GLEAM and HyGLEAM

Software Documentation

Version 5.0

Wilfried Jakob
KIT, Campus North, Institute for Automation and Applied Informatics (IAI)
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
email: wilfried.jakob@partner.kit.edu

Contents:

1 Introduction....3
2 The Package Concept and its Implementation in GLEAM....5
2.1 Motivation and Idea of the Package Concept....5
2.2 Implementation of the Package Concept in GLEAM....6
2.2.1 Package Header File <pack>.h....7
2.2.2 Internal Header File <pack>Int.h....7
2.2.3 Language Header Files <pack>_<lg>.h....7
2.2.4 Module Files <pack>_<modul>.c or *.c....7
2.2.5 make Include File <pack>_inc.mak....7
2.3 Exceptions to the Package Structure Defined in 2.2....7
2.4 Directory Structure....8
2.5 Content of the sources Directory....9
2.6 Make Files....10
2.7 Initialization Files....13
3 The GLEAM Software System....14
3.1 Compiling and Linking....14
3.1.1 Switch File schalter.h....14
3.1.2 make Files....17
3.1.3 Some Standard Configurations....18
3.2 Style Guide....18
3.3 Packages....20
3.3.1 Package sys....20
3.3.2 Package ctio....21
3.3.3 Package fbhm....21
3.3.4 Package lgsw....22
3.3.5 Package chio....23
3.3.6 Package bew....23
3.3.7 Package appl....23
3.3.8 Package hmod....23
3.3.9 Package aufg....24
3.3.10 Package chedi....24
3.3.11 Package simu....24
3.3.12 Package evo....25
3.3.13 Package webIO....26
3.3.14 Package men....26
3.3.15 Package parPop....26
3.4 Cross Reference Tool gxref....26
4 Literature....27

Updates:

From V4.2 to V5.0:

Translation into English and update according to the published GLEAM and HyGLEAM software. The documentation is in some parts more extensive than the published version.

1 Introduction

The software is structured so that the following executable programs can be created:

1. with a simple **textual user interface**, abbreviated: **TUI**
2. with **graphical user interface**, abbreviated: **GUI**[1] (not supported)
3. **command line version**, abbreviated: **CLV**

Currently only a simple line-oriented user interface with menu control for a Unix shell is implemented as a textual user interface (TUI). This version is mainly characterized by the fact that all user IOs have been centralized and messages occurring in the depths of the SW are managed in a multi-line text buffer instead of being output immediately.

Conventions:

- **`Files`**, **`directories`** and **`package names`** are written in **`bold face Liberation Mono font`** if they appear in the text.
- `Constants`, `type declarations`, `variables` or `routines`[2] can be recognized in the text by the `Courier Font`.

Terminology:

Since an Evolutionary or a Memetic Algorithm is closely linked to an application, especially by a coresponding gene model and evaluation, we speak of a GLEAM or HyGLEAM application, or an **application** for short. An application can be represented by at least two program **versions**: the CLV and the TUI version. Of these, linguistic **variants** can exist, currently for English and German. The latter has influence on the initialization files which contain name-value pairs, i.e. on the EXP and TSK files [1, Sect. 3]. In this sense, the distribution contains three applications in their own application directories, namely:

- GLEAM/AE in the `gleam_ae` directory containing two standard or test applications, namely a set of benchmark functions and a collision-free path planning task for industrial robots, see [2].
- HyGLEAM/A in the `hy_gleam_a` directory containing the same two standard applications.
- HyGLEAM/OPALV in the `hy_gleam_opalv` directory containing a scheduling problem from process industry (`OPAL/V`) in addition to the two standard applications [2]. This application servers as an example for the integration of a new task into HyGLEAM [2] resulting in a new application in the sense described above.

Some remarks on Program Generation

Separate main programs and **`make`** files are available for the TUI and CLV versions. Both versions are based on common source packages, which are configured by conditional compilation controlled by macro definitions used as switches centrally managed per application for the respective application by the a file called **`schalter.h`**. Also, not all modules of a package are included in the CLV, because especially the modules of the user interface are superfluous. Details are documented in sections 2.6 and 3.1.

1: The creation of this version was aborted. It is mentioned here, because in some places of the SW there is talk about a GUI.

2: The term *routine* summarizes *functions* and *procedures*. Procedures correspond to `void functions` in C, i.e. functions that do not return a result. The term routine is often used in the source code comments.

The previous implementations include the following operating systems, but not all of them are supported anymore. The latter are only listed here because they still occur in some packages (e.g. **roco**). The macro switches are based on the OS names.

- **BS_LINUX** Currently supported operating system (Ubuntu, Debian and earlier Suse-Linux) with gcc or g++.
- **BS_SOLARIS** Solaris version (Unix). Has not been maintained since the Sun computers at the institute were abolished at the beginning of this millennium. But it should still work and is still included in several sources.
- **BS_DOS** Old DOS version with BGI. Outdated and no longer supported. Currently only found in Package **roco** (Rosenbrock- and Complex-local hill climbers)

GLEAM and HyGLEAM are written in C, while the two local search methods, which are integrated in HyGLEAM as standard, are written in C++. C was chosen for the implementation at the 2nd half of the 90s, because tests had shown that the functions essential for an EA run about four times faster in C than in C++. The integration of the two local search procedures as third party software written in C++ was done by passing the parameter via a file interface, while vice versa, the call of the evaluation function contained in GLEAM from the local procedures was realized by a function call. The use of the file interface was of course only intended as a short-term makeshift solution. But as is well known, nothing lasts as long as a temporary solution.

This documentation is broader than the published scope of GLEAM and HyGLEAM. This is especially true for the interface to external simulation services (ESS) via a web interface and for cluster parallelization based on a structured population according to the island model.

2 The Package Concept and its Implementation in GLEAM

The software is structured according to the package concept. A package combines logically related functions and data structures and can consist of several modules (C-files).

2.1 Motivation and Idea of the Package Concept

Figure 1 shows a conventional modularization in C: Each module (= c-file) has a header file (.h), which contains the export of the module. Exported, i.e. made known to the outside, can be the declarations of constants, data types, variables and functions. This modularization method has two disadvantages. At first it is not obvious to the file pairs whether they logically or semantically belong together and realize a sub task of the overall system or not. This could still be achieved by naming conventions, whereas the second problem can not be solved that easy: The logically related and accordingly color-coded modules will export data and routines, which are only needed by other modules of the same "color group". The other modules, however, now know this part of the export, which is actually none of their business, and can access it intentionally or not. This can result in side effects that are difficult to detect. Thirdly, the SW becomes more confusing and difficult to maintain and extend.

