested at any depth. The following operators can be used for this purpose.

- *&&* – (*number*) The boolean "and" operator.
- *!* – (*number*) The boolean "not" operator.
- *||* – (*number*) The boolean "or" operator.
- */* – (*number*) The division operator.
- *%* – (*number*) The modulus operator.
- *** – (*number*) The multiplication operator.
- *-* – (*number*) The subtraction operator.
- *+* – (*number or string*) The addition operator.
- *==* – (*number or string*) The equal operator.
- *!=* – (*number or string*) The not equal operator.
- *>* – (*number or string*) The greater than operator.
- *>=* – (*number or string*) The greater than or equal operator.
- *<* – (*number or string*) The less than operator.
- *<=* – (*number or string*) The less than or equal operator.
- *(* – (*number or string*) The opening bracket to start a group (a part) of formula.
- *)* – (*number or string*) The closing bracket to finish a group (a part) of formula.

D.3 String functions

A large number of functions is defined that require "string" values as main input and/or output variables. Each function must be called with a fixed number of arguments.

- *charAt(<string> , <int>)* – Returns the character at the specified index in the source string. Arg1 is the source string and arg2 (int) is the specified index.
- *compareTo(<string> , <string>)* – Compares two strings lexicographically. Arg1 is first string and arg2 second string.
- *compareToIgnoreCase(<string> , <string>)* – Compares two strings lexicographically, ignoring case considerations. Arg1 is first string and arg2 second string.

- *concat(<string> , <string>)* – Concatenates the second string to the end of the first. Arg1 is first string and arg2 second string.
- *endsWith(<string> , <string>)* – Tests if the string ends with a specified suffix. Arg1 is the source string and arg2 is the specified suffix.
- *equals(<string> , <string>)* – Tests one string equals another. Arg1 is first string and arg2 second string.
- *equalsIgnoreCase(<string> , <string>)* – Tests one string equals another, but ignores case. Arg1 is first string and arg2 second string.
- *eval(<string>)* – The function returns the result of a compatible expression. Arg1 is the compatible expression as string.
- *indexOf(<string>, <string>, <int>)* – Returns the index within the source string of the first occurrence of the substring, starting at the specified index. Arg1 is the source string, arg2 is the substring and arg3 (int) is the specified index.
- *lastIndexOf(<string>, <string>, <int>)* – Returns the index within the source string of the last occurrence of the substring, starting at the specified index. Arg1 is the source string, arg2 is the substring and arg3 (int) is the specified index.
- *length(<string>)* – Returns the length of the source string. Arg1 is the source string.
- *replace(<string> , <char> , <char>)* – Returns a new string with all of the occurrences of the old character in the source string replaced with the new character. Arg1 is the source string, arg2 is the old character and arg3 is the new character.
- *startsWith(<string>, <string>, <int>)* – Tests if the string starts with a specified prefix beginning at a specified index. Arg1 is the source string, arg2 is the substring and arg3 (int) is the specified index.
- *substring(<string> , <int> , <int>)* – Returns a string that is a substring of the source string. Arg1 is the source string, arg2 (int) is the starting index (inclusive) and arg3 (int) is the ending index (exclusive).
- *toLowerCase(<string>)* – Returns the source string in lower case. Arg1 is the source string.
- *toUpperCase(<string>)* – Returns the source string in upper case. Arg1 is the source string.
- *trim(<string>)* – Returns the source string with white space removed from both ends. Arg1 is the source string.

D.4 Math functions

The defined mathematical functions are listed below. They require numeric values as input and/or output variables. Each function must be called with a fixed number of arguments.

- *abs(<double>)* – Returns the absolute value of a double value. Arg1 is the double value.
- *acos(<double>)* – Returns the arc cosine of an angle. Arg1 is the angle.
- *asin(<double>)* – Returns the arc sine of an angle. Arg1 is the angle.
- *atan(<double>)* – Returns the arc tangent of an angle. Arg1 is the angle.
- *atan2(<double> , <double>)* – Converts rectangular coordinates to polar. Returns

the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, θ). This method computes the phase theta by computing an arc tangent of y/x in the range of $-\pi$ to π. Arg1 is y – the ordinate coordinate and arg2 is x – the abscissa coordinate.

