

Operation of K2

Operators' Manual

Version 2.4

The K2 system was prepared by Effice Kft.
e-mail: effice@effice.hu

For internal use only, by the Budapest Stock Exchange and its members

© Copyright Effice Mérnöki Iroda Kft. 1999, 2001

This document may not be stored, either partially or in full, on any information system, it may not be reproduced or forwarded in any format or for any purposes on any electronic, photographic, photostatic, or magnetic devices, or in any other manner, without the prior consent of Effice Mérnöki Iroda Kft. (2083 Solymár, Toldi u. 21., Tel.: 429-0028).

Version 1.1 MMTS I. 1999

Version 2.2 MMTS II. 2001

Version 2.3 BSE review, September 2002

Version 2.4 BSE review, September 2004

Contents

1. INTRODUCTION OF THE K2 SYSTEM	4
1.1. THE FUNCTIONS OF THE K2 SYSTEM	4
1.2. STRUCTURE OF K2	4
1.3. MAJOR PARTS OF THE SYSTEM	6
1.3.1. Shared memory.....	6
1.3.2. Shared memory tables	6
1.3.3. The PGW process.....	7
1.3.4. The IFSS process.....	8
1.3.5. The IFSC	11
2. TYPES OF BASIC OPERATIONAL TASKS.....	11
2.1. OPERATION OF THE LINUX OPERATING SYSTEM	12
2.2. STARTING THE K2 SYSTEM	12
2.3. SHUTTING DOWN THE K2 SYSTEM.....	12
2.4. RESTARTING THE K2 SYSTEM	13
2.5. IFSC RECOVERY	13
3. SETTING THE RUNNING PARAMETERS.....	13
3.1. ENVIRONMENTAL VARIABLES NEEDED FOR THE RUNNING OF PGW.....	14
3.2. SETTING THE RUNNING PARAMETERS OF THE PGW	14
3.2.1. Default parameters.....	14
3.2.2. Configuration file	14
3.2.3. Command line parameters	14
3.3. THE ENVIRONMENTAL VARIABLES NEEDED FOR THE RUNNING OF IFSS.....	15
3.4. SETTING THE RUNNING PARAMETERS OF THE IFSS	16
3.4.1. Default parameters.....	16
3.4.2. Configuration file	16
3.4.3. Command line parameters	16
3.5. REGISTERING THE IFSS USERS AND THEIR AUTHORISATIONS.....	16
3.6. CONFIGURABILITY OF THE K2 SYSTEM	17
3.6.1. The configuration file uploaded upon the installation of K2.....	17
3.6.2. Configuration parameters not changeable by K2 users.....	17
3.6.3. The LINUX/UNIX user ID used by the BSE and the related authorisations	17
4. TRACKING THE OPERATION OF THE K2 SYSTEM - DEBUGGING	17
4.1. HOW CAN THE PGW AND THE IFSS PROCESSES BE MONITORED?	17
4.2. THE CONTENTS OF THE RECOVERY LOG FILE	18
4.3. WRITING THE CONTENTS OF THE SHARED MEMORY	20
4.4. MONITORING THE STATUS OF THE K2 SYSTEM.....	20
4.5. REGISTERING THE RESULTS OF MONITORING.....	22
4.6. VIEWING THE RESOURCES USED BY THE PGW	23
4.7. RELEASE OF RESOURCES AFTER THE ERRONEOUS LOGOUT OF THE PGW OR THE IFSS	23
4.8. MODIFICATION OF THE K2 USER PASSWORD	23
5. OPERATION OF THE DEMO AND TEST PROGRAMS DEMONSTRATING THE CO- OPERATION OF K2 AND THE IFSC.....	23
5.1. GET_TABLE	23
5.2. GET_OB	24
5.3. OB_WATCH	25
5.4. SEND_ORDER	25
5.5. GEN_ORDER	26
5.6. DEMO.....	26
5.7. DATA FORMAT TRANSFORMING AWK PROGRAMS.....	27
5.7.1. get_ob.awk	27
5.7.2. get_table.awk	27
5.7.3. pobo.awk	27
5.8. IFSTEST: TESTING USER PROGRAMS.....	28

Contents

6. APPENDIX A: ERROR MESSAGES.....	28
---	-----------

1. Introduction of the K2 system

1.1. The functions of the K2 system

K2 (a connection server) enables a high-capacity, real-time connection between the MMTS¹ trading system and the related user interfaces.

K2 consists of two similar systems: the K2_a and K2_d sub-systems. The former ensures the connection with the cash trading system (MMTS I.), while the latter maintains the connection with the futures and options system (MMTS II.). The structure of the two systems is completely identical, the only differences show up in the fields of the memory tables and the handling of messages (ASN.1). Therefore, hereinafter the expression “K2” will refer to *either* of the two sub-systems if there is *no difference* between them in terms of the specific characteristic being discussed.

Besides the use of the broker Trader Workplaces the K2 enables direct real-time data flow between MMTS and the connected system. Thus, for example, the broker’s system is immediately notified of all orders and transactions in MMTS, and the pre-orders collected and checked in the broker’s system are also sent to the trading system instantly. As K2 was structured in such a way that its speed and performance parameters are identical to those of the MMTS central system, it also enables users to carry out online trading functions through the broker’s system in place of or in addition to the trading workstations. When K2 is used, it is still possible to continue the use of traditional workstations, but the broker’s system may take their role either partially or fully, if necessary.

K2 is a highly reliable, robust application which is able to perform its tasks with greater speed and reliability than the workstation-based solutions applied or planned so far.

Major areas of use in the case of K2:

- Real-time download and update of trading information
- Entry of prepared orders
- Cancellation and modification of orders
- Triggered entry, cancellation, modification of orders
- Automated trading
- Interfacing a broker firm’s own trading system to the MMTS trading system of the Exchange
- Creation of individual trading workstations
- Real-time processing of market information, trader support
- Alarm functions, intervention
- Trading supervisory and surveillance functions

1.2. Structure of K2

¹ Multi-Market Trading System – the trading system of the Budapest Stock Exchange (BSE) including the system which serves the cash market as well as the system for the futures and options markets.

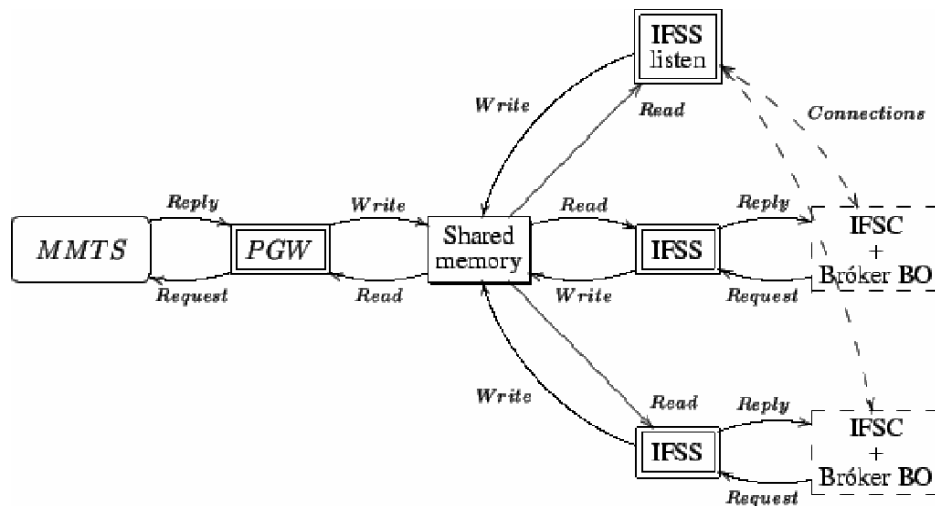


Figure 1.1: The structure of K2_a and K2_d

The structure of the system is shown in Figure 1.1.² The connection with the MMTS trading system is ensured by the PGW process while with the user systems is enabled by the IFSS process with the help of the IFSC library. The two processes keep in touch through shared memory and a semaphore.

PGW (polling gateway) - the program which replaces the MMTS workstation. From the perspective of MMTS, it operates as a workstation. However, the data are not shown in windows but are stored in the shared memory.

IFSS (interface socket server) - a server program which serves the user systems through a network connection by reading and writing the tables of the shared memories.

IFSC (interface socket client) - a client library which helps other systems connect to K2. The IFSC runs on the computer of the connecting system and provides an interface for the connection in the form of library functions.

K2 connects to the MMTS system as well as the user systems through network protocols. Towards MMTS, it uses a message transmitter called TSMR³ which requires TCP/IP and UDP/IP. The IFSS is standard TCP/IP socket-based.

Basically, two types of transactions flow through K2:

query The PGW process periodically inquires of the changes to the trading data and writes these into the shared memory. The IFSS process will send the changes in response to the queries it received.

order entry transaction During the entry of orders (sending of new orders, modification or cancellation of existing orders), the IFSS takes a request and inserts it into the shared memory. At end of the query cycle, the PGW will check if there is a new order entry record and send it to MMTS. It writes the response given by MMTS into the shared memory. The user may get information on the status of the *transaction* through appropriate *queries*.

² The figure is to be understood for both K2_a and K2_d separately.

³ Attention: the MMTS I. and MMTS II. systems – thus K2_a and K2_d, as well - use different versions of the TSMR!

1.3. Major parts of the system

1.3.1. Shared memory

The shared memory is an important element of K2. It ensures the data exchange between the processes.

The memory is handled (creation, connection etc.) by calling *System V IPC* functions. The synchronisation of memory access is ensured by a semaphore.