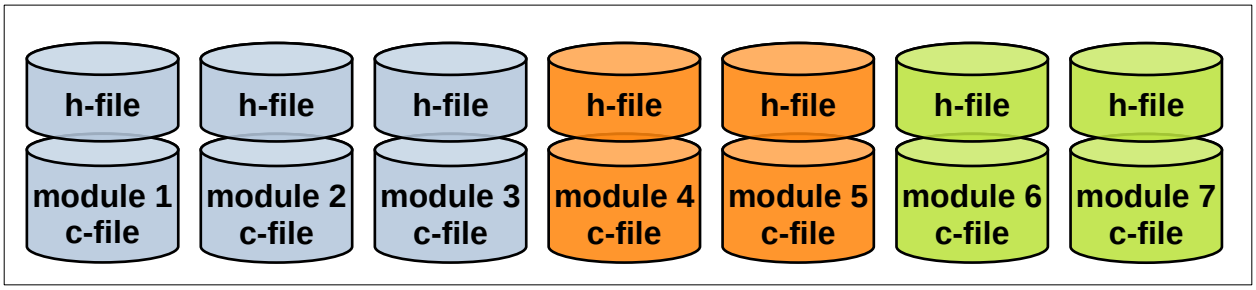

Fig. 1: Conventional modularization in C

For these reasons the package concept was developed. It allows an improved structuring of the software with the help of conventions that must be strictly adhered to using the means of the programming language C. The following rules apply to the packages:

- Each package has an h-file for package-local declarations, which is or can be included by each module (C-file) of the package.
- Each package has an h-file for package export and only this may be included by modules of other packages.

Figure 2 shows the modules of Fig. 1 organized as packages. Only the header files of the package export are visible to the outside and may be used (included) by other packages or the main program. Since the C compiler does not monitor this, modules of other packages can of course include an other internal header file. Compliance with the convention depends on the discipline of the programmers and must be enforced by the project manager.

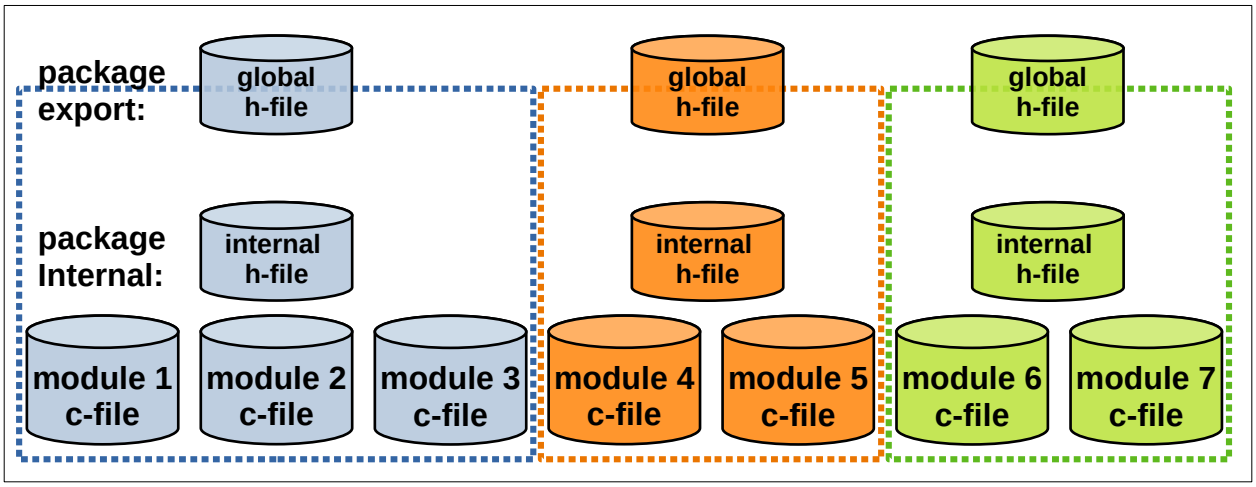

Fig. 2: Modularization according to the package concept

2.2 Implementation of the Package Concept in GLEAM

The package concept was extended by functionalities for software generation (make) and language selection (currently only German and English).

GLEAM and HyGLEAM are translated and linked with the **`make`** utility. **`make`** allows to limit the translation to the actual updated sources considering dependencies (includes) and a complete regeneration (build function) by deleting the objects. For this purpose, the **`make`** file(s) must describe what depends on whom and how to generate the dependent parts. Thus, a **`make`** file contains the compiler and linker calls to be used including switches, the directories, the dependencies of the objects on source files, a list of the objects to be linked, etc. Therefore, the **`make`** file also serves to document the generation of an executable and the involved packages and their modules. Since a **`make`** file can include other **`make`** files, it is possible to split them according to the package structure. For this reason, packages in GLEAM not only contain the source files (.c and .h files) but also at least one include file for the **`make`**, which describes the dependencies of the modules in the package. It also names related module groups with their own name, which is referred to in the higher **`make`** files. Originally, there was exactly one include file per package. In the meantime, some have more than one, see Sect. 2.6.

In addition, each package contains language-specific .h files, which contain texts to be displayed as constant definitions. Their selection is controlled by the central header file **`schalter.h`**.

Each package has a short name (usually three to five characters) which is specified below with **`<pack>`**. The minimum content of a package includes the package header file **`<pack>.h`** with the package export, a module (.c-file) and the `make` include file named **`<pack>_inc.mak`**. Depending on the size of a package, an internal header file named **`<pack>Int.h`** and the language files are added, currently **`<pack>_d.h`** and **`<pack>_gb.h`**.

For each package there is exactly one subdirectory called **`<pack>`**, which contains all source files of the package and the **`make`** include file(s). There is nothing else to be found in the directory of the package except a possible directory named **`testfield`** with test frames for modules of the package including their **`make`** and data files. The few exceptions of files beyond this scope are described in Sect. 2.5. All source files have a uniform structure and documentation, which is described in Sect. 3.2.

The following sections briefly describe the files of a package.

2.2.1 Package Header File `<pack>.h`

It contains the complete export of the package, i.e. constant and type declarations, the `extern` declarations of global variables and routines. The comment section contains a short description of the Package and a list of all contained modules.

2.2.2 Internal Header File `<pack>Int.h`

It contains all constant and type declarations as well as the `extern` declarations of package-global variables and routines, which must be exchanged in the package between the modules (C-files) of its implementation. It contains nothing that has to be visible from the outside and nothing superfluous, i.e. nothing that is required by only one module of the package.

For each routine and for each global variable of this header file there must be a comment indicating in which C-file of the package the corresponding declaration or implementation is located.

2.2.3 Language Header Files `<pack>_<lg>.h`

For each language used there is a language header file with constant declarations of strings used in the modules of the package, if routines of the package generate messages. **`<lg>`** stands for the used language. Currently these are **`d`** for German and **`gb`** for English.

2.2.4 Module Files `<pack>_<modul>.c or *.c`

Each package consists of at least one module file containing the implementation of the package. The modules of a package exchange declarations via the package header file and the internal header file. Apart from the exceptions described in Sect. 2.5 there are no other header files. In cases where the Package consists of only one module, the module is usually called **`<pack>.c`**.

2.2.5 `make` Include File `<pack>_inc.mak`

Each package has such an include file for the **`make`** utility. These files require the declarations contained in the global include files **`glob_inc.mak`** and **`common_def.mak`** and are therefore independent of directories and paths. It contains one or more listings of objects and dependencies. Different groups of objects can result, for example, from translating with and without user interface modules.

In some packages there are additional include files, see Sect. 2.6.