- *ceil(<double>)* – Returns the ceiling value of a double value. It is the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer. Arg1 is the double value.
- *cos(<double>)* – The function returns the trigonometric cosine of an angle. Arg1 is the angle.
- *exp(<double>)* – Returns the exponential number e (i.e., 2.718...) raised to the power of a double value. Arg1 is the double value.
- *floor(<double>)* – Returns the floor value of a double value. Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer. Arg1 is the double value.
- *IEEEremainder(<double> , <double>)* – Returns the remainder operation on two arguments as prescribed by the IEEE 754 standard. The remainder value is mathematically equal to $f1 - f2 \times n$, where n is the mathematical integer closest to the exact mathematical value of the quotient f_1/f_2, and if two mathematical integers are equally close to $f1/f2$, then n is the integer that is even. If the remainder is zero, its sign is the same as the sign of the first argument. Arg1 is f_1 – the dividend and arg2 is f_2 the divisor.
- *log(<double>)* – The function returns the natural logarithm (base e) of a double value. Arg1 is the double value.
- *max(<double> , <double>)* – Returns the greater of two double values. Arg1 is first double value and arg2 is second double value.
- *min(<double> , <double>)* – Returns the smaller of two double values. Arg1 is first double value and arg2 is second double value.
- *pow(<double> , <double>)* – Returns the value of the first argument (arg1) raised to the second power of the second argument (arg2).
- *random()* – The function returns a random double value greater than or equal to 0.0 and less than 1.0. This function has no arguments.
- *rint(<double>)* – Returns the double value that is closest in value to the argument (arg1) and is equal to a mathematical integer.
- *round(<double>)* – Returns the closet long to a double value. Arg1 is the double value.
- *sin(<double>)* – Returns the sine of an angle. Arg1 is the angle.
- *sqrt(<double>)* – Returns a square root of a double value. Arg1 is the double value.
- *tan(<double>)* – Returns trigonometric tangent of an angle. Arg1 is the angle.
- *toDegrees(<double>)* – Converts an angle measured in radians to the equivalent angle measured in degrees. Arg1 is the angle measured in radians.
- *toRadians(<double>)* – Converts an angle measured in degrees to the equivalent angle measured in radians. Arg1 is the angle measured in degrees.

D.5 Additional GRUAN functions

In addition to the default "math" and "string" functions, a number of further functions for GRUAN purposes is provided.

Note: It is possible to add more functions if needed. Please contact the author or the GRUAN Lead Centre (gruan.lc@dwd.de) to provide a proposal.

D.5.1 Scalar functions

The following scalar functions are of the same type as the default "math" and "string" functions and complement them.

- *contains(<string> , <string>)* – Checks if a source string contains a search string. Arg1 is the source string and arg2 is the search string.
- *emptyToNullFunction(<string>)* – Returns null in case of an empty string (with length 0). Arg1 is the source string.
- *ifm(<bool> , <double> , <double>)* – The function is a "IF" statement for numbers. Arg1 (bool) is the expression with a boolean result, e.g. 1 (true) or 0 (false). Arg2 is the result in case of true (1) and arg3 is the result in case of false (0).
- *ifs(<bool> , <string> , <string>)* – The function is a "IF" statement for strings. Arg1 (bool) is the expression with a boolean result, e.g. 1 (true) or 0 (false). Arg2 is the result in case of true (1) and arg3 is the result in case of false (0).
- *isNaN(<double>)* – Returns true (1) if the specified number is a Not-a-Number (NaN) value, false (0) otherwise. Arg1 is the value of type double.
- *match(<string> , <string>)* – Returns true (1) or false (0) if the source string matches the given regular expression. Arg1 is the source string and arg2 is the regular expression.
- *part(<string> , <string> , <int>)* – Splits the provided text into an array using the given separator and returns one part of this splitted string. Arg1 is the source string, arg2 is the separator char (or stirng) and arg3 is the index of the part (starting with 0).
- *partCount(<string> , <string>)* – Splits the provided text into an array using the given separator and returns the count of parts. Arg1 is the source string and arg2 is the separator character (or string).
- *posDistFunction(<double> , <double> , <double> , <double> [, <dbl> , <dbl>])* – Calculates the geodetic curve between two points on a specified reference ellipsoid (if 4 arguments are given), or calculates the three dimensional geodetic measurement between two positions measured in reference to a specified ellipsoid (if 6 arguments are given). The ellipsoid "WGS84" is used. Arg1 is the latitude of point A and arg2 is the longitude of point A. Arg3 is the latitude of point B and arg4 is the longitude of point B. Optional arguments arg4 and arg5 are the elevations of point A and B.
- *rounding(<double> , <int> , <string>)* – Returns a rounded value using the given fraction and rounding mode. Arg1 is the value of type double, arg2 is the fraction and arg3 is the rounding mode. Following modes are available: *UP*, *DOWN*, *CEILING*, *FLOOR*, *HALF_UP*, *HALF_DOWN*, *HALF_EVEN*.