The shared memory is created by the PGW after logging on to MMTS. The size of the memory adjusts to the parameters¹ of the MMTS memory table size, since it is defined by the PGW based on the data received from MMTS. Having created the shared memory, the PGW will connect to it and upload it through TSMR queries. The IFSS will connect to the established shared memory. The programs (PGW, IFSS) will not cancel the shared memory. It can be cancelled with the `ripcc` tool which is based on the `ipccrm` operation system command. (see Chapter 10).

The PGW and IFSS processes connect to the shared memory on a permanent basis. Occasionally, of course, the `dump` and `MEM` tools may also connect to this. With the help of the semaphore, the connecting processes require exclusive access for the time the shared memory is being read and written.

1.3.2. Shared memory tables

The shared memory contains the following tables:

1. Markets
2. Instruments
3. Sector table
4. Boards (shares, government securities etc.)
5. Secboard table
6. Users
7. Firm table (brokerage firms)
8. Orders
9. Trade table
10. Negdeal table (negotiated deals, fixed transactions)
11. Order books²
12. Order Entry table³
13. Order Entry error message table⁴

¹ The sizes of the tables related to the entry of orders is defined by the configuration file.

² The handling of the table of order books is different from that of the other tables, and it also deviates significantly in the K2_a and K2_d systems.

³ This does not contain data from MMTS, but rather order entries made by the users in MMTS.

In addition to these tables which contain data related to trading, the shared memory also includes other data which are connected to the operation of the system. Both the PGW and the IFSS processes need these data (e.g. size of tables and addresses, highest order entry identifier, system identifier.)

The individual tables consist of identically-sized records and, except for the order books, they are organised based on a pre-defined key. The tracing of record changes is enabled by the change numbers. Except for the order books, a change number table containing a change number/record number pair is allocated to each table, organised by change numbers. If a new record is inserted, it will get the highest change number and a new line will be added to the change number table. If an existing record is updated, then *it* will get the highest change number, and the records following the former record of the change number table will move ahead by one unit (the original record will be deleted) and at the end, a new record will be inserted. Thus only the last change number must be remembered during queries, and in order to get the new or changed records during the next query, only those records must be read from the change number table which follow the last change number.

In K2_a, one record of the order book table contains the order book of one security (at present, up to the depth of 20 as defined by MMTS). It is not possible to monitor the order books of all securities as MMTS can not ensure this. For this reason, it is only possible to trace a pre-defined number of order books or just have a limited number of order books queried in a query cycle. However, in this case certain order books are not updated in all cycles. The user systems may define, through the IFSS, which security order books should be monitored by the PGW.

In K2_d, however, another solution had to be applied due to the different operation of MMTS II. The MMTS II system sends the complete order book, and for this reason there is a separate *order book* table in K2_d for the storing of data. The size of the order book table can be given with the *ordlistsize* parameter included in the configuration file. The *unfolded order book* “table” is actually a view of the order book table. (Market by Order)

The data of the *seboard table* can be traced not only by changes at the record level but also by changes at the field level. The fields of the *seboard table* can be divided into two groups: the first part is static and is only downloaded once after the start. Each record gets a serial number (*idx*) which grows continually, starting from zero. Static data may be downloaded by querying the records according to the IFSC physical order, which refers to the serial numbers. The rest of the fields may change during the day, possibly on a continuous basis. The PGW registers the field change numbers in a matrix with a size of *record* x *field*. If one of the fields of a record changes, then the corresponding element of the matrix will receive a new (highest) change number. The IFSC field change query will refer to these change numbers, and the response will contain those field code/value pairs whose change number is higher than that of the query.

1.3.3. The PGW process

The PGW process is a client program of the MMTS system. Similar to a workstation, it logs on and sends requests to the MMTS system from which it always receives a reply. Naturally, it handles these messages through the TSMR, therefore, the existence of the TSMR¹ – at least, the client side – is a basic requirement of the PGW. The PGW will store the trading data it handles in the shared memory (see Section 2) which is also available to other programs. The K2_a and K2_d systems use different PGW's which are called *pgw1* and *pgw2*. Their operation from an external point of view is identical.

There are two types of requests sent to the MMTS system:

queries Updating new or changed data generated since the previous query. One query is always relevant for one specific data type. (order, security, the order book of a given security . . .)

⁴ This table supplements the order entry table, it may not be queried separately.

¹ Consequently, the SYSSRV is also necessary!

order entry requests Entering new orders, or modifying or cancelling an existing one. The response only says whether the transaction was or wasn't successful, and it also contains the message intended for the user². If a new order is successfully entered, the message will contain the order number of the new order which the PGW will search for and re-write into the order entry record. If the message is "unsuccessful", it will insert the error message into the error message table. The entered order, along with its changes, may be received with a *query* appropriate for orders.

The PGW will connect to MMTS through the TSMR just like any other user. After a successful logon, the PGW will periodically send queries to MMTS. The first query after a successful logon queries the so-called *stats* array of MMTS, which includes the current momentary size of the MMTS memory tables along with other global MMTS data. Based on these data and the configuration file, the PGW creates the shared memory tables. Then, the basic records are queried of those tables in which the number of records is constant. (*market*, *instr*, *board*, *sector*, *secboard*, *firm* and *user*) The actual periodical operation only starts after this. During this periodical operation, the PGW will query the changes relevant for the following tables: *market*, *instr*, *board*, *firm*, *user* and *secboard*. These may only contain changes relevant for existing records and data for new records can never arrive here. Queries of the *order*, *negdeal* and *trade* tables are only made in the periodical phase. Both changes relevant for an existing record and new data records may derive from these queries. This is especially true for the *trade* table, where an extra query (*last_n_trades*) is included so that the PGW may get information on all transactions. At the end of each query cycle, the cycle time is supplemented with *sleep* for the period of time defined in the configuration file.

The query of order books takes place in a slightly different way. While the PGW of K2_a registers the series of *market_byorder* and *market_byprice* order books in a separate list, K2_d keeps the data in an order book table and creates the *market_byorder* and *market_byprice* order books from this. The PGW queries the order books which were set based on the momentary configuration.

The PGW also forwards requests to MMTS in respect of the entering, modification, and cancellation of orders. These are searched for in a shared memory table written by the IFSS. They are forwarded to MMTS and the returned status and possibly the order number or error message are registered in the table. The IFSS may send a *SIGUSR1 signal* to the PGW if a new record was written into the order entry table, thus accelerating the operation of the system. When PGW receives the *signal*, it suspends the current *sleep* and sends the new order entry record or records to MMTS. However, this is only possible when the value of the *sendsignal* parameter in the configuration file is not zero.

PGW protects the shared memory during reading and writing by a semaphore. This semaphore is first set to free status only at the beginning of the periodic operation. During the periodic operation, the semaphore is switched to closed status while queries and order requests are being sent. However, when the *freeseenabled* configuration parameter is set, the semaphore is released for TSMR message exchanges so that it would be possible to get information (e.g. on the status of the PGW process) from the shared memory even during delays derived from communication errors.

The MMTS II. system permits limited functions (only queries) for MMTS users between the first logon and the first successful password change. Therefore, in such cases, the PGW process sends a warning message. It is better to stop K2 and change the password with the *k2changepwd* script. For the purposes of security, the password must be changed in the MMTS I system when first logging onto the live system, in order to change the password received from the BSE.

1.3.4. The IFSS process

The IFSS (interface socket server) process is the K2 server for user systems. It responds to queries and order entry requests through the TCP/IP socket connection. It gets trading data from the shared

² The language of the message is appropriate for the user.

memory uploaded by the PGW, and it writes the order entry requests there, as well. K2_a and K2_d use different IFSS's (`ifss1` and `ifss2`) which look the same to an external user.

The operation of IFSS consists of the following phases:

1. Start of the server process waiting for connection
2. Making a connection – splitting off the server process
3. Exchange of messages
4. Disconnect

During start-up, it reads the configuration file along with the IFSS user data and the start-up switches. It opens the log files and copies itself into a separately running process. The program running in the original shell logs out. The separate process produces the socket where it remains on alert and then waits for the connections.

In the case of a connection, the client connects to the listening socket and the separately running IFSS process recognises the connection and splits a server process from itself to serve the given client. The server process connects to the shared memory and starts a cycle in which it reads the messages sent to the socket and gives the responses to them. First it expects a login message.

Each message has its own type. This can be found in the header at the beginning of the message. The type of the message defines the format of the data in the message. The data of the message are represented in a string form which is terminated with a zero. The type of the data is determined in advance.

The server process waits for a request message on its socket. If all characters of the message are received, it interprets the type of the message and submits it for the appropriate procedure. The given procedures interpret the data of the request and compile a response message which is sent through the socket. After this, they wait again for the next message.

In the case of queries, response messages (which include data for one record only) are put into a package in order to increase efficiency. The client opens up the package and gives the first message to the calling program. Until this program asks for the next record, the client does not send any query messages but it takes the next message in the package. If the package is emptied, then it will send a query message again. If the calling program processes one record after the other then this type of packaging increases the speed of multiple queries by one unit.

Types of messages:¹

Log out and log in: As a result of the login (`IFS_LOGIN_REQ`), the server process checks the name, password, and status of the user. If everything is OK, it handles any further messages in accordance with the user's authorisation. If user data are not appropriate or the user's status is suspended, then it will send an error message, disconnect the socket connection, and the operation will be finished. The response message contains the ID of the system start-up which helps to decide whether previous queries are still valid or not.