2.3 Exceptions to the Package Structure Defined in 2.2

Third-party software that has been integrated into GLEAM or HyGLEAM will usually not comply with the rules of the package concept. So far, this is the implementation of the two local search methods Rosenbrock- and Complex-Algorithm, whose sources are located in the pseudopackage **`roco`**. It contains **`roco_inc.mak`** for the **`make`** utility and two header files with the call interface of both procedures: **`complbox.h`** and **`rosencon.h`**. The **`roco`** package is part of HyGLEAM.

2.4 Directory Structure

The package concept is reflected in the structure of the directory **sources** containing the program sources. There are two subdirectories for the standard and the application-related packages. In parallel, there is a directory that may be named after the operating system used, for example, Linux in the following and which contains directories corresponding to the (Hy)GLEAM applications in which the generation of executables takes place. This structure is supported by the **make** files included in the distribution. It is described in detail below, assuming that everything is located in a root directory <root>, whose name can be chosen arbitrarily. It contains the directory **sources** and exemplary the directories **linux** for the executables and **InitFiles** for initialization files of GLEAM and HyGLEAM respectively. So we have the following top level structure:

```
<root>                    root directory, usually called gleam
  |
  +-- InitFiles           set of commonly used initialization files. Among them the
  |                       error message files ftext_d.txt and text_gb.txt
  +-- linux               contains the directories for the translated applicatiions
  |
  +-- sources             main programs, central .h files, common_def.mak, etc.
      |
      +-- appl_packages   application-specific packages
      |
      +-- packages        standard packages of GLEAM and HyGLEAM
```

The different initialization files are described in the user manual [1], the documentation of existing and new GLEAM applications [2] and in the documentation of gene model files [3] and the parameterization of the genetic machine [4].

The **linux** directory can be seen as an example of an operating system oriented directory, which combines the directories for the creation of executables for a target operating system. In the current distribution these are the directories **gleam_ae**, **hy_gleam_a**, and **hy_gleam_opalv**, each of which contains the following files:

- **schalter.h** Contains the macro switches, which have to be set depending on the operating system, version, and variant of the software.
- **makefile** **make** file for the interactive version
- **<appl>CLV.mak** **make** file for the command line version of application **<appl>**. With standard GLEAM this would be e.g. **gleamCLV.mak**
- **glob_inc.mak** Global **make** include file of this application. This file is included by the upper two and contains all application specific definitions that are independent from the version or variant.
- **gleam_<lg>.h** Language-specific text files for software version and generation date

Figure 3 shows the complete directory structure starting from a root directory **<root>**. This directory structure is supported by the **make** files included in the distribution (see Sect. 2.6) and should not be changed.

```
<root>                      root directory, usually called gleam
  |
  +-- InitFiles             error message files, set of commonly used initialization files
  |
  +-- linux                 summarizes the program directories for Linux
  |   |
  |   +-- gleam_ae          standard GLEAM
  |   |   +-- testfield     for test purposes, usually contains some EXP files
  |   |
  |   +-- hy_gleam_a        standard HyGLEAM
  |   |   +-- testfield     for test purposes, usually contains some EXP files
  |   |
  |   +-- hy_gleam_opalv    standard HyGLEAM extended by the OPAL/V application
  |       +-- testfield     for test purposes, usually contains some EXP files
  |
  +-- sources               main programs, central .h files, common_def.mak
      |
      +-- appl_packages     summarizes application-specific packages
      |   +-- opal          packages of the additional OPAL/V application
      |
      +-- packages      summarizes all standard packages
          +-- appl      interface for application-specific extensions
          +-- aufg      program parameter, task-oriented data of the robot application
          +-- bew       evaluation and penalty functions, BEW file I/O
          +-- chedi     chromosome output and chromosome editor
          +-- chio      chromosome file I/O (MEM and AKS files)
          +-- ctio      elementary terminal functions, scrolling, menu functions
          +-- evo       evolution control, genetic operators, genetic repair, EVO file
          +-- fbhm      error messages and exchange buffer for texts (virtual screen)
          +-- hmod      gene model, hamming distance, chromosome creation, MOD file
          +-- lgsw      basic list routines
          +-- men       many menu functions, package initialization, initialization files
          +-- parPop    modules of master and slave of the mpi version (unsupported)
          +-- roco      Rosenbrock procedure and Complex algorithm for HyGLEAM
          +-- simu      simulator frame for integrated applications and ext. simulators
          |   +-- lskp  robot simulator
          +-- sys       system related functions, random numbers, pipe administration
          +-- webIO     I/O of chromosome and result lists using the CURL library
```

Fig. 3: Directory structure supported by the **make** files

2.5 Content of the sources Directory

Figure 3 lists a number of (global) files in the **sources** directory. They are briefly introduced in this section:

chaindef.h declarations of genes and chromosomes or actions and action chains.

vers_<lg>.h short description of the different GLEAM applications, here called versions, see menu item Info/Version

f_codes.h error codes matching the texts of **ftext_<lg>.txt**

fcodes_par.h	error codes matching the texts of **ftext_mpi_<lg>.txt**[3]
glob_<lg>.h	language specific text files with some global texts for all versions
gleamCLV_<lg>.h	language specific text files with texts for the main program of the CLV
gleam_usr_<lg>.h	language specific text files with texts for the main program of the interactive version with textual user interface (TUI)
szsunsol.h	operating system specific German umlauts as constant declarations. Originally intended for Sun/Solaris and adapted to Linux.
gleam_ae.c	main program of the interactive TUI version of (Hy)GLEAM
hyGleamEngine2.c	main program of the command line version (CLV) of (Hy)GLEAM
essEvalFrame.c	main program for wrapper of external simulators for the external simulation services (ESS)[3]

The header files of this directory are included by modules from several packages. The file **ttype.h**, which is provided by the base package **sys** and which is included by all modules, also falls into this category. For more details see Sect. 3.3.1.

2.6 Make Files

The **make** files are used to create the executable programs. In GLEAM the property of **make** files to be able to include others is used to divide them among others according to the package structure, see also Sect. 2.2. There are four levels of **make** files:

1st level:

makefile for the respective TUI versions and e.g. **gleamCLV.mak** or **hyGleamOpalvCLV.mak** for CLV versions. They are located in the generation directories such as **gleam_ae** or **hy_gleam_opalv**. They contain the following:

- Definition of the file name of the executable. Should be equal to the EXE_NAME of the **schalter.h** file.
- Definition of the compiler
- Definition of the compiler witches
- Definition of the object groups to be linked. An object group contains a list of object file names.
- Description of global dependencies
- Linker section
- Include of **glob_inc.mak**, which is read by all **make** files of a generation directory.

Call parameters:

- none: generates the binary and all necessary object files.
- clean: deletes the binary and all object files (.o files) of the packages involved.
- min_clean: deletes the binary and an application specific part of the object files.

2nd level:

The **glob_inc.mak** file of a generation directory. It contains the common and thus global definitions of an application for the interactive and the command line versions. This file contains in particular:

- Definition of the **<root>**-directory
- Definition of some general directory declarations relative to **<root>**

3: This is not part of the distribution and/or is not supported by it.

