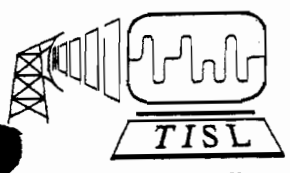
Netalyse
A Network Analysis Tool

Rene J.W. Laan
Roel J.T. Jonkman
Douglas Niehaus
Victor S. Frost

TISL Technical Report TISL-1660-11

November 1994



Telecommunications and Information Sciences Laboratory
The University of Kansas Center for Research, Inc.
2291 Irving Hill Drive Lawrence, Kansas 66045

Abstract

This report describes the design of a measurement analysis processing program called Netalyse. Netalyse is a generic tool which should be capable of handling almost any type of time sorted output of a measurement tool. After preprocessing the data and storing it in a binary internal format, the user can perform various calculations on the data and display it in diverse graphs. Netalyse consists of three main parts, namely Tcl/Tk, MatLab and database manager implemented in C.

Samenvatting

Dit verslag beschrijft het ontwerp van een data analyse programma genaamd Netalyse. Netalyse is een algemene tool die de tijdgesorteerde data geproduceerd door diverse meet-tools kan verwerken. Na een preprocessingslag waarna de data in intern binair formaat word opgeslagen, kan de gebruiker er diverse berekeningen op uit voeren en het vervolgens in grafisch formaat displayen. Netalyse bestaat uit drie delen, namelijk Tcl/Tk, Matlab en een database manager geschreven in C.

Acknowledgements

All the people who thought at first, Oh my God there is another bunch of Dutch guys coming over.
Darren "GUI-guy" "This is gonna be a spiffy GUI"
Victor "Making good progress??"
Doug "People with red hair should stick together"
Jane "Take care or i'll have you deported"
Steve "Can we try this?"
Mike "We return jack-didley do we?"
Khoi "1 2 3 4 test"
Joe "This bloody thing doesn't work!!"
Cameron "You're a stud"
Ben "No"
Vinai "I'm gonna kill you, you know"

To all the people at TISL who know the following line :
"Al sla je me dood, ik zou het niet weten"
("Even if you beat me to death i wouldn't know")

Contents

Chapter 1

Introduction

The Telecommunications and Informations Sciences Labratory of the University of Kansas was formed in 1983. The faculty consists of 13 professors, several PostDoc's, several PhD students and a lot of grad students. A main part of the research is done in the area of telecommuncations and datacommunications. The research is done both on software-level and hardware-level. One of the hottest topics at the laboratory right now is high-speed networking. The laboratory is involved in designing and developing a gateway which interfaces a local area high-speed network (LAN 155 Mbits/s) to a wide area high-speed network (WAN, 2.4 Gbits/s)

The laboratory is involved in one of the six testbeds for high-speed networking. (MAGIC, Multidimensional Applications and Gigabit Internetwork Consortium) The main goal of the laboratory within this testbed is to do research concerning performance of high-speed networks. To do the gathering and analysis of data, software is needed, specific software like this is not available. The gathering is mainly based on customized programs which interface directly with either the kernel or a device-driver. The analysis which was performed on the gathered data was mostly done within matlab, this implied specified knowledge of matlab and or several. There was clearly a need for a specific analysis-tool. This report describes the first and partly the second revision of Netalyse. Netalyse is an analysis-tool which has database- display- and various calculation-capabilities.

Chapter 2

The perfect program

To analyse the performance of the MAGIC network and to compare the models with reality, measurements need to be taken. Because of the high speed of the MAGIC network, the delays and the performance of the computers connected to the network are also important. This means that measurements need to be taken at different levels in the computer architecture. The levels at which measurement can be taken at the moment are :

- application level
- TCP/IP level
- AAL level (ATM Adaption layer)
- ATM level

The programs which take these measurements are not the subject of this thesis since they are mainly written by other programmers. An experiment is taken by starting all the relevant measurement programs and then starting the application of which the perfomance is being measured. The global setup of an experiment is illustrated by 2-1. The data generated by these measurement programs needs to be analysed. Ultimately this analysis program can perform the following functions :

- set up and run an experiment.
- store the data in a convenient way.
- analyse the data.
- display the results graphically.