In the case of a logout (`IFS_LOGOUT_REQ`), the server process sends a response, breaks up the socket connection, and logs out.

Query: Query of the record changes of a given table (`IFS_SEQ_TABLE_REQ`) based on change numbers. This can be sent in respect of all tables, except for that of order books. In response, the server process will search for the record of the given table which traces the given change number and then it will prepare response messages of as many records as fit into a package.

¹ The precise data form is included in the `ifs_msg.h` file.

Then it sends this package as a response. Each message contains the change number of the given record with which the next record can be queried.

In the case of a securities table, it is possible to track changes at the field level, as well. The static fields of the securities table can be queried based on the physical order of the records (IFS_IDX_TABLE_REQ). The rest of the continuously changing fields can be traced with the field change request (IFS_FIELD_CHG_REQ). Similar to the query of record changes, both queries apply packages to accelerate the exchange of query messages.

Handling of order books: The order book belonging to the given security (more precisely to the secboard) first must be added to the monitoring list with the order book entry message (IFS_SET_OB_CONF_REQ). This results in the server process adding the given security (secboard) to the monitoring list if it had not happened before and if there is some free space.

The monitored securities (secboards) may be queried with the order book list message (IFS_OB_LIST_REQ). In response, the server process will send the identifiers of the monitored securities (secboards).

The order book belonging to the given security (secboard) may be queried with the IFS_ORDERBOOK_REQ message. In response, the server process will send the complete order book in a message. The maximum depth of the order book is determined by the MMTS system. At present, it is 20. The server process responds with messages of various length, depending on the given depth number.

Order entry request: In response to the order entry request (IFS_ORDERENTRY_REQ), the server process will insert a new record into the order entry table, giving it an individual order entry ID. At the same time, it also updates the change number. After this, it performs a simple check in terms of the format on the data content of the order and depending on the result, it sets the status of the order. Finally, it sends a response message which contains the ID of the order entry request. The data content of the message depends on whether the entry of a new order or the modification or cancellation of an existing one was requested.

In response to the status change request in respect of an order entry request (IFS_ORDERENTRY_STC_REQ), the server process checks the original and the requested status and if these are appropriate, it changes the status of the record with the given ID.

In both cases, the IFSS may send a *SIGUSR1 signal* to the PGW thus prompting it to perform a quicker (immediate) sending.

Disconnection: As a result of a logout, the server process disconnects the socket connection and logs out. If there is an error in the connection (either a communications error or the client program logs out), then the socket reading will return with an error. In this case, the server process also logs out.

Users

In K2 there are two types of users:

MMTS user When starting up the PGW process, it logs into the MMTS system under a pre-determined MMTS username. Inquiries and order entry transactions are initiated in the name of this MMTS user, and MMTS treats the authorisations accordingly.

IFS user When the user system connects to the K2/IFSS, then it defines an IFS username and a password. The IFSS treats any further requests based on this. Authorisations are allocated to each username. These usernames are only used by the IFSS and are not forwarded to the MMTS system.

Details in respect of the IFSS user authorisations are set out Section 3.5. (page 16).

1.3.5. The IFSC

The IFSC is composed of functions which are included in a programmer library. This library compiles request messages based on the parameters defined in the function calls and sends them to the IFSS. Then it takes the response messages and passes on the data of these in the buffer returned in the parameters of the functions. In the K2_a and K2_d sub-systems, it is the *same* IFSC that is at work! The functions store the following common information in the connection descriptor variables:

Change number of record changes The change number of record changes returned in the last query, by table.

Physical record index The query index of the physically last record, by table.

Field change number The last change number of the field change, by table.¹

Maximum number of order books The maximum number of acceptable order book trackings which is sent by the IFSS upon logging in.

Change number of order books The last change number of order books, by order book.

Receipt buffer The last message received.

Sending buffer The last request sent.

Tracing The tracing method.

Tracing file The descriptor of a tracing file.

The IFSC functions store the result of the query in the receipt buffer and the buffer pointer gives it back. The user program must copy the data before the following function call, since these will be overwritten during the receipt of the following message. When sending order entry requests, the IFSC function receives the data in a buffer and it copies them from there. The data in the buffer are character series with a fixed length, separated by zeros.² The IFSC utility functions help to separate data into program variables or to compile them from variables.

The separate description of the IFSC functions and utility functions are included in the “K2/ifsc User Manual”. The following declaration files should be referred to in the user programs:

ifsapi.h Declaration of the IFSC functions.

ifsdefs.h Definition of the IFSC constants.

ifsutil.h Declaration of the IFSC utility functions.

fields.h Field codes of the secboard table.

The declaration header files were written in the C language. They can be transformed into PASCAL using `c2p.awk`.

On a Unix platform, user programs must be linked with the `libifs.sl` (`libifs.so`) shared library. On a Windows NT platform, the `ifsc.dll` should be used. Some translators even require the `ifs.lib` static library which fulfils the functions given in the `ifs.dll`. The `ifs.lib` must be generated with the tools which are available in the development environment of the given translator.

2. Types of basic operational tasks

¹ At present, this kind of query can only be made for the securities (secboard) table.

² The data may also contain zero characters!

2.1. Operation of the Linux operating system

We will not go into the details of the operational tasks of Linux in this Manual. In respect of the operation of this system, the BSE requires continuous operation, the periodical saving and/or deletion of logging areas, as well as a backup copy from the complete system every other week, but definitely after the distribution of the software.

2.2. Starting the K2 system

The PGW only works when it can maintain connection with MMTS (i.e. it must be booted every morning and shut down in the afternoon at the end of trading). K2 must be booted with empty data after each start of MMTS as all previous data will lapse.

The “k2start 1” or “k2start 2” commands launch the PGW and IFSS processes.

K2 may be started or restarted with the following commands:

```
k2start 1 [options]
```

or

```
k2start 2 [options]
```

After booting, if the password was not given in the command line, the PGW will ask for it. If no foreground operation was specified in the PGW, then after requesting the password, the parent program will undock a background process and finish its operation. If foreground operation was specified, then the `k2start` will *not* start the IFSS process!

Starting process:

The PGW booting creates the semaphore, then the PGW logs into MMTS, creates the shared memory, and uploads it through queries. Depending on the `watchpgwenabled` configuration parameter, the PGW is separated into two processes. The child process will afterwards periodically query the changes and forward the order entry requests, while the parent process monitors the state of the child process.

IFSS booting: 5 seconds after booting the PGW, the `k2start` will also start the IFSS process. As soon as the PGW releases the semaphore, the IFSS will be able to serve the clients.

2.3. Shutting down the K2 system

The PGW and the IFSS may be stopped with the `k2stop 1` or `k2stop 2` commands:

```
k2stop 1
```

or

```
k2stop 2
```

The shutdown process:

The `k2stop` script will search for the appropriate PGW and IFSS process with the `ps` Unix command and send them a `SIGTERM` signal. After receiving the PGW `SIGTERM` signal, it logs out

of MMTS and terminates the process. After the processes stops, it deletes the shared memory as well as the semaphore.

The PGW process may be stopped individually with the Unix `kill` command. If the PGW stops, it will not delete the shared memory and the semaphore, the user systems will still have the opportunity to read them. The shared memory and the semaphore may be deleted with the `ripc` command.

2.4. Restarting the K2 system

If the PGW stops for any reason (e.g. it is definitively separated from the MMTS system), the whole K2 system must be rebooted. The restart of the PGW is supported by writing it into the recovery logfile where the order entry requests arriving from the user system are stored. During restart, the PGW also creates the semaphore and the shared memory and uploads them with data, but the order entry table is uploaded from the recovery log file. If the logfile is damaged, then it is possible to give the identifier of the following order entry record.

The IFSS process is for making the connections only, it does not have any memory. Rebooting it is the same as a normal start. The server processes of the IFSS connect to the prepared and uploaded shared memory and store the login data of the user. A new process is generated upon each connection.

2.5. IFSC recovery

If a user system connection (IFSC connection) fails, then it is the user system which must initiate the connection again. After the new connection, the user system will receive all data during the queries again. However, using the following method, it will have the opportunity to omit the queries which preceded the disconnection:

1. When the connection is successful (`ifsc_connect` function call), the user system will receive the K2 start identifier. If this identifier is the same as the one received during the former connection, then the same PGW is running on the shared memory. Then it will be possible to set the change numbers received during the query which preceded the failure:
2. The user system has the opportunity to set the change number used during any query of any table in the IFSC memory with the `ifsc_set_get_seq` function. The current change number belonging to the given table was given to the inquirer during the previous query. During the next query (`ifsc_get_next_record`), the IFSC queries the next record in respect of the given table with the given change number.

NOTE! If the K2 system is restarted, its internal boot identifier changes, thus using the old change numbers (i.e. the ones given before the restart) on the client-side may result in the *loss of data* (the client will not receive all data). Therefore, we recommend that the client's side be prepared for this and if the K2 internal boot identifier changes, then the change numbers on the client-side must be zeroed.

3. Setting the running parameters

The setting of running parameters is generally possible at three levels in the case of both the PGW and the IFSS. These are the following:

1. The level of the default values set by the program (translated in the program)

2. The level of the values in the configuration file
3. The level of the values given in the start-up command line

The higher the above listed number, the higher the priority. This means that after start-up, the program will give a default value to its parameters, then it will read its configuration file and overwrite the default values with the ones it found in the configuration file. The command line is processed³ after this, and during this process the given values are overwritten again. The program will fulfil its functions only subsequent to this.

3.1. *Environmental variables needed for the running of PGW*

The setting of the appropriate environmental variables necessary for the running of the PGW was completed upon installation.