- Definition of application-specific directory declarations relative to **<root>**
- Include of application-specific package include files (**<pack>_inc.mak** and **<pack>_<var>_inc.mak**, where **<var>** stands for **ess**, **mpi** or **cec**)
- Definition of the two standard object file groups for TUI and CLV
- Definition of special dependencies
- Eventual special rules for object file creation
- Definition of the files to be deleted by `clean` and `min_clean`
- Including the general definitions from **common_def.mak**
- If necessary, include of **<pack>_common_inc.mak**

3rd level:

common_def.mak: Contains the global definitions of all applications and program versions or variants. It is located in the **sources** directory and includes in particular:

- Definition of standard directory declarations relative to **<root>**
- Description of general dependencies
- Include of the standard package include files common to all versions (**<pack>_inc.mak**)
- Definition of the standard include directories for the compiler call
- Description of the dependencies of both main programs
- Rules for object generation

4th level:

Package **make** include files. There is always the file **<pack>_inc.mak**. In addition, there may be others, as described below. The package include files contain descriptions of the dependencies and the object groups for the various versions and variants. They are located in their respective package directories.

In the course of the development of the program system, the concept of application-neutral package include files, which map the differences to alternative object groups, had to be abandoned because different dependencies had to be taken into account. Specifically, this involved the applications for the external simulation services (**ess**), for the MPI-based parallel software (**mpi**) and for the application based on the CEC'05 benchmarks (**cec**). In addition, a general distinction must be made between the object groups for the version with textual user interface (TUI), with graphical user interface (GUI) and the command line version (CLV). Table 1 gives an overview.

Note: The version supporting a graphical user interface, the applications based on external simulation services, and the MPI-based parallel applications are not included in this distribution. In particular, the GUI version was never finished and given up. The MPI-based parallel application was used for some studies and than no longer supported. In contrast to that the CEC'05 benchmarks can be easily reactivated. This is described here, because there are still some traces of these versions and applications in particular in the package **make** include files.

Package	Listed in common_def.mak	Seperated Include Files	Object-Groups			
			OBJ	TUI	CLV	GUI
sys	✓		✓			
ctio	✓			✓	✓	
fbhm		✓ mpi	✓			
lgsw	✓		✓			
chio		✓ mpi	✓			
chedi	✓			✓		✓
aufg		✓ ess		✓	✓	
hmod	✓		✓			
bew	✓			✓	✓	✓
appl	✓		✓			
lskp	✓			✓	✓	
simu		✓ ess, cec		✓	✓	
evo		✓ ess, mpi		✓	✓	
roco			✓			
webIO			✓			✓
men		✓ ess, mpi		✓	✓	✓
parPop			✓			
opalv			✓			

Table 1: Package **make** include files

In **common_def.mak** there are only those common package **make** include files included, which are needed by all variants and versions. Therefore the packages **roco**, **webIO** and **opalv** are not marked in the column **Listed in common_def.mak**.

The right part of the **Seperated Include Files** column specifies the additional package include files. The **<pack>_inc.mak** file always exists. If there are additional package include files besides the **<pack>_inc.mak**, they each include a common file **<pack>_common_inc.mak** located in the package directory

Table 2 gives an overview of the modules with switch-dependent (see file **schalter.h**) references to other packages, which are reflected in corresponding `#include` statements.

Module	Package	Include of
fbhm.c	fbhm	mpar.h
chain_io.c	chio	mPar.h, chioInt.h
tsk_data.c	aufg	webIO.h
simucmen.c	simu	webIO.h, aufg.h
simu_gsw.c	simu	webIO.h
cec_sim.c	simu	cec/global.h, cec/rand.h
evo_gsw.c	evo	mPar.h
evo_steu.c	evo	bew.h, simu.h
men_gsw.c	men	webIO.h
f_load.c	men	mPar.h

Table 2: Package-includes depending on the switches from the respective **`schalter.h`** files

The described specifications have the following effect when preserving the directory structure described in the previous chapter:

- When porting to another Linux machine using GNU-C, only the definition of the **`<root>`** directory has to be adapted. With different compiler versions an adjustment of the compiler switch definitions of the **`make`** files from level 1 can become necessary.
- When porting to a different operating system and compiler, in addition to the previously mentioned customizations, adaptations of the path declarations to the target file system may be necessary. For example, a change from Unix to Windows paths would affect all levels because of the different directory separators.

2.7 Initialization Files

It has proved practical to group the initialization files of some applications centrally in a directory **`InitFiles`** directly under **`<root>`**. In this directory are also the error text files, to which a soft link can point, which is located in the respective working directories. Alternatively, the environment variable GLEAM_ROOT can be defined, which contains the path to the **`InitFiles`** directory. Only the EXP files, any MEM files and local files are then located in the respective working directories.

The prepared experiments and their EXP files are listed and briefly described in [2].

3 The GLEAM Software System

GLEAM and HyGLEAM are based on the same packages and modules in their interactive (TUI or GUI) and command line (CLV) versions, as far as they perform common tasks in all versions. The control of which parts to compile and link is done by the **make** files and by macro definitions used as switches for conditional compilation, which are to be set in the central switch file per application **schalter.h**. The details are described in sections 2.6 and 3.1.

The software is written in ANSI-C and has a uniform design. The underlying conventions should be observed for extensions and adaptations. They are described Sect. 3.2. The concept of inline documentation in module and procedure headers was implemented.

3.1 Compiling and Linking

In the switch file **schalter.h** it has to be set in the header whether an interactive or a command line version of GLEAM or HyGLEAM is to be generated for which operating system or application. Regarding the **make** utility there are at least two files, one for the interactive version (**makefile**) and one for the command line version (z.B. **hyGleamCLV.mak**).

Note: make file and version or application to be compiled MUST match regarding the switch settings in the schalter.h file used. Otherwise there will be numerous inexplicable error messages!

The **schalter.h** files included in the distribution are set to generate an English language TUI variant. The root directory declaration included at the beginning of the **glob_inc.mak** files is:

```
ROOT = /home/wilfried/gleam
```

This declaration must be adapted to the conditions of the target system!

3.1.1 Switch File schalter.h

Conditional compilation switches are used to control the generation of the different applications, versions, and variants. They are in the file **schalter.h** in the respective application directory. The file is included by each source, whereby there can be exceptions with third-party SW. The most important control switches are placed at the beginning of the file and are explained below. A number of other switches and settings are derived from these.

The **schalter.h** file of HyGLEAM/A located in the HyGLEAM directory **<root>/linux/hy_gleam_a** serves as the example for the explanation. It contains the settings shown in Fig. 4, from which the following is derived:

- operating system specific settings and size defaults,
- details of the user interface,
- the inclusion of the two standard and possibly other LHCs with MIT_LSV[4] set and
- language-specific definitions of file names

In addition, the file contains

- default file names,
- the include statement for **ttype.h**, so that every source includes this file and
- the operating system specific selection of the umlaut file (here **szsunsol.h**).