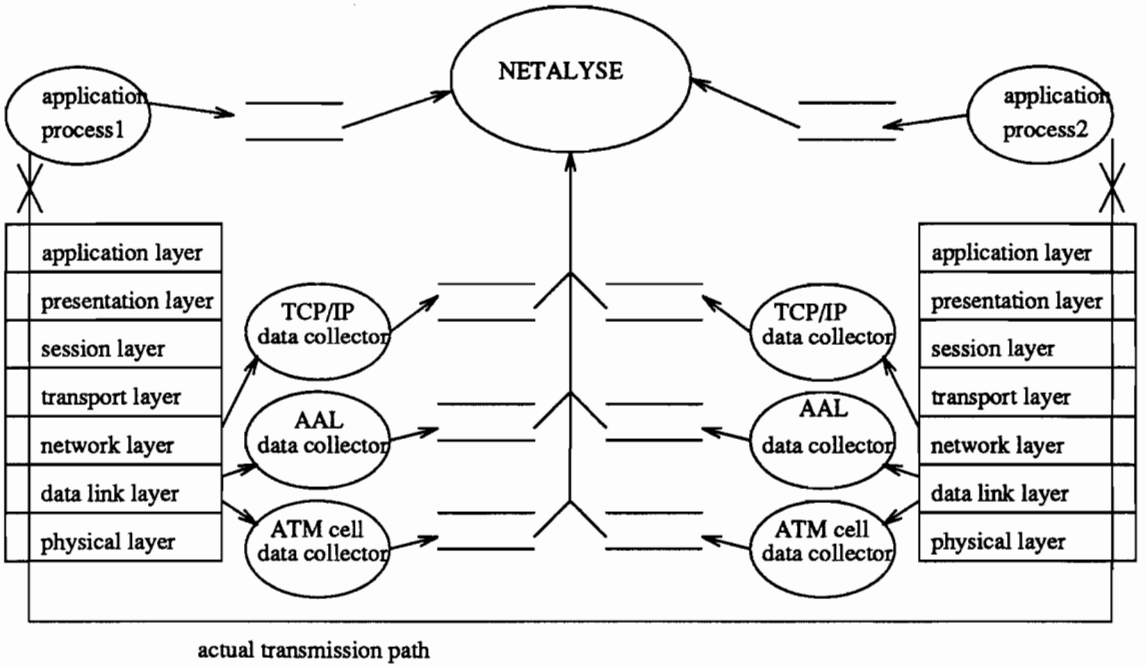


Figure 2-1: Setup of network measurements

2.1 Set up and run an experiment

The user indicates the computers involved, which measurement tools to use and which application to use. The parameters with which the experiment is taken will be stored together with the data files generated during the experiment. These parameters could be: which computers are involved, packet sizes, the application which is run, the parameters for this application etc.

Analysing the data

The data in the files generated during an experiment will be analysed by displaying it in graphs and by applying various mathematical functions to the data. These functions include auto- and crosscorrelation functions, as well as fourier transforms or ordinary additions and multiplications. Also the user wants to see which event on a certain level results in another event on another level. For instance: what happens on the TCP/IP level if the application sends a picture to another computer.

2.2 The specifications for the program

The main goal, for us, was to write a program that could display the data from the current measurement tools during the MAGIC meeting on the fifteenth of august 1994. Because of the limited time we had to limit the functionality to the following:

- Setting up an experiment and running it can be done using scripts, so there is no immediate need for a program that performs this function.
- The program has to be able to read the data files generated during an experiment. The files would not be grouped together as an experiment.
- The program has to be able to analyse the data and display it in graphs.

For the input files was specified

- The files might be big, to big to fit in memory at once.
- Every measurement-program generates it's own format because the type of the data depends on the level on which it was generated.
- Specifications for the format:
  - both ASCII - and binary files must be read.
  - binary files can be little-endian and big-endian.
  - ASCII files may contain multiple records per file. In this case the first field of the record will identify its type.
- New formats will be used for new data collection tools which will be developed later.
- Data from various files must be related because the data files are generated by several different processes and must be related for analysis.

The program will have a GUI (Graphical User Interface) so it will be easy to use.

2.3 The tools.

The application can be divided into four parts:

- processing datafiles
- a mathematical part for analysis
- a part which does the graphing
- a GUI

The GUI is implemented using TCL/TK, an interpreter with which GUI's can be created fairly easy. Also, writing a TCL/TK program was to become the standard way to create a GUI at TISL

For the mathematical and graphing part we had several options:

- create our own graphing utility with TCL/TK and write the calculation routines
- use the public domain program Xgraph and write the calculation routines
- use Matlab which can do both graphing and the calculations

The advantage of programming our own graphing utility is that it could be integrated perfectly in the rest of the program. On the other hand it would be a lot of work and not as flexible as Matlab.