After the system installation of K2, the `setup.sh` and the `aststools.sh` scripts are added to the K2 user `.profile` portfolio used for starting the system. When this is finished, no other manual setting is necessary. `aststools.sh` contains the settings of the `aststools` components and the path and environmental variables generally needed to access the MMTS environment (TSMR, SYSSRV, AMP). `setup.sh` checks whether the settings to be performed by `aststools.sh` exist, and upon logging in, it sets the path and environmental variables necessary for the PGW and the IFSS before the programs are started.

3.2. *Setting the running parameters of the PGW*

3.2.1. Default parameters

The default values of the PGW running parameters are included in the program source code itself.

3.2.2. Configuration file

The program searches for the configuration file (its location and name) in the appropriate environmental variable. If there are no such variables, then it will try to open a `pgw.cfg` file in the start-up library.

3.2.3. Command line parameters

The PGW command line parameters may be as follows:

-b<maxbadreplymsg> The maximum number of order entry error messages which can be stored (the length of the *badreplymsg* table). This overwrites the `maxbadreplymsg` parameter.

³ Only certain parameters can be given in the command line.

- f The PGW will run in the foreground. This overwrites the `foreground` parameter.
- F<firstorderentry> The PGW will add an identifier to the order entries (it will number them). The numbering will start with the number given with the parameter. This overwrites the `firstorderentry` parameter.
- m<maxorderentry> The maximum number of orders which can be entered (the length of the `orderentry` table). This overwrites the `maxorderentry` parameter.
- N<numoforderbook> The maximum number of order books which can be accepted for monitoring. Separately in the two types of lists. This overwrites the `numoforderbook` parameter.
- S The PGW will not perform the restart logfile checking. (But it will read it!) This overwrites the `skiplogchk` parameter.
- p<password> The password of the MMTS user. If it is not given, the PGW will ask for it after start-up. This overwrites the `password` parameter.
- P In this case, the PGW acts as a simple password changing device. It asks for the user's password, logs into MMTS, asks for a new password from the user and repeats the request for checking purposes. After this, it will modify the password of the user in MMTS and log out. The required username may be given with the -u<username> switch, and if the switch is absent, the password of the user given in the configuration file will be changed. *In this case, the PGW will not perform any data queries.* This overwrites the `chgpwd` parameter.
- r<orderratio> The ratio of the MMTS and PGW order tables. The PGW will receive the length of the order book from MMTS and by dividing it by the **orderratio**, it will get the length of its own table. This is because the PGW will only receive its own (own company's) orders and this is significantly less than the number of orders which can be entered into MMTS. This overwrites the `orderratio` parameter.
- t<seconds> The query cycle time of the PGW in seconds. Only a whole value may be given here. This overwrites the `cycletime` parameter.
- u<username> The PGW will log onto MMTS with this username. This overwrites the `username` parameter.
- v The PGW will write the data received from MMTS onto the default output. It writes down all data; this is only worth using it for the purposes of checking. This overwrites the `verbose` parameter.

3.3. The environmental variables needed for the running of IFSS

The setting of the appropriate environmental variables necessary for the running of the IFSS was completed upon installation.

After the system installation of K2, the `setup.sh` and the `aststools.sh` scripts are added to the K2 user .profile portfolio used for the starting of the system. When this is finished, no other manual setting is necessary. `aststools.sh` contains the settings of the `aststools` components and the path and environmental variables generally needed to access the MMTS environment (TSMR, SYSSRV, AMP). `setup.sh` checks whether the settings to be performed by the `aststools.sh` exist, and upon logging in, it sets the path and environmental variables necessary for the PGW and the IFSS before the programs are started.

3.4. Setting the running parameters of the IFSS

3.4.1. Default parameters

The default values of the parameters are included in the program source code.

3.4.2. Configuration file

The program searches for the configuration file (its location and name) in the appropriate environmental variable. If there are no such variables, then it will try to open a `pgw.cfg` file in the start-up library.

3.4.3. Command line parameters

-f The IFSS runs in the foreground. This overwrites the `foreground` parameter.

-s <service> A service name where the IFSS waits for the connections. This overwrites the `ifssservice` parameter.

3.5. Registering IFSS users and their authorisations

IFS user When the user system creates a connection with the K2/IFSS, it will define an IFS username and password. The IFSS will treat further requests based on this. Authorisations are allocated to each username. These usernames are only used by the IFSS, and are not forwarded to the MMTS system.

The IFSS users may have different authorisations which the IFSS will check upon making queries (i.e. the user may only use those messages on K2 in respect of which it has authorisation). The database of the user authorisations can be found in the `IFSS.uaf` file. The following authorisations may be set:

query Authorisation to query data.

entry Entry, modification and cancellation of orders

confirm Confirmation of entered orders in K2

config Configuration of order book list

bypass The orders entered will automatically be confirmed

admin Not in use at the moment

The user authorisation file (`uaf`) contains the data of the IFSS user accounts. Sample settings may be found in the `cfg/IFSS.uaf` file. One line contains one user's data separated by a colon:

username IFSS username.

password User password.

status The status of the user. It may be active (a) or suspended (s).

privileges Authorisations are separated by a comma.

Example for the contents of the .uaf user data file:

```
#
user1:password:a:entry,confirm,query,config,admin,bypass
user2:password:a:query
```

Note: The Remote Trader may change the IFSS.uaf file on the K2 computer it purchased without restrictions.

3.6. Configurability of the K2 system

3.6.1. The configuration file uploaded upon the installation of K2

When installing the K2 system, changing the initial contents of the configuration files written by the BSE is not recommended. The structure of the K2a and K2d configuration files connecting to the cash and the derivative systems is identical.

3.6.2. Configuration parameters not changeable by K2 users

The K2 user may not change the value of certain parameters of the configuration files because these are protected by the licence file. This means that if the user changes the values of the above parameters given upon the installation, the K2 program will not start.

3.6.3. The LINUX/UNIX user ID used by the BSE and the related authorisations

During installation, BSE personnel will set a user on the K2 server (Linux or Unix). The related password is only known by BSE personnel. This user is needed for the purposes of software upgrades, monitoring, and checking. According to the contract, the Remote Trader cannot limit the authorisations of the user used by the BSE and cannot change its password.

4. Tracking the operation of the K2 system - Debugging

4.1. How can the PGW and the IFSS processes be monitored?

The PGW writes two log files:

Event log It writes error and tracing messages into the pgw1.log and pgw2.log files. The explanation of the messages is set out in Appendix A.

Restart log The format of the file name is `YYYYMMDD_1.rlg` and `YYYYMMDD_2.rlg` where `YYYY` indicates the year, `MMM` is the name of the month, `DD` is the date. The order entry data are inserted here, namely the order entry and the order entry status change. During its restart, the PGW reads this file to restore the order entry records.

The location of both log files can be specified in the configuration file⁵ (`log_path` and `rlog_path`). The event log is a log text file which can be read in the usual way. For interpreting the contents of the restart log, a tool called `rldump1` or `rldump2` is available (see Section 10).

When the IFSC library is used, the `lastquerytime` and `pgwstate` fields of the shared memory will give information on the status of the PGW process. These can be gotten by the `ifsc_get_systime` function. The PGW process will write the current time into the `lastquerytime` field at the end of each query cycle. Through this, we can get to know when the last query and order entry responses were processed. The PGW process divides into two after downloading the static data when the `watchpgwenbled` configuration parameter is given. The child process will continue the queries and keep sending the order requests while the parent process only monitors the status of the child process with the `wait` system procedure. If the child process logs out, then it enters the `PGWTERMINATED` status into the `PgwState` field. (Until division, it is under `PGWINITED` status, while later until the logout it is in `PGWRUNNING` status.)

The log written by the IFSS is the separate event log:

Event log This writes the error and tracing messages into the `ifss1.log` and `ifss2.log` files. The explanation of the messages is set out in Appendix A.

Rlg file-ok For the purposes of recovery in the case of the PGW, the aforementioned logs are available. The file name format is `YYYYMMDD_1.rlg` and `YYYYMMDD_2.rlg`, where `YYYY` is the year, `MMM` is the name of the month, `DD` is the date. The order entry data are inserted here. When the PGW is restarted, it reads this file to restore the order entry records.

4.2. The contents of the recovery log file

With the `rldump1 <filename>` or `rldump2 <filename>` commands, the content of the `<filename>` restart log file may be written onto the normal output in an ASCII format. The restart log file includes the order entry requests and the status changes of those requests with a timestamp. Its format is as follows:¹:

Table C.1: Order entry record – MMTS1

Name	Length
Idx	6
Timestamp	20
orderid	6
Time	6
OrdNo	6
TrdAccId	12
BuySell	1
OrderMethodSet	32
BoardId	4
SecId	12

⁵ `PGW1CONFIG` or `PGW2CONFIG` point to this.