4: LHC stands for local hill climber or in German lokales Suchverfahren. “With” is “mit” in German and thus, the switch is called MIT_LSV. There are other switches and variables with German names as it would be very time-consuming to translate all of them and to make sure that no errors are introduced into the SW as a result.

```
/* ---------- Global Switches for Versions and Operating Systems: --------- */
#undef BS_SOLARIS        /* set: Sun/Solaris     } only one BS_... switch  */
#define BS_LINUX          /* set: Linux           } may be set!             */
#define GLEAM_USR         /* set: With user interface, otherwise CLV        */
#undef EXT_MATLAB_SIMU   /* set: Matlab/Matpower as ext. Simu. } only one  */
#undef EXT_SIMU_SERV     /* set: extern. Simu.Services (ESS)   } may be set*/
#undef MPI_PAR_VERS      /* MPI based parallel implmentation for Linux     */
#undef MIT_AEHNL_CHECK   /* with similarity check before simulation        */
#define MIT_LSV           /* set: GLEAM with local hill climbers           */
#undef CEC_MBF            /* Logging and job term. for CEC-MBFs. CLV ONLY! */

/* --------------------------- Language Selection: ----------------------- */
#undef DEU               /* set: Deutsch.                    } only one   */
#define ENG               /* set: English.                     } may be set */

/* ------------------- Name of Program and Executable: -------------------- */
#ifdef MIT_LSV
  #define PROGRAM_NAME   "HyGLEAM/A"
  #ifdef GLEAM_USR
    #define EXE_NAME       "hyGleam"
  #else /* no (textual) user interface */
    #define EXE_NAME       "hyGleamCLV"
  #endif /* no (textual) user interface */
#else /* no MIT_LSV */
  #define PROGRAM_NAME   "GLEAM/AE"
  #ifdef GLEAM_USR
    #define EXE_NAME       "gleam_ae"
  #else /* no (textual) user interface */
    #define EXE_NAME       "gleamCLV"
  #endif /* no (textual) user interface */
#endif /* no MIT_LSV */

/* --------------------------- User-Interface: --------------------------- */
#if defined(GLEAM_USR)
  #define TEXT_MODE        /* textual user interface  }     alter-          */
  #undef GUI_MODE         /* grafical user interface }     native          */
  #define GNUPLOT          /* integration of GNUPLOT for eval.fct.display   */
#else /* ------------------------- no GLEAM_USR -------------------------- */
  #undef TEXT_MODE        /* textual user interface  }     alter-          */
  #undef GUI_MODE         /* grafical user interface }     native          */
  #undef GNUPLOT          /* integration of GNUPLOT for eval.fct.display   */
#endif /* no GLEAM_USR */

/* --------------------- Global Switch for Test Code: --------------------- */
#define GLO_CHECKS         /* details are determined in the packages        */

/* ---------------- Switches for Statistics and Log File: ----------------- */
#define MIT_OP_STATISTIK       /* opt. statistics of genetic operator usage */
#define LOGFILE_MODE    "at"   /* writing file in append mode               */

/* -------------------------- Simulator Switches: ------------------------- */
#define MITSIMULOG         /* with log file for the simulator (interface)   */
#define SIMU_LOGF_MODE "wt"/* writing the file as a new one                 */
#undef SIMU_DBG           /* debug output to scroll window and log file    */
#undef ONLINE             /* data exch.with simulator OR test I/O from file*/

/* ----------------- Names of the Environment Variables: ------------------ */
#define SIM_MOD_ROOT_ENV     "SIM_MOD_ROOT"
#define GLEAM_ROOT_ENV       "GLEAM_ROOT"
```

Fig. 4: Switches to be set in the **`schalter.h`** file for the HyGLEAM/A application

Section *Global Switches for Versions and Operating Systems*:

- Set of Switches to select the operating system: **`BS_*`**:
  Only one switch of this set may be active and that is currently **`BS_LINUX`**. The **`BS_SOLARIS`** switch should no longer be used without checking the sources.
- **`GLEAM_USR`**:
  Decides between the interactive version (set as in Fig. 4) and the command line version (not set).
- **`EXT_MATLAB_SIMU`**:
  Is to be set when integrating the Matlab simulator (and thus also MatPower) into (Hy)GLEAM.
  This also applies to the modules **`essEvalFrame`** and **`essEval`** of the ESS application.
- **`EXT_SIMU_SERV`**[5]:
  Simulator interface to the external simulation services (ESS) under realization of a master-slave parallelization for evolution runs. Single simulations via the ESS interface are also possible.
- **`MPI_PAR_VERS`**[5]:
  If this switch is set, an MPI-based engine is created which runs under MPI in parallel on a parallel computer, e.g. the HC3 cluster. Only valid if **`BS_LINUX`** is set and both **`GLEAM_USR`** and **`MIT_LSV`** are not set.
- **`MIT_AEHNL_CHECK`**[5]:
  Addition of modules of the **`simu/vgl`** subpackage which, for chromosomes of type 1, performs a comparison of the parameters of the chromosome with stored values and decides whether simulation is required or whether a stored value vector is similar enough. This extension has not been maintained since early 2014.
- **`MIT_LSV`**:
  Differences between GLEAM (undefined) and HyGLEAM (defined). If the switch is set, the dependent switch **`MIT_ROCO`** is set and thus the two standard LSVs (Rosenbrock method and Complex algorithm) are included. In principle, other LSVs can also be implemented and included. If the derived settings are changed, the two standard LSVs mentioned can also be omitted.
- **`CEC_MBF`**[5]:
  Special switch for performance measurements in the context of the CEC'05 benchmarks set. Only applies in conjunction with the command line version. Should no longer be used without a check of the sources.

Section *Language Selection*:
Selection of the language to be used for the user interface, the (error) messages and the name-value pairs in the EXP and TSK files.

Section *Name of Program and Executable*:
Determination of the program name and the name of the executable depending on the set switches. The latter should be identical with the name used in the associated **`make`** file. The names are used in messages of the program, and because help texts also belong to that, the name of the executable file specified here should be correct.

5: This is feature is not part of the published distribution

Section *User-Interface*:
The sub-switches are used to distinguish between the currently implemented textual user interface (**TEXT_MODE**, corresponds to the TUI version) and an unfortunately only partially implemented GUI (**GUI_MODE**). The switch **GNUPLOT** allows the display of assessment and penalty functions as well as simulation results with the help of the program **gnuplot**.

Section *Global Switch for Test Code*:
With this switch all switches in the modules for the generation of test code can be switched on or off centrally. The test code is mostly used to check the permissibility of parameters when calling routines.

Section *Switches for Statistics and Log File*:
The **LOGFILE_MODE** switch determines how the file is opened. Default is "at", i.e. as text file in append mode. If the **MIT_OP_STATISTIK** switch is set, detailed statistical information on the use of the genetic operators can also be written to the log file, provided that the corresponding program parameter has been set interactively or via TSK file.

Section *Simulator Switches*:
If the **MITSIMULOG** switch is set, a log file is created for the simulator. If an external simulators is connected, it can be used for logging the data exchange between GLEAM and the simulator. The **ONLINE** switch can be used to activate the interfacing via pipes (set) or to read the simulator's output from a file for test purposes (not set).
In the ESS-applications (external simulation services, switch **EXT_SIMU_SERV** is set) the switch **ONLINE** serves only to indicate with output of the program name and version whether it is a productive online version or a test version with file interface. The switch is then to be found in the header area.