The program Xgraph is a program that takes a file as input and displays that as a graph. The program is very limited and doesn't let you modify the graph. Besides that we would still have to program the mathematical functions. It would probably be the least desirable solution.

Matlab is a very complete and flexible program and can do both graphing and calculation. Matlab can be controlled by other programs. The disadvantage of Matlab is that it is a commercial product so everyone who wants to run the Netalyse program must buy matlab. Because Matlab is used in most university environments this does not have to be a great drawback. Another disadvantage is that it is hard to exercise the control sometimes needed. For instance the user can kill a graph without Matlab sending a signal to the Netalyse program.

Because development time was important and we wanted to be able to do many different calculations on the data we choose to use Matlab.

We decided to write the rest of the routines in the programming language C for the following reasons: The interfacing with both Matlab and Tcl/Tk can be done with C, C-compilers are available for every computer-architecture at TISL and we had experience with programming in C.

Chapter 3

Specifications and Analysis

The choices of Matlab and Tcl/Tk leave the following to be written in C:

- Reading the datafiles into memory
- converting the data to Matlab input
- Interface to Tcl/Tk and Matlab

Below the functionality is derived from the specifications. When the functionality we defined required new specifications then these are called derived specifications.

3.1 General Specifications

Specification :

The files might be big, too big to fit in memory at once.

Analysis :

Since the data files can be too big to fit in memory, the user has to be able to specify what part of the file he wants to read. The user will do this by specifying the start time and end time. After the specified data is read the user might want to read more data from the same file. Since the files might be huge it would be slow to reread all the data. It is faster to read only the extra information the user asked for. This can be realised by approaching the specified start- and endtime as the start and end of a window on the file. All data within the window is read. The window can be resized and moved within the file, if the user wants more or other data.

Derived specification :

A windowing system for file access is required.

All the input files have to contain timestamps and they must be sorted on time. Network measurements always result in time series because a network measurement is either a registration of events versus time or averages versus time. So all files will have time and will be sorted on time automatically.

Specification :

The user wants to relate data from various sources. The data from different sources is collected by different programs and stored in different files. So data from various files must be related for analysis.

Analysis :

The data from various files must be related so the user will have data in memory from several files. Because the user may want to read more data from the same file, the file must be kept open until the user does not need the data from that file anymore. These two requirements together imply that several files will be open at the same time.

Specification :

Every measurement-program generates it's own format.

Analysis :

The program has to be able to convert input in various formats into a standard format Netalyse can work with. The files can be in ASCII format or binary format. The ASCII format must be converted to numbers the computer can work with an binary files may need endian conversion.

Specification :

New formats will be used for new tools which will be developed later.

Analysis :

The type of input files the program will use is not fixed and may change in the future. Therefore the user has to be able to specifiy the data contained in a file. A file with a new data type will only occur if a measurement program is modified, or if a new measurement program is developed. Therefore this data can be considered static during the execution of a program.

Derived specifications :

The timestamps in the files may be represented in different forms. Since timestamping is required to select what data to read, it needs to be converted from the various ways time can be represented in a file to a standard representation. Since the time formats were not specified, we have thought of the following representations of time in a file:

Absolute time :

Human readable formats like month/day/year hours:minutes:seconds.microseconds Time in seconds since 00:00 01/01/1970 GMT (the time format used by UNIX). This time format is usually extended with microseconds to get a better precision.

Incremental time :

The timestep between the previous and the current record is specified in the current record. This is usually a time in microseconds

Implicit time :

The timestep between the previous and the current record is constant and not specified in the record itself.

3.2 Specification on the measurements :

The time on the computers connected to the network is synchronised by the Network Time Protocol (NTP). The NTP consists of one computer which receives the accurate time from the global positioning system. This computer sends this time to all the computers connected to the network. The NTP also gives information on the accuracy of the clock in each computer. This accuracy depends on the load of the network. The accuracy of the measurements will be in the order of hundreds of microseconds due to the accuracy of the NTP, which won't get more accurate results even on a lightly loaded network.

analysis :

Microsecond clock resolution will be enough because the accuracy of the measurements will be in the order of hundreds of microseconds due to the accuracy of the NTP. Also the computers on the network run UNIX and UNIX is not a realtime operating system so more accurate timestamps are not expected.