- *signum(<double>)* – Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero. Arg1 is the value of type double.
- *substringBetween(<string> , <string> , <string>)* – Returns the string that is nested in between two strings. Only the first match is returned. Arg1 is the source string which contains the substring, arg2 is the opening string before the substring and arg3 is the closing string after the substring.
- *toNumber(<string>)* – Convert given string to number. Arg1 is the source string.
- *toString(<double>)* – Convert given number to string. Arg1 is the value of type double.

D.5.2 Array functions

Array functions are implemented especially for the calculation of results related to data series. Therefore, they can only be used in specific contexts. With the current version, the *gtRsl* does not provide a suitable context.

- *avgKernel(<string> , <string> , <int>)* – Calculates a running mean of a given variable (array) using a kernel function with a given size. Arg1 is the kernel type (currently only "test" is available), arg2 is the variable name, and arg3 (int) is the kernel size.
- *stat(<string> , <string> , <int>)* – Calculates a statistical function of a given variable (array) with a given length. Arg1 is the actual function name, arg2 is the variable name, and arg3 (int) is the length. The following statistical functions, applying each to the values in the entire actual array, are available:
  - *mean* – Arithmetic mean.
  - *geometricMean* – Geometric mean.
  - *max* – Maximum.
  - *min* – Minimum.
  - *product* – Product.
  - *sum* – Sum.
  - *sumLog* – Sum of the natural logarithm.
  - *sumSq* – Sum of the squares.
  - *variance* – Variance.
  - *stdDev* – Standard deviation.
  - *populationVariance* – Population variance.

E History of development of *gtRsl*

The *gtRsl* is constantly being developed. This chapter briefly describes the changes back to first functional version (v0.2).

E.1 History of Version 0.5.x

```
* 0.5.1_08  (2020-08-18) - change: improve MWX file definition for MW41 v2.15
* 0.5.1_07  (2020-08-05) - change: update file format IGNC-RAW to v1.2.1
                         - bug fixed: get correct StartDateTime in
                           'IpslGruanNcRawFileMdExtractor'
* 0.5.1_06  (2020-04-29) - change: update file format IGNC-RAW to v1.2.0
                         - optimised and restructured 'DataFile' classes which
                           allows easier adding of new config properties
* 0.5.1_05  (2020-02-28) - bug fixed: crash in case of several files and
                           several SN
* 0.5.1_04  (2019-10-16) - bug fixed: overwritten special value definitions
                           from config file
                         - change: update MWX file definition for MW41 v2.15
* 0.5.1_03  (2019-06-28) - change file format name ('Sirta...' to 'Ipsl...')
* 0.5.1_02  (2019-05-06) - bug fixed: crash during check of wrong date
                           (05/06/2017 >> 06/05/2017)
* 0.5.1_01  (2019-05-06) - optimisation of GSFZ (real meta-data only)
* 0.5.1     (2019-05-06) - add handling of GSFZ
* 0.5.0_01  (2019-04-15) - remove '*SpecialValues.properties' files
* 0.5.0     (2019-04-02) - change handling of special values
```

E.2 History of Version 0.4.x

```
* 0.4.4_01  (2019-03-28) - change name of file type from SGNC-RAW to IGNC-RAW
* 0.4.4     (2019-03-19) - add handling of SGNC-RAW
* 0.4.3_03  (2018-10-23) - change ftp connections GruanIncomingRawFTP and
                           GruanIncomingTestRawFTP to use gruan.info (at
                           Hetzner)
                         - change: update MWX file definition for MW41 v2.11
* 0.4.3_02  (2018-10-23) - new compiled only
* 0.4.3_01  (2018-02-09) - first test to improve temporary file deletion of
                           MWX files
                           # add umount MWX file as archive
* 0.4.3     (2018-02-08) - bug fixed with use of tmp-dir
* 0.4.2     (2018-02-08) - add option '--tmp-dir'
                           # is used by file reader classes MwxFile and
                             Dc3DbFile
* 0.4.1     (2018-01-04) - add use of proxy server (look at file 'config/')
* 0.4.0     (2018-01-03) - add option '--ftp-conn'
                           # uploading all files from --out-dir to GRUAN
                           # delete all uploaded files locally
                           # search in directory 'config' if a file with
                             <NAME>.properties exist (e.g. GruanIncomingRawFTP
                             .properties)
                           # following two default FTP connections are
                             included: GruanIncomingRawFTP and
                             GruanIncomingTestRawFTP
```