Section *Names of the Environment Variables*:

SIM_MOD_ROOT_ENV: Path to the model files of the external simulators. Deprecated.

GLEAM_ROOT_ENV: Directory where the error message text files are expected. If **GLEAM_ROOT_ENV** is not set, these files are expected in the working directory. The program terminates if it cannot load the error texts! Furthermore, the initialization files are searched in this directory if they cannot be found in the working directory.

3.1.2 make Files

The **make** files determine what is translated and linked together. This can function however only if the macro definitions (switches) in the respective switch file **schalter.h** were set before suitably to it! This concerns in particular the switch **GLEAM_USR**.

As long as one works in the context of an application, version and variant of (Hy)-GLEAM, a supplementary translation can be started under utilization of a preceding translation run (in the present case start of the respective **make** file without parameters). Otherwise, the parameter **clean** must first be used to delete the respective executable and all object files. The subsequent call without parameters then recompiles everything, regardless of when the sources were last updated (corresponds to a *build*). In case of doubt, a **clean** is always appropriate before searching for unexplainable errors.

The structure of the **make** files and their hierarchy was discussed in Sect. 2.6.

3.1.3 Some Standard Configurations

It is possible to create several different basic GLEAM configurations using switch settings and **`make`** file definitions, of which there is usually a CLV and a TUI version.

The default configurations include the LESAK robot application and a set of mathematical test functions. This means that these can always be used (for test purposes) by suitable initialization files.

Table 3 below shows the relationships between the TUI, CLV and GUI versions, the switches and some modules that may or may not be linked to them according to the **`make`** files. The switches **`TEXT_MODE`**, **`GUI_MODE`** and **`GNU_PLOT`** are set depending on **`GLEAM_USR`**.

	gleam_ae with variants	**hy_gleam_gui**	**GleamCLV** with variants
GLEAM_USR	T	T	F
TEXT_MODE	T	F	F
GUI_MODE	F	T	F
GNU_PLOT	T/F	F/T	F
`user_gsw, scr_gsw` from **`ctio`**	yes	no	no
package **`chedi`** (all modules)	yes	no	no
`ch_ausg` (from package **`chedi`**)	yes	yes	no

Table 3: Switch settings and involved modules of the different versions

3.2 Style Guide

Basically, the files of the existing software can serve as examples for the design conventions of the source code. With software extensions it is even expressly recommended to orient oneself at the existing source code, since for time and space reasons not all details can be listed here. The rules in detail:

1 In the export header file of each package there is a short summary of the task of the package and a list of its modules with a short explanation.

2 Each module contains a comment header that includes the license notice and describes the function and managed data of the module. More complex algorithms are also documented here. It also contains the prototypes and the documentation of all exported routines of the module: First the package export and then the routines known only within the package. The comment ends with a change documentation and a line with the module status, an author note and the date of the last modification.

3 The structure of a module (C file) is:

- comment header,
- c library includes,
- package- and local includes,
- local switches (macro definitions) for conditional compilation,

- declarations of exported variables in the order:
  - package export
  - package-local variables
- local declarations in the order:
  - constants,
  - types,
  - variables,
  - routines (local to the module)
- implementations of routines
  For reasons of clarity, care was taken to ensure that the implementations of the routines are in the same order as the associated declarations of the prototypes and the documentation in the comment header. Module-local routines stand in the proximity of their first use (usually before it) and are uniformly documented at the place of the implementation.
  The routine headers are designed in such a way that the parameters are placed one below the other as a list.

4 Routine documentation is done without a formal scheme. However, it must describe all parameters used and any returned result. In addition, the functionality, the behavior in special or borderline cases, and - very importantly - the error behavior must be documented. Instead of formally following rules, this documentation is intended to enable the reader (especially one who did not write the routine himself) to use it without looking at the code.

5 Before each routine there are three blank lines and a full-width comment line of "=" containing the routine name in the middle. The start of the code and the end of the body of each routine are marked by a comment with the routine name. Blocks are marked by an identical comment after the block brackets, which improves the overview of long routines.

6 The indentation of the source text according to its logical structure is done uniformly according to the C-style `Ellemtel` implemented in **`xemacs`** (among other, indexing by 3 characters). Under no circumstances should tabs be introduced into the source text, since they make editing with **`xemacs`** very difficult.

7 The following naming conventions are determined by the fact that no integrated development environment was available at the time of the original implementation. Here first the rules, after which the largest part of the sources is designed:

- Constant names and macros serving as switches for conditional compilation are written in capital letters and, if necessary, with a separating "`_`".
- Type names are written in capital letters and, if necessary, with a separating "`_`". A **`_TYP`** or a **`_TYPE`** is always appended to these names.
- Variable names always consist of at least two characters. Auxiliary variables can be named `ii` or `ij`, but never `i` or `j`. This rule makes it much easier to find variables with **`xemacs`** or **`fgrep`**. Variable names are lowercase and, if necessary, with a separating "`_`".
- For routine names the rules for variable names apply analogously.

In addition to the "old style" of continuous upper or lower case described here, the modern style is also possible, in which upper case letters divide the name, whereby the following always applies: constant or type names begin with an upper case letter and variables or routine names begin with a lower case letter. Examples:

`globWorkerStatus` instead of `glob_worker_status` example of a variable

`MsgM2W_DoIt`	instead of `MSG_M2W_DoIt`	example of a constant
`MyNewType`	instead of `MY_NEW_TYPE`	example of a type identifier

8 Modules and packages have a three-digit version number with the following meaning: the first digit reflects significant functional changes, the second changes to the module/package export, and the third internal modifications.

9 For historical reasons, the maximum line length of a large part of the sources is 80 characters. For the sake of clarity, new source text should not contain significantly more than 100 characters per line.

Final note to the exhausted reader:
I understand that conventions are annoying. Especially when they are not your own. But the maintainability of a large software system also depends essentially on the good readability of the source code. And this includes a uniform design.
Certainly, some people will now want to regulate things differently and perhaps even better. But uniformity is worth much more than a little bit better in detail.

3.3 Packages

This section lists and briefly describes the packages that are part of the GLEAM standard and specifies their modules. Not described are the packages roco (Rosenbrock and Complex algorithms as LHCs) and mPar of the mpi application.

In this part of the documentation the terms *chromosome* and *action chain* (*AC*[6]) are used synonymously. The same applies to *gene* and *action*.

For all packages, it is specified which modules generate output to the text buffer.

This part of the documentation was only reviewed and supplemented in February 2019. There may be isolated discrepancies with the sources.

A large part of the inline documentation is in German. Only the package **`appl`** for the integration of new applications and the module **`tsk_data.c`** from the **`aufg`** package are completely documented in English. **`tsk_data.c`** is furthermore completely written in English.

3.3.1 Package sys

SYStem. The package contains all system-related functions and is intended to facilitate porting to other platforms and operating systems. Therefore, functions that are actually considered standardized in the UNIX context are also located here.

The package was redesigned during the renovation at the beginning of 2016 in such a way that all modules are included from all applications and versions and the routines not required in the respective case are hidden by the conditional compilation controlled by macros (switches).