¹ There are separators between the individual fields: a "-" between the `Idx` and the timestamp, after the timestamp a "-" ... , and between the rest of them a "|"

Price	10
Yield	10
Quantity	10
Hidden	10
BrokerRef	40
ExpDate	8
ExpTime	6
TriggerPrice	10
SettleDate	8
MinFillQty	10
OUserIdx	6
FirmIdx	6
OrderDate	8
Status	1
Type	1
MsgIdx	6

Table C.2: Order entry record – MMTS2

Name	Length
orderid	4
time	4
OrdNo	22
OrdNoSpeedIdx	4
TrdAccId	13
BuySell	1
OrderType	1
Duration	1
PurgeOnLogoff	1
AllowSoftQtyLimit	1
AllowSoftPriceLimit	1
PositionType	1
IsPrivate	1
BoardId	5
SecId	21
Price	12
Yield	12
Quantity	4
VisibleQuantity	4
BrokerRef	41
ExpTime	8
TriggerPrice	12
TradeRef	31
MinFillQty	4
OpCode	1
PopCode	1
MultiLegOrdNo	21
MultiLegSpeedIdx	4
InstrId	5
UserId	13
FirmId	13
OrderDate	8
MarketMaker	1
Status	1

Type	1
MsgIdx	4
InternalRef	41

Table C.3: Order entry status change

Name	Length
Idx	6
Timestamp	20
Status	1

Table C.4: Order entry rejected status

Name	Length
Idx	6
Timestamp	20
Status	1
MsgIdx	6
Msg max	255

Example:

```
000000-1999-Sep-20 11:01:01-->005001|110101|000000|0013-0000001|...
000000-1999-Sep-20 11:01:02-->U|
000000-1999-Sep-20 11:01:02-->E|
```

The `rldump` gives a warning message if the content of the file does not resemble that of a restart log file or when we have the `rldump1` read the logfile from 2, or vice versa.

4.3. Writing out the contents of the shared memory

The `dump1` or `dump2` command writes the contents of the shared memory (per table) in a character format. The identifier of the shared memory is read from the file given in the `PGW1CONFIG` and `PGW2CONFIG` environmental variable. Upon its start, it connects to the shared memory and waits for the free signal of the semaphore. With the help of the semaphore, it protects the shared memory and puts the data into the normal output. The semaphore is reset afterwards.

4.4. Monitoring the status of the K2 system

System status monitoring - MEM

A monitoring process can be launched with the `mem1 [-fcn][-s<service|port>][-a<host|network>][-t<time>]` or the `mem2 [-fcn][-s<service|port>][-a<host|network>][-t<time>]` commands. This process will collect information on the status of the system by reading the shared memory and send it on the given network with a UDP or to a host.

Start-up switches

Switches of this command:

- f** The PGW will run in the foreground.
- c** Run continuously. Without this switch, it only collects information once and then logs out.
- n** It sends the collected information with the UDP. Without this switch, it writes only into the normal output.
- t<seconds>** The query cycle time in seconds.
- s<service|port>** This sends the collected information to the given port. If a service name is given, it will resolve the port into a number with the help of the running computer.
- a<host|network>** This will send the collected information to the given host or network. If a host name is given, then it will resolve the host name into an address with the help of the running computer.

Settings

In order to run the MEM, the following should be set:

PGW1CONFIG environmental variable This should refer to the K2_a PGW configuration file.

PGW2CONFIG environmental variable This should refer to the K2_d PGW configuration file.

PGW configuration file This should contain the identifier of the shared memory and the semaphore.
See Section 3.3.2.

service The **service** entry should be added to the service database of the operating system.

host The **host** entry should be added to the host database of the operating system.

Operation

After booting, it connects to the shared memory and waits for the free signal of the semaphore. Then it will collect and send the following information periodically by reading the shared memory:

- shared memory and semaphore identifiers
- sizes and saturation of memory tables
- stats info from the TE: statsversion, logtime, transaction num, rejection num
- PGW, IFSS and MEM process information: pid, ppid, ucomm, vshmsize, shmsize, start, utime, stime, process status

MEM output:

```
==> yellow 1
KEY: 0x70677731 SHMID: 420 SEMID: 570
-----Num
Max Recsize
market 3 3 48
sector 22 22 44
instr 11 11 48
board 9 9 88
secboard 260 260 752
firm 422 422 312
user 345 346 328
order 34 10000 248
trade 0 60000 224
negdeal 0 1000 172
orderentry 0 1000 188
```

```

badreplymsg 0 500 256
-----Last
orderno: 76
Last tradeno: 0
Last last_n tradeno: 0
Last negdealno: 76
Stats Version: 0x37f89a80
Log Time: 37839
Transactions: 33543
Rejections: 1
-----pid
ppid comm vshm shm starttime utime stime status
00003121 00000001 ifss 0 0 939047573 0 0 Sleep
00003113 00000001 pgw 4294 4294 939047561 0 0 Sleep
00003125 00000001 mem 4294 4294 939047585 0 0 Run
=====

```

We can find the name of the computer and the serial number of the message in the first row (the row that starts with ==>). If we send the messages through the network, then the customer can find out from the serial number whether there was a message¹ omitted.

4.5. Registering the results of monitoring

ear.pl – MEM monitoring and interpreter

`ear.pl` is a short Perl script which monitors the packages sent out by the MEM on the network. The packages received will be written into separate files by *hosts*. It monitors how saturated the tables are and writes an error message into the `ear.log` error file. An error message will be added to the file even if one of the PGW or IFSS processes is not running. An error message will be generated when no packages arrive from a *host* for a long time (provided that in the case of the operation of several K2's, these send the packages with the same frequency). When the `-m` command line option is set, it will automatically send an e-mail regarding the above messages to the given address.

For example, `ear.pl` may be controlled with the following switches:

- p<port>** The number of the port on which the `ear.pl` monitors the arriving packages.
- f** If this is given, `ear.pl` will run in the foreground.
- w<warning level>** The warning level as a percentage, in decimal form. (e.g. 0.80 means that a warning message will come when the tables are 80% full.)
- e<error level>** The error level as a percentage, as described in the previous point.
- t<timeout>** The package receipt timeout.
- m<E-mail address>** The e-mail address.
- h** Writing the above as a standard output.

¹ The transporter UDP protocol may lose a message.

4.6. Viewing the resources used by the PGW

Resource viewer script – sipc [1|2] The sipc command will show the resources used by the PGW.

4.7. Release of resources after the erroneous logout of the PGW or the IFSS

resource cleaning script – ripc [1|2] This will delete the system resources (the shared memory and the semaphore) which remained after the stopping of the PGW. During the logout of the PGW and the IFSS processes, this does not delete the system resources (the shared memory and the semaphore), thus it is possible to review any possible erroneous logouts. The ripc command removes these resources, which are needed before the following start-up.

4.8. Modification of the K2 user password

password changing script – k2changepwd [1|2] This launches the PGW process of K2_a or K2_d with a -P switch to change the MMTS password. Changing the password is mandatory when first entering the live MMTS trading system. The password changing script is used for this.

5. Operation of the demo and test programs demonstrating the co-operation of K2 and the IFSC

The following demo programs were made to demonstrate and test the use of the IFSC:

get_table Query of a table with the IFSC.

get_ob Query of the order book with the IFSC.

ob_watch Setting the monitoring of the order book with the IFSC.

send_order Sending an order request with the IFSC.

gen_order Preparation of order requests by monitoring the order book.

demo Short sample program table query, order entry, order entry record query and the parallel use of the IFSC utility functions.

5.1. get_table

The program launched with the gettable <-1|-2> <table>[-ctqig][-h<host>][-s<service>] [-u<user>][-p<password>] command queries the content of the given table using the IFSC library.

Start-up switches

The switches of the command:

- c** Run continuously, in the following cycle it only queries the changes.
- t** The tracing method.
- q** A change number method (default).
- i** An index method.
- g** A field change method.
- h<host>** The IFSS is on this host.
- s<service>** The IFSS waits for the clients which want to connect on the port defined with this.
- u<user>** An IFSS username which the client uses for logging in.
- p<password>** The password of the IFSS user.
- 1** Connect to the K2_a server. If **-2** is also given, then it connects to both servers.
- 2** Connect to the K2_d server. If **-1** is also given, then it connects to both servers. The service value belonging to the K2_d server can only be given with the IFS2SERVICE environmental variable.
- <table>** The name of a table which was queried, use “all” to query all tables.

The host, service, user and password parameters can be given with the IFSHOST, IFSSERVICE, IFSUSER and IFSPWD environmental variables.

Settings

To run the get_table, the following must be set:

service The service entry must be added to the service database (/etc/services) of the operating system.

host The host entry must be added to the host database (/etc/hosts) of the operating system.

Operation

After its start, it will log onto the IFSS server with the ifsc_connect function and call the ifsc_get_next_record function in a cyclic manner. It will transform the data it received into a character format and write them into the normal output.

5.2. get_ob

The program launched with the get_ob <secboardid>[-cmt][-h<host>][-s<service>][-u<user>][-p<password>] command queries the order book of the given secboard using the IFSC library.

Start-up switches

The switches of the command:

<secboardid> The identifier of the secboard.

-c Run continuously, it queries the complete order book every second.

-m An aggregate order book query. (the unfolded one is the default)

-t The tracing method.

For a description of the rest of the switches, see get_table

Whether it should connect to the K2_a or K2_d server can only be given with the `-s<service>` setting.

For the settings valid before the start-up, see `get_table`.

Operation

After its start, it will log onto the IFSS server with the `ifsc_connect` function and call the `ifsc_get_next_orderbook` function in a cyclic manner. It will transform the data it received into a character format and write them into the normal output.

5.3. *ob_watch*

The program launched with the `ob_watch <secboardid>|-l[-rmt] [-h<host>][
-s<service>] [-u<user>][-p<password>]` command sets the monitoring of the order book belonging to the given security (secboard) using the IFSC library.

Start-up switches

The switches of the command:

<secboardid> The identifier of the secboard.

-l Instead of a setting, it queries which securities are being monitored.

-r Instead of additions, it cancels the order book monitoring in respect of the given security.

-m Aggregate order book query. (the unfolded one is the default)

-t The tracing method.