E.3 History of Version 0.3.x

```
* 0.3.5     (2017-12-19) - bug fixed: in case of using *SpecialValues.
                           properties files
* 0.3.4     (2017-12-18) - add option '--no-create-if-corrupt'
                         - bug fixed: in case of missing *SpecialValues.
```

```
                             properties files (DC3DB)
                           - bug fixed: detecting double files (no comparing of
                             same file anymore)
* 0.3.3     (2017-12-14) - bug fixed: in case of missing *SpecialValues.
                             properties files (MWX)
                           - better error message in case of missing
                             file_* keys in change file
* 0.3.2     (2017-12-04) - optimize handling of special values
* 0.3.1     (2017-12-01) - add possibility to block files and to move blocked
                             files to a quarantine directory
                           - add option '--block-formula'
                           - add option '--move-blocked-files'
* 0.3.0     (2017-11-29) - add detecting and moving double files to a
                             quarantine directory
                           - add option '--move-double'
                           - add option '--quarantine-dir'
```

E.4 History of Version 0.2.x

```
* 0.2.27    (2017-11-01) - improve handling of double/after launches with
                             more than 2 parallel launches
* 0.2.26    (2017-10-23) - add handling of DC3DB-ADD (same like DC3DB)
* 0.2.25    (2017-03-31) - changed handling of parse errors (can be ignored
                             or not ignored now)
* 0.2.24    (2016-11-09) - bug fixed: improved file deletion
* 0.2.23    (2016-11-08) - improved file deletion (because error with last
                             MWX file)
* 0.2.22    (2016-10-20) - new version for Lauder (Invercargill)
                           - with MWX support
* 0.2.21    (2015-10-05) - new version for Sodankyla
* 0.2.20    (2015-09-30) - new compiled only
* 0.2.19    (2015-08-25) - renewed NetCDF libraries
* 0.2.18    (2015-07-03) - new compiled only
* 0.2.17    (2015-04-24) - compiled with java-1.8.0-openjdk-1.8.0 (to JRE 1.7
                             features)
                           - Add JEval library to jar
* 0.2.16    (2015-02-02) - arbitrary 'SpecialValues' with DC3DB files
                           - Add option '--check-gap'
                           - automatic calculation of check1_date to check5_date
                             # check1_date is launch_date minus 'check-gap'
                             # check2_date is check1_date minus 'check-gap'
                             # ...
* 0.2.15    (2015-01-02) - bug fixed: related to extraction of metadata from
                             DC3DB files (Config)
* 0.2.14    (2014-12-11) - bug fixed: related to 'SpecialValues'. Now empty
                             values allowed.
* 0.2.13    (2014-10-29) - extended possibilities of extraction of metadata
                             (SpecialValues)
* 0.2.12    (2013-11-19) - extension of file definitions with CODE
* 0.2.11    (2013-09-13) - Better handling of cancelling if files already
                             exist
* 0.2.10    (2013-08-30) - DC3DB file can handle RS80 now
                           - standardised internal configuration files
                             (Properties)
* 0.2.9     (2013-08-26) - Add new automatic parameter 'creation_tool'
* 0.2.8     (2013-02-12) - Correction of compare-method of class
```

```
                             TemporalChange (sometimes crashes)
* 0.2.7     (2013-01-07) - compare files within a time step using SN
                             (if possible)
                           - new option --launch-gap [default 10 min]
* 0.2.6     (2013-01-04) - automatic detection of correct time range
                           - new options --auto-date and --start-time
                           - use sub-directories during upload (e.g.
                             ‘Lindenberg/Radiosonde’)
                           - improved detection of several errors and issues
* 0.2.5     (2012-12-20) - correction of the handling of dual launches
                             (mostly afterstarts of a failed/cancelled launch)
                           - extract launchDate and SN from DC3DB files
                           - improved error handling in case of corrupt files
                             (during parsing)
* 0.2.4     (2012-08-07) - Well running version with new features related to
                             dual launches
```

Acronyms

CRC	Cyclic redundancy check
DPS	Data processing system
FTL	File template language
FTP	File Transfer Protocol
GMD	GRUAN meta data
GRUAN	GCOS Upper-Air Network
gtRsl	GruanToolRsLaunch
GUI	Graphical user interface
IP	Internet Protocol
ISO	International Organization for Standardization
JRE	Java Runtime Environment
UTC	Coordinated Universal Time
RLC	GruanRsLaunchClient
LC	Lead Centre