Since memory requirements and value ranges of standard data types such as int or float depended on the hardware and compiler used when the SW was created, **`ttype.h`** was introduced, in which all data types and string sizes are defined centrally. Thus the package exports meanwhile always two files.

Additional export:

`ttype.h` belongs to the (separate) package export and is included via **schalter.h**.

6: Action chain is Aktionskette in German, abbreviated AK. The term AK is frequently used in the source code.

Modules:

sys_gsw Contains all system-related functions that are not in the other modules. These are in particular: random number generator, time and date functions, mathematical routines.

file_gsw File I/O functions, file existence check, directory routines. Output routines for file names and simulator models (text buffer).

extkoppl Low-level connection of external programs (specifically: **gnuplot** and **simpilot/eldo** or **mathematica**).

term_io Keyboard routines of the TEXT_MODE version.

3.3.2 Package ctio

C-Terminal-I/O. The package contains the low-level terminal I/O routines of the TEXT_MODE version and some string and dialog routines.

The package was completely revised and everything was deleted, which had belonged to the old BGI interface. In addition, it was divided in such a way that with TEXT_MODE not set only the module **str_gsw** is exported and included.

Modules:

scr_gsw Startup message, dialog functions, menu routines, package initialization and exit handler for terminal restoration.

user_gsw Low-level data input of different data types. Dialog routines with default values and limit check for different data types. Dialog routines for entering AC addresses[7] and file identifiers and for file selection.

str_gsw Two string editing routines for creating a time specification and AC addresses[7].

3.3.3 Package fbhm

Error handling/messages (in German: FehlerBeHandlung/Meldungen). The package provides

- error and message handling based on text files, and
- a multi-line text buffer.

It also contains two output routines that are only available in TEXT_MODE.

Error and Message Handling:

There are three classes of messages and corresponding message routines: fatal(), fehler()[8] and meldung()[9]. The parameters of a fatal/error/message are stored in one locally managed buffer, which can be emptied by an output routine and converted into ready-prepared output strings. The buffer counter, which can be queried externally, indicates whether there are any messages in the buffer at all. The calling software is informed by return states (GLEAM_OK, GLEAM_ERROR and GLEAM_FATAL) whether an error occurred with which severity. The package manages the internal error state in error_state which, in contrast to the exported err_state, cannot be reset after the level of GLEAM_FATAL has been reached.

7: The address of a chromosome or action chain (AC) consists of its fitness class and a consecutive number, see [1, Sect. 1 and Sect. 4.2.3].

8: error

9: message

When the buffer is full, further incoming messages are counted and ignored. When the buffer is emptied (fifo), a note with the number of ignored messages is then generated at the end. It is assumed that, firstly, the buffer should be large enough and, secondly, that the first messages are more likely to allow a conclusion to be drawn about the cause of the error than the most recent subsequent errors.

The message texts are read with the package initialization from the file **`ftext_<lg>.txt`**. A message text can be supplemented by up to two integer values and a string.

The `TEXT_MODE` version contains a routine that completely empties the buffer, outputs it to the terminal and, depending on the package initialization, also writes it to the log file.

Text Buffer Management:

The package contains a multi-line text buffer which allows the standard modules of the packages to prepare outputs. Standard modules implement the (Hy)GLEAM functionalities without having anything to do with a user interface. The text buffer routines fill the multi-line buffer, which can be output by routines of a higher-level user interface layer or by corresponding routines of the CLV version. The buffer consists of `MAX_BUF_ANZ` lines. This value depends on the maximum expected number of segments (`SEGM_ANZ_MAX`), which are to be displayed by the chain editor (package **`chedi`**). Acs (chromosomes) with more segments are nevertheless possible, but cannot be handled by the chain editor.

Routines for writing to the text buffer usually start with the prefix `prep_` and are usually grouped in one module per package. Exceptions to this are, for example, `mem_info()` (module **`mmsw`** in package **`lgsw`**) or `zeige_aktion()`[10] from module **`ch_ausg`** in package **`chedi`**.

There are two output routines for the text buffer. One writes it to the log file and depending on the parameter also to the terminal, the other is only for the `TEXT_MODE` and allows a "page wise" output within a scrolling area.

Module:

`fbhm`	Error text management and (error) message routines. Text buffer management. Output routines for collected (error) messages and the text buffer.

3.3.4 Package `lgsw`

Basic list management software. (in German: ListenGrundSoftware). The package contains all routines for the generation, manipulation and management of lists as required for gene representation in chromosomes.

Additional Export:

`chaindef.h`	The file is located in the **`sources`** directory and is included in the package export. But it can also be included separately.

Modules:

`mmsw`	Low-level memory management, memory usage report via text buffer.
`balisto`	Basic list manipulation, deletion and copying of chains and sequences, segment management. Function to test the integrity of an AC with report output to the text buffer.
`bacha`	Chain or chromosome memory and its management.
`ch_init`	Chain or chromosome creation and initialization.

10: “show_action” for displaying the contents of a gene or action

3.3.5 Package chio

CHain-I/O. The package contains routines for linearization and delinearization of ACs for the purpose of stream I/O. The current implementation implements binary and textual file I/O as well as mpi based I/O for the mpi application, see switches `MPI_PAR_VERS` and `SLAVE`.

Logging of I/O results to the log file and/or text buffer can be enabled and disabled.

Modules:

chain_io	File input/output of the ACs (chromosomes). Restoration of the chain structures.
mpi_chio	Higher level routines of MPI-based AC communication.

3.3.6 Package bew

Valuation (in German: BEWertung). The package implements the calculation and management of the cascaded weighted sum, see [1, Sect. 4.2.6]. It implements the evaluation sub-menu and contains routines for calculating unweighted fitness values, for weighting, for evaluation data preparation, for saving as well as restoring the evaluation data (BEW file) and for preparing GnuPlot outputs.

Modules:

bewert	Implements the I/O and the normalization and penalty functions.
bew_gsw	Display routines for the `GLEAM_USR` versions (text buffer and GnuPlot).
bew_cmen	Evaluation sub-menu of the `TEXT_MODE` version.

3.3.7 Package appl

APPLicationas. The package contains the interface for coupling specific SW for applications other than those included in the GLEAM standard. For this purpose, depending on the scope and type of the application-specific functions, the corresponding modules are to be replaced by application-oriented ones.

Modules:

appl_if	Interface module
appl_sim	Application-specific simulator and package initialization.
appl_go	Application-specific genetic operators
appl_rep	Application-specific genetic repair
appl_gen	Application-specific chromosome generation
appl_fio	Application-specific additional file I/O with result output to the text buffer.

3.3.8 Package hmod

Gene or action model (in German: HandlungsMODell). The package contains routines for reading the MOD file, generating segmented ACs (chromosomes) and determining the Hamming distance between two chromosomes.

Modules:

mod_data	Reading the MOD file and managing the field of action resp. gene descriptors `act_descr`.

segm_gen General AC (chromosome) generation and segmentation and parameter modification. General forms of Genetic Repair.

hamming Calculation of the Hamming distance of two ACs resp. chromosomes.