For a description of the rest of the switches, see `get_table`

Whether it should connect to the K2_a or K2_d server can only be given with the `-s<service>` setting.

For the settings valid before the start-up, see `get_table`.

Operation

After its start, it will log onto the IFSS server with the `ifsc_connect` function and call the `ifsc_orderbook_conf` function, or if a query was requested, then the `ifsc_get_orderbook_list` function. It will transform the data it received into a character format and write them into the normal output.

5.4. *send_order*

The program launched with the `send_order <-1|-2> -e<E|A|W>[-f<orders>][-t][
-h<host>] [-s<service>][-u<user>][p<password>]` command sends order entry requests to the server using the IFSC library.

Start-up switches

The switches of the command:

-e<E|A|W> An order entry request type: E (entries), A (amends), W (withdrawns)

-f<orders> The name of the file where the order entry records can be found. If this is not given, then it will take them from the normal input.

-t The tracing method.

For a description of the rest of the switches, see `get_table`

For the settings before the start-up, again see `get_table`.

Format of the order entry record

The fields of the read order entry records are the same as those accepted by the `ifsc_orderentry`. They have a character-type format, the separating character is a '|'. One order entry record is one line.

Operation

After its start, it will log onto the IFSS server with the `ifsc_connect` function, then by transforming the read order entry record into an IFSC format, it will call the `ifsc_orderentry` function, and write the order number of the entered records.

5.5. *gen_order*

The program launched with the `gen_order <boardid> <secid> <B|S> <price> <qty> <trdacc> [-ct][-h<host>][s<service>][-u<user>][-p<password>]` command compiles order entry requests using the IFSC library. Monitoring the order book of the given security, it compiles contractable orders with prices better than the given offers, up to the quantity defined. The compiled order requests can be sent to the server with the `send_order`. Thus, the program actually performs an automated process during which it monitors the offers on the market and participates in trading.

Start-up switches

The switches of the command:

<boardid> A board identifier.

<secid> A security identifier.

<B|S> Buy or sell.

<price> Minimum price which reacts to offers.

<qty> The maximum quantity up to which the program compiles the orders.

<trdacc> Trading account identifier.

-c Continuous operation. If this is not given, it only compiles the orders with an order book query.

-t The tracing method.

For a description of the rest of the switches, see `get_table`.

At present, it only works with the K2_a system. For settings performed before booting, see `get_table`.

Operation

After its start, it will log onto the IFSS server with the `ifsc_connect` function and query the current order books with the `ifsc_orderbook_conf` function in a cyclic manner. After analysing the results, it compiles order entry records and writes them into the normal output.

5.6. *Demo*

The program launched with the `demo [-h<host>][-s<service>][-u<user>][-p<password>]` command queries the contents of the table specified in its source using the IFSC library.

Start-up switches

The switches of the command:

-h<host> The IFSS is on this host.

-s<service> The IFSS waits for the clients which want to connect on the port defined with this.

-u<user> An IFSS username that the client uses for logging in.

-p<password> The password of the IFSS user.

The `host`, `service`, `user` and `password` can be given with the `IFSHOST`, `IFSSERVICE`, `IFSUSER` and `IFSPWD` environmental variables. At present, it only works with the K2_a system.

Settings

To run the demo, the following must be set:

service The `service` entry must be added to the service database of the operating system.

host The `host` entry must be added to the host database of the operating system.

Operation

After its start, it will log onto the IFSS server with the `ifsc_connect` function and query the securities records with the `ifsc_get_next_record` function in a cyclic manner until it receives data. Then it will transform these into a character format and write them into the normal output. Subsequent to this, it compiles an order entry and sends it to K2 and with the `ifsc_get_next_record` function, it queries the order entry records again where the last order entry record is the record entered previously. Finally, with the `ifsc_disconnect` function, it performs a disconnection.

5.7. Data format transforming awk programs

get_ob.awk It shows the result of order book monitoring in a table format.

get_table.awk It shows the result of table queries in a table format.

pobo.awk It transforms the trading workstation's own order book file into a K2 order entry request format.

5.7.1. get_ob.awk

This interprets the lines read on its input in accordance with the order book (one line corresponds to a complete order book) and it writes table-format character lines into the normal output. The order book data are generated by the `get_ob` program.

5.7.2. get_table.awk

This interprets the lines read on its input as a record of the table query and puts in sequence, selects and tabulates the fields in accordance with the type of table, and writes them into the normal output. The table query records may be generated with the `get_table` program.

5.7.3. pobo.awk

This interprets the lines read on its input in accordance with the trading workstation's own order book format, and transforms these into an IFSC order entry format by resolving references and changing

the order of the fields. To resolve the references, during booting it compiles two reference arrays with the `.get_table secboard` and `get_table board` commands.

5.8. *ifstest: Testing user programs*

If the user program is linked to the `libifstest.sl/libifstest.so/ifstest.dll` test program library, it is possible to test the IFSC function calls without the K2 connection server as well as to try the user/programmer interface, and to check the use of the applied data structures.

`libifstest.sl/libifstest.so/ifstest.dll` contains the same IFSC functions (for the Unix, Linux and Windows NT systems) and works in a similar manner as the `libifsc.sl/libifsc.so/ifsc.dll` libraries. However, the `create` and `connect` calls do not build up a TCP/IP socket connection, but the returned connection descriptor points to the `ifscdata.dat` file opened in the start-up library of the tested program and serves the IFSC requests from this. The values of the tables and fields returned by the function calls are not derived from the K2 system connected to MMTS, but from the above data file. Thus, for testing purposes, the test data file which was given with the `ifstest` library should exist in the start-up library.

The `libifstest.sl/libifstest.so/ifstest.dll` can only serve decreased requests i.e. it is not suitable for testing the full range of `ifsc` function calls and the trials carried out with any of the parameters. The attached `apitest.c` demo program gives an example for the servable requests and based on the analysis of the source code, the framework of the test program or the trial application may also be worked out. During testing without K2, therefore, it is worth testing those `ifsc_calls` in the trial application with the parameters which can be found in the source program of `apitest.c`. In addition, `apitest.c` is suitable for taking the `ifscdata.dat` test data file, if we connect it to an operating K2 system and launch it with the `-c` option. However, this is not necessary as it has already been done at the BSE and the test data can be found in the `ifscdata1.dat` and `ifscdata2.dat` files given to `ifstest`.

There are separate test data files to test the `K2a` cash and `K2d` derivative connections: `ifscdata1.dat` and `ifscdata2.dat`, in accordance with `mmts1` and `mmts2`. We can only test using one of these at a time by copying the appropriate file into the `ifscdata.dat` file in the start-up library of the tested program.

6. Appendix A: Error messages

The PGW and the IFSS processes and the IFSC programmer's library may send the following error and tracing messages¹.

BADREPLY - Bad reply received when sent 0x%x

The MMTS system sent an `MSGTYPE_REPLY_BAD` reply to the PGW query. The number of the query message is in hexadecimal form.

BADREPMMSGFULL - Not enough space for new badreplymsg (buffer size is %d)!

There is no more space for order entry error messages.

BINDFAILED - %s - unable to bind address!

The monitoring address was not given successfully.

¹ In alphabetical order of their ID.

BRDCHGINVLEN - board %s change message length, supposed %d, received %d

The length of the response given for the MSGTYPE_BOARD_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

BRDINV - board %s doesn't exists

The board in the response given for the MSGTYPE_BOARD_CHANGE_REQ query by MMTS is unknown.

BRDINVLEN - board %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_BOARD_REQ query by MMTS is erroneous. Despite this, the message has been processed.

CANNOTFORK - cannot fork!

Error in new process creation.

CHECKSPACEERR - check_space() failed

The checking of the free space indicated an error – no free space for new data in the shared memory.

CLIENTLICEXCEED - The number of client %d is greater then in the license.

The number of clients given in the licence was superseded.

COMPILED - Compiled: %s %s

Information on the version of the program, the time of compilation, etc.

CONNACCEPETD - %s: Accepting a connection from %s port %u

The server accepted the connection from the given location.

CONNECTFAILED - IFS_COMERROR - %s - unable connect to %s!

The connect to the given server failed.

CONNECTTRACE - Connected to %s on port %u at %s

The IFSC connected to the given server.

DISCONNECT - disconnecting

IFSC is disconnecting.

DOWNLOADSUCC - All data has been downloaded.

Information message at the end of the first query period. This indicates that the static data and the changes were successfully queried.

EOFRECEIVED - Receiving end of file without disconnect message - connection with %s aborted!

The connection with the IFSC aborted.

FOPENERR - %s - %s

Error in opening a file.

FORCELOGGEDOFF - The MMTS user of pgw force logged off

The MMTS user of the PGW was forced to log off.

FRMCHGINVLEN - firm %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_FIRM_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

FRMINV - firm %s doesn't exist

The broker firm name included in the response given for the MSGTYPE_FIRM_CHANGE_REQ query by MMTS is unknown.

FRMINVLEN - firm %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_FIRM_REQ query by MMTS is erroneous. Despite this, the message has been processed.

INCHGINVLEN - instr %s change message length, supposed %d, received %d

The length of the response given for the MSGTYPE_INSTR_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

INSINV - instr %s doesn't exist

The instrument included in the response given for the MSGTYPE_INSTR_CHANGE_REQ query by MMTS is unknown.

INSINVLEN - instr %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_INSTR_REQ query by MMTS is erroneous. Despite this, the message has been processed.