3.3.9 Package aufg

Task (in German: AUFGabe). The package implements the application-specific task sub-menu of the LESAK application and contains routines for task data preparation and for saving and restoring the task data (TSK file). The package was later extended by the module **tsk_data** for the administration of the different program parameters, which are in the TSK file and in the EXP file (filenames).

In the command line version the TSK file is also and above all used to parameterize optimization jobs.

Modules:

aufg_gsw Saving and restoring task data (TSK file), application-neutral basic menu functions, including in particular a display function (via text buffer).

aufgcmen Task sub-menu of the TEXT_MODE version.

tsk_data Management of boolean, integer, real-valued and string program parameters. In addition there are enumeration types, which of course are mapped to integers.

3.3.10 Package chedi

CHain-EDItor. The package contains routines for action resp. gene and chain output with GLEAM_USR set and the chain editor for TEXT_MODE.

Modules:

ch_ausg Output of chains and chain elements using the text buffer from the package **fbhm**.

c_ch_edi Chain editor for the TEXT_MODE.

3.3.11 Package simu

SIMUlator. The package contains the simulator frame, the coupling of external simulators and the simulator menu for the TEXT_MODE. In addition, it contains in the subpackage **lskp** in the subdirectory of the same name the robot simulator of LESAK (without graphics outputs and menus) and with an adapted evaluation, which provides only the raw data.

In addition to the **simu_gsw** module, the **gen_kopl**, **eldokopl** and **mathkopl** modules also generate output to the text buffer.

Modules:

simucmen Contains the interactive part, the simulator menu of the TUI version.

simu_gsw Basic routines of the simulator displays (text buffer).

simu The simulator framework with data preparation and evaluation.

mbf_sim Mathematical benchmark functions (MBF), test functions

cec_sim MBFs of the CEC'05 collection (functions in the **cec** subdirectory)

matlab_koppl Interface to an external Matlab simulator

extSimuServKoppl Interface for connecting to the external simulation services (ESS)

ext_sim	Interface to external simulators and general services for their coupling.
gen_kopl	General interface for coupling to external simulators
mathkopl	Coupling to the external simulator "mathematica".
eldokopl	Coupling to the external simulator „simpilot / ELDO“.

Unterpackage lskp:

joints.h	Package export of two data types (anachronistic filename).
rob_sim	LESAK robot simulator (without animation).
rob_gsw	Basic and display routines of the robot simulator (text buffer).
rob_dial	Dialog routine `teach_in()` of the robot simulator to specify space points in the TUI version.
lsk_grep	LESAK plausibility tests and genetic repair routines.
pasromin	Basic routines and declarations of geometric data types.
rob_int.h	Package-global h-file of the robot simulator.
lsk_mod.h	Constant declarations of the action model (package-local)
no_rob	Dummy module for applications without the LESAK simulator.
robsimhy	Modified robot simulator for HyGLEAM: Each motor has its own action instead of a general motor action with motor number. Replaces **rob_sim**.
hy_grep	Replaces **lsk_grep** for the modified simulator.
lskmodhy.h	Replaces **lsk_mod.h** for the modified simulator.

3.3.12 Package evo

EVOlution. The package contains the genetic operators, functions for calculating a generation, for processing an optimization job and the job list and for reading the evolution parameters (EVO file). In addition, the menu for the administration of optimization jobs as well as display functions. Finally for HyGLEAM modules for the integration of local search methods in general and the two LHCs Rosenbrock method and COMPLEX algorithm in particular.

In addition to the **evo_anzg** module, the two modules **evo_gsw** and **evo** also contain routines for result display and result saving that fill the text buffer.

Modules:

go_gsw	Basic routines and initialization by the EVO file.	
go_rxo	Standard recombination and cross-over operators.	
go_pmut	Parameter mutations.	
go_amut	Mutations of actions resp. genes.	
go_smut	Segment mutations.	
rc_koppl	Coupling of the two LHCs Rosenbrock and Complex (third-party software from ITEM) for HyGLEAM.	
lsv_steu	Data elements of the (adaptive) LHC control and central LHC call.	
adapt_di	Adaptive LHC selection and parameterization as well as `allP` adaptation.	
evo	Processing of one generation with sequential evaluation.	} alter-
parSimEvo	Processing of one generation, parallel simulation/evaluation by external Simulation Services (ESS), currently without LSVs.	} na- } natives

evo_gsw	Contains routines shared by **evo** and **parSimEvo**. Developed from the original **evo** in order to be able to create a version of **evo** for parallel simulation and evaluation.
evo_steu	Editing the job list and an optimization job.
evo_anzg	Package-local routines for statistics, logging & display (text buffer).
evo_cmen	Evo/Opt menu of the TEXT_MODE version.

3.3.13 Package webIO

This package contains functions for reading and writing chromosome and result lists based on web or file IO for applications based on the external simulation services (ESS). The module **fileIO** is for testing purposes only. The modules **curlIO** and **fileIO** implement the same interface and are to be added alternatively.

Modules:

listIO	Functions for reading and writing chromosome and result lists based on dynamic buffers.
curlIO	Implementation of the functions from **listIO** by HTTP get and post functions using libcurl.
fileIO	Implementation of the functions from **listIO** by File-IO for test purposes.

3.3.14 Package men

MENus. The package contains the package initialization, load functions of the init files and display routines (text buffer) for the menu functions of the TEXT_MODE version.

Modules:

g_cmen	All menus of the TEXT_MODE version , as far as they are not covered by the packages **chedi**, **bew**, **aufg**, **simu** and **evo**.
men_gsw	Basic and display routines for the menu functions (text buffer).
f_load	General package initialization and load functions of the initialization files. The load functions store their results in the text buffer.

3.3.15 Package parPop

Functions for parallel execution of an optimization job based on the island model.

Modules:

parPop	Routines for the initializer and the epochs.

3.4 Cross Reference Tool gxref

The **gxref** tool is a shell script located in the **linux** directory for searching a string in the source and header files of the GLEAM software system. It relies on the directory structure described in 2.4 and searches in the sources directory and its sub directories as well as in the application directories **gleam_ae**, **hy_gleam_a** and **hy_gleam_opalv** in the **linux** folder.

Call: **gxref <dir> <search-string>**

The script searchs for the **<search-string>** in (Hy)GLEAM source files starting at directory **<dir>**. The directory **<dir>** must be or contain the **sources** directory. A "." indicates the current working directory.

4 Literature

[1] W. Jakob: *HyGLEAM - Hybrid General Purpose Evolutionary Algorithm and Method: User Manual.* Technical Paper, V1.0, IAI, 2021.
See `HyGLEAM-Manual_V1.0.pdf`

[2] W. Jakob: *Applications Included in GLEAM and Integration of New Ones.* Technical Paper, V1.0, IAI, 2021.
See `GLEAM-Applications_Docu_V1.0.pdf`

[3] W. Jakob: *MOD File Documentation*. Technical Paper, V1.3, IAI, 2020.
See `MOD-File-Docu_V1.3.pdf`

[4] W. Jakob: *EVO File Documentation*. Technical Paper, V1.0, IAI, 2020.
See `EVO-File-Docu_V1.0.pdf`