INVALIDFIRM - The %s user's firm doesn't match the firm given in the config file

The PGW MMTS user's firm does not match the one given in the configuration file.

INVALIDLICENSE - The license given in the config file is invalid.

The license given in the configuration file is invalid.

INVLINE - %s – invalid line

The given line of the configuration file is invalid.

INVOACTION - Unknown watch action in watch config ask of SecBoard %s

The given order-monitoring method is not valid.

INVPRIVILEGE - User discarded, invalid privilege %s in record %d

The given IFSS user authorisation is invalid.

INVSTATUS - Invalid status (%c) received!

The requested status of the given order entry record is invalid.

INVSTATUSCHNG - Status (%c) of orderentry %d can not be changed!

The status of the given order entry record cannot be changed.

INVTRANSCODE - Unknown transition code %d!

The type of the given order entry request is invalid.

INVUSERNAME - User discarded, name %s in record %d

The given IFSS username is either too long or empty.

KILLFAILED - Kill failed - %s

Error during sending signals (kill call).

LINETOOLONG - %s – line too long

The line of the configuration file is too long.

LISTEN_FAILED - %s - unable to listen on socket!

The server cannot listen on the established socket.

LNTRDINVLEN - last_n_trades no %d message length, supposed %d, received %d

The length of the response given for the MSGTYPE_LAST_N_TRADES_REQ query by MMTS is erroneous. Despite this, the message has been processed.

LOGAPPEND - Log file (%s) opened for append.

Successful opening of log file for appending.

LOGONFAILED - Logon failed

The logon to the MMTS system failed.

LOGONINVREPLY - Logon failed, invalid reply from host

The logon to the MMTS system failed, the response from MMTS cannot be interpreted.

LOGONSUCCESS - Logon successful

The logon to the MMTS system was successful.

LOGOPENERR - Log file (%s) cannot be opened, so stderr will be used.

Error in opening log file. The error messages will be written onto *stderr*.

MBPADDED - Market by price of SecBoard %s added to watch.

The aggregate order book of the given security was added to those watched.

MBPLISTED - Market by price of SecBoard %s has been already watching!

The aggregate order book of the given security is already on the watch list.

MBPLISTFULL - Not free place to watch market by price of SecBoard %s.

There is no free space for watching the new aggregate order book. The watching of another security order book must be cancelled.

MBPNOTFOUND - Market by price of SecBoard %s can not found!

The aggregate order book of the given security is not monitored – its monitoring cannot be switched off.

MBPNOTLISTED - Market by price of SecBoard %s can not found!

No aggregate order book can be found for the given security – the monitoring of the order book must be set first.

MBPQUERY - %s market by price query; seq %d.

The IFSC has accepted the following aggregate order book data.

MBPREMOVED - Market by price of SecBoard %s removed from watch.

The aggregate order book of the given security will not be monitored in the future.

MEMHOSTNAMEERR - gethostname() failed

The *gethostname()* function could not give the name of the host.

MEMNONET - Network initialisation failed, stdout will be used

The initialisation of the network failed, *stdout* will be used as the output.

MEMSENDTOERR - sendto() error - %s

The sending of messages to the socket was unsuccessful.

MEMWRITEERR - write() error - %s

Writing onto *stdout* is unsuccessful.

MKTCHGINVLEN -market %s change message length, supposed %d, received %d

The length of the response given for the MSGTYPE_MARKET_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

MKTINV - market %s doesn't exists

The market included in the response given for the MSGTYPE_MARKET_CHANGE_REQ query is unknown.

MKTINVLEN - market %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_MARKET_REQ query by MMTS is erroneous. Despite this, the message has been processed.

MSGERROR - IFS_MSGERROR - The received len %d is differ from len %d in**header!**

The length of the message received differs from that given in the header.

MSGPROTVERDIFF - The IFS_PROTOCOLVERSION %d of ifss differs from version %d of ifsc.

The IFS_MSG_PROTOCOL_VERSION of the server is different from the version in the client library.

MSGTOOLONG - IFS_MSGERROR - The length of message %d is greater then length of buffer %d!

The message is too long.

NEGDINVLEN - negdeal no %d message length, supposed %d, received %d

MMTS did not give a message long enough for a query of fixed transactions.

NOCONFIGPRIV - no config privilege %s

The IFSC user does not have authorisation for the setting of order books.

NOCONFIRMPRIV - no confirm privilege %c

The IFSC user does not have authorisation for confirmation.

NOENTRYPRIV - no entry privilege %d.

The IFSC does not have authorisation for making entries.

NOHOSTBYNAME - IFS_COMERROR - gethostbyname() failed, host: <%s> !

The *gethostbyname()* returned with an error – the name of the server cannot be resolved.

NOLICENSE - The %s query has not been licensed.

There is no licence for the requested function.

NOPGWCONF - PGWCONFIG doesn't exist or too long

The PGWCONFIG environmental variable is not set, or its value is too long.

NOQUERYPRIV - no query privilege %d.

The IFSC user does not have authorisation for making queries.

NOSECBOARD - SecBoard %s can not found!

The given security cannot be found.

NOSERVBYNAME - IFS_COMERROR - getservbyname() failed, service: <%s> !

getservbyname() returned with an error – the name of the server port can not be resolved.

NOSOCKNAME - IFS_COMERROR - %s - unable to read socket address!

The IFSC was unable to read the socket address.

NOSUCHMSG - Unhandled message %d !

The message cannot be handled.

NOSUCHTABLE - Unhandled table %d.

The given table query cannot be answered.

NOTENOUGHMEM - malloc() failed at stats

There is not enough memory.

OBADDED - OrderBook of SecBoard %s added to watch

The unfolded order book of the given security was added to those watched.

OBINVLEN - byorder message length, supposed %d, received %d

The length of the response given for the MSGTYPE_MARKET_BYORDER_REQ query by MMTS is erroneous. Despite this, the message has been processed.

OBINVSECB – by order secboard %s doesn't exists

The secboard included in the response given for the MSGTYPE_MARKET_BYORDER_REQ query is unknown.

OBLISTED - OrderBook of SecBoard %s has been already watching!

The unfolded order book of the given security is already included in the watch list.

OBLISTFULL - Not free place to watch OrderBook of SecBoard %s

There is no free space to watch the new unfolded order book – the monitoring of another security order book must be cancelled.

OBNOTFOUND - OrderBook of SecBoard %s can not found!

The unfolded order book of the given security is not monitored – its monitoring cannot be switched off.

OBNOTLISTED - OrderBook of SecBoard %s can not found!

No unfolded order book can be found for the given security – the order book monitoring must be set first.

OBQUERY - %s orderbook query; seq %d.

The IFSC received the following unfolded order book data.

OBREMOVED - OrderBook of SecBoard %s removed from watch

The unfolded order book of the given security will no longer be monitored.

OEADDED - Orderentry added (orderid: %d trans_code %d)

The order entry request was successfully added.

OEINVREPLY - orderentry status remains unknown idx: %d

The response given by MMTS to the MSGTYPE_ORDER_REQ message cannot be interpreted.

OENOTCSTRING - IFS_OENOTCSTRING - The orderentry is not NULL terminated!

The order entry request is not a NULL-terminated C string.

OENOTEXISTS - Orderentry (orderid %d) can not found!

The given order entry cannot be found.

OESTATUSCHNG - Status %c of Orderentry (orderid %d) changed to %c

The status of the given order entry record was changed successfully.

OETABLEFULL - Not enough space for new orderentry (buffer size is %d)!

There is no more space for order entry requests.

OETOOLONG - IFS_OETOOLONG - The order_len %d is greather then len of the aviable buffer!

The order entry record is too long.

OETOOMANYMSG - more than two message received at orderentry

The response given for the MSGTYPE_ORDER_REQ message by MMTS contains more than two messages. The first one is relevant for the issue of the order entry while the second one is the regular null-terminated message.

OETYPEINV - Invalid orderentry type %c idx: %d

Erroneous order entry type. The order entry type can be:

- entry
- modification
- cancellation.

OPTARGREQ - Option -%c requires an argument

A value must belong to the switch given in the command line.

OPTUNKNOWN - Unrecognized option: - %c

Unrecognised command line switch.

ORDINVLEN - order no %d message length, supposed %d, received %d

The length of the response given for the MSGTYPE_ORDER_REQ query by MMTS is erroneous. Despite this, the message has been processed.

PARAMDECIMAL - %s : %d

Decimal configuration parameter.

PARAMDOUBLE - %s : %e

Double configuration parameter.

PARAMHEXNUM - %s : 0x%x

Hexadecimal configuration parameter.

PARAMSTRING - %s : %s

String configuration parameter.

PGWTERMINATED - The pgw process terminated!

The Pgw process has terminated.

PWDCHGATTEMPT - MMTS password change attempt.

MMTS password change attempt.

PWDCHGFAILURE - MMTS password change failure %s.

The MMTS password change attempt failed.

PWDCHGSUCCESS - MMTS password changed successfully.

MMTS password was changed successfully.

RECONOK - reconnect succeeded.

The reconnection was successful.

RECONSTART - reconnect starting.

The reconnection has started.

RECRESMBP - receive repsonse of market by price; status: %d

The IFSC received the following aggregate order book data.

RECRESMBPCONF - receive repsonse of market by price conf; status: %d

The IFSC received the following aggregate order book setting response.

RECRESMBPLIST - receive repsonse of market by price list; status: %d

The IFSC received the following aggregate order book list data.

RECRESPOB - receive repsonse of orderbook; status: %d

The IFSC received the following unfolded order book data.

RECRESPOBCONF - receive repsonse of orderbook conf; status: %d

The IFSC received the following unfolded order book setting response.

RECRESPOBLIST - receive repsonse of orderbook list; status: %d

The IFSC received the following unfolded order book list data.

RECRESPOE - receive repsonse of orderentry; status: %d

The IFSC received the following order entry response.

RECRESPTAB - receive repsonse of %d; status: %d

The IFSC received the following response.

RECVFAILED - IFS_COMERROR - %s - Error at recv!

Error while receiving the message.

RECVTRACE - recv >%s<

The IFSC received the following message.

REPLYBAD - Bad reply received!

BadReply (error) message received for the PGW process query.

RESTRICTEDLOGON - The pgw logged on with RESTRECTED mode - You must change your MMTS password!

The PGW can only log onto the MMTS system in restricted mode – the password must be changed!

RLDFILEOPENERR - The restart log file (%s) cannot be opened - %s

The *rldump* cannot open the restart log file.

RLDINVFILE - This file (%s) doesn't seem to be a restart log file!

According to the *rldump* the file is not a restart log file.

RLDINVRECTYPE - Invalid record type %d in record %d

The record type is unknown for the *rldump*.

RLOGDATAREAD - Couldn't read the data from the restart log file - %s

Data reading from the restart file was unsuccessful.

RLOGDATAWRT - Couldn't write the data into the restart log file - %s

Data writing into the restart file was unsuccessful.

RLOGHEADREAD - Couldn't read the header (index/type/length) from the restart log file - %s

The header reading from the restart log file was unsuccessful.

RLOGHEADWRT - Couldn't write the header (index/type/length) into the restart log file - %s

The header writing from the restart log file was unsuccessful.

RLOGINVIDX - Invalid index in the restart log file - recno: %d

Erroneous index in the restart log file. This order entry record index does not exist.

RLOGINVMSGIDX - Invalid message index in the restart log file - recno: %d

Invalid error message index in the restart log file.

RLOGINVRECTYP - Invalid record type read from the restart log - record: %d type: %d

Non-existent order entry record type. The following types are valid:

- order entry from the IFSS
- order entry status change
- MMTS rejected the order entry.

RLOGINVTIMESTAMP - The existing restart log file timestamp is different from the shared memory timestamp

The timestamp in the restart log file is different from the MMTS timestamp.

RLOGINVRLOGFILE - The MMTS type of existing restart log file is different from the file reader.

The MMTS type of the restart log file (I or II) is different from that of the file reader (MMTS type of the program).

RLOGNAMEFAILED - Restart log file name cannot be created

The restart log file name cannot be created.

RLOGNOEND - Cannot go to the end of the restart log file - %s

It is not possible to go to the end of the restart log file.

RLOGOPENFAILED - Restart log file name cannot be opened

The restart log file cannot be opened.

RLOGREPORT - %d record read from the restart log - number of orderentries %d

This message provides information on how many records were read from the restart log file.

RLOGREWDERR - Couldn't rewind the restart log file - %s

It is not possible to go to the beginning of the restart log file.

RLOGTIMEREADERR - Couldn't read the timestamp from the restart log file - %s

The timestamp could not be read from the restart log file.

RLOGTIMEWRTERR - Timestamp cannot be written to the restart log file - %s

The timestamp cannot be written into the restart log file.

RLOGTYPEREADERR - Couldn't read the system type from the restart log file - %s

The MMTS type cannot be read from the restart log file.

RLOGTIMEWRTERR - System type cannot be written to the restart log file - %s

The MMTS type cannot be written into the restart log file.

SBDFELDNOERR - secboard %s number of fields, counted %d, received %d

The actual number of fields in the MSGTYPE_SECBOARD_CHANGE_REP message received from MMTS is not in line with the number of fields given in the message.

SBDINV - secboard %s doesn't exists

The secboard received in the MSGTYPE_SECBOARD_CHANGE_REP message from MMTS is unknown.

SBDINVFIELD - secboard %s invalid field indicator %c

The field indicator in the MSGTYPE_SECBOARD_CHANGE_REP message from MMTS is invalid.

SBDINVLEN - secboard %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_SECBOARD_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

SCTINVLEN - sector %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_SECTOR_REQ query by MMTS is erroneous. Despite this, the message has been processed.

SEMCREATEFAILED - %s - create_sem() failed

The semaphore could not be created.

SEMDELETEFAILED - %s - delete_sem() failed

The semaphore could not be deleted.

SEMGETFAILED - %s - get_sem() failed

The value of the semaphore could not be queried.

SEMOPENFAILED - %s - open_sem() failed

The semaphore could not be opened.

SEMSETFAILED - %s - set_sem() failed

The semaphore could not be set.

SEMWAITFAILED %s - wait_sem() failed

Waiting for the semaphore returned an error.

SENDFailed - IFS_COMERROR - %s - Error at send (sent byte: %d)

Error during sending of a message.

SENDTRACE - send >%s<

The IFS sent the following message.

SEQHASHUPDERR - Cannot update seqhash, index: %d num: %d

The change number reference could not be completed.

SETSOCKOPTFAILED - %s - Error at setsockopt: REUSEADDR

The REUSEADDR socket parameter could not be set.

SHMATFAILED - %s - shmat() failed

The connection to the shared memory failed.

SHMCREATEFAILED - %s - create shmget() failed

The shared memory cannot be created.

SHMDETACH - Detached from shared memory

The disconnection from the shared memory was successful.

SHMDTFAILED - %s - shmdt() failed

The disconnection from the shared memory was unsuccessful.

SHMINFOFAILED - %s - shmctl() -info- failed

No information can be queried from the shared memory.

SHMEMSIZE - Shared memory size: %d bytes

The size of the shared memory.

SHMOPENFAILED - %s - open shmget() failed

The shared memory cannot be opened.

SHMREMOVEFAILED - %s - shmget() -remove- failed

The shared memory could not be deleted.

SHMWAIT - wait for shared memory, key: 0x%x

Waiting for shared memory.

SOCKETFAILED - IFS_COMERROR - %s - unable to create socket!

Error in creating a socket.

STARTSUCCESS - Successfully started

The PGW started successfully. It is running in the background and has created the semaphore. After this, it will log onto MMTS.

STATSINVLEN - stats message length, supposed %d, received %d

The length of the response given for the MSGTYPE_STATS_REQ query by MMTS is erroneous. Despite this, the message has been processed.

SYSIMEINVLEN - systime message length, supposed %d, received %d

Error in system time length.

SYSIMESYNCD - offset is %d seconds

The system time is not synchronized.

SYSIMESYNCF - query time is %e seconds

Query time.

TRACEFILEERR - IFS_TRACEFILEERR - trace file (%s) cannot be opened.

Error during opening the IFS tracing file (ifstrace.log).

TRACESET - trace set to %d

The IFS tracing method was set at the following value.

TRDINVLEN - trade no %d message length, supposed %d, received %d

The length of the response given for the MSGTYPE_TRADE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

TRDLOST - last_n_trades no %d is an old trade but cannot be found (last trdno %d)

The trade included in the response of MMTS to the MSGTYPE_LAST_N_TRADE_REQ query cannot be found.

TSMRDISCONNECT - Disconnected from the TE

The TSMR message link layer disconnected from the trading system.

TSMRERRMSG - %s

A TSMR message link layer error was detected.

TSMRLOGOFF - Logged off from TE

Logged off from the trading system.

UAFFULL - There is no room for record %d in UAF file (max %d)

There are no more free records for the IFS user data.

UNKNOWNMSG - Unknown TSMR message 0x%x

Unknown TSMR message (connection layer error).

USERINTERR - %s internal error.

Internal error in program.

USERINVPWD - %s password invalid.

Invalid IFSC user password.

USERLOGGEDIN - %s logged in.

The given IFSC user has logged in.

USERLOGOUT - logout.

The IFSC user has logged out.

USERNOFIRM - %s user doesn't have firm

No brokerage firm is allocated to user.

USERNOTACTIVE - %s isn't active.

The IFSC user is inactive.

USERNOTEXISTS - %s doesn't exists.

That IFSC user ID does not exist.

USERNOTLOGGEDIN - %s isn't logged in yet.

The IFSC user has not logged in yet.

USGDUMP - Usage: %s

dump utility program command line description.

USGIFD - Usage: %s [-f] [-t<cycle>]

ifd (interface database) utility program command line description.

USGIFSS - Usage: %s [-f]

IFSS command line description.

USGMEM - Usage: %s [-c] [-n] [-f] [-t<cycle>] [-s<service/port>] [-a<address/hostname>]

mem utility program command line description.

USGPGW - Usage: %s [-f] [-v] [-S] [-t<cycle>] [-u<username>] [-p<password>]

PGW command line description.

USGUNKNOWN - Unknown program (%s), no usage available.

There is no command line description of the program.

USRCHGINVLEN - user %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_USER_CHANGE_REQ query by MMTS is erroneous. Despite this, the message has been processed.

USRIDNOTUNIQ - Userid isn't unique, %s in record %d

The given IFSS username is not unique.

USRINV - user %s doesn't exists

The user included in the response given for the MSGTYPE_USER_CHANGE_REQ query by MMTS does not exist.

USRINVLEN - user %s message length, supposed %d, received %d

The length of the response given for the MSGTYPE_USER_REQ query by MMTS is erroneous. Despite this, the message has been processed.

WRONGPWD - User discarded, password %s in record %d

The given IFSS user password is either too long or is empty.

WRONGSTATUS - User discarded, invalid status %c in record %d

The given IFSS user status is an invalid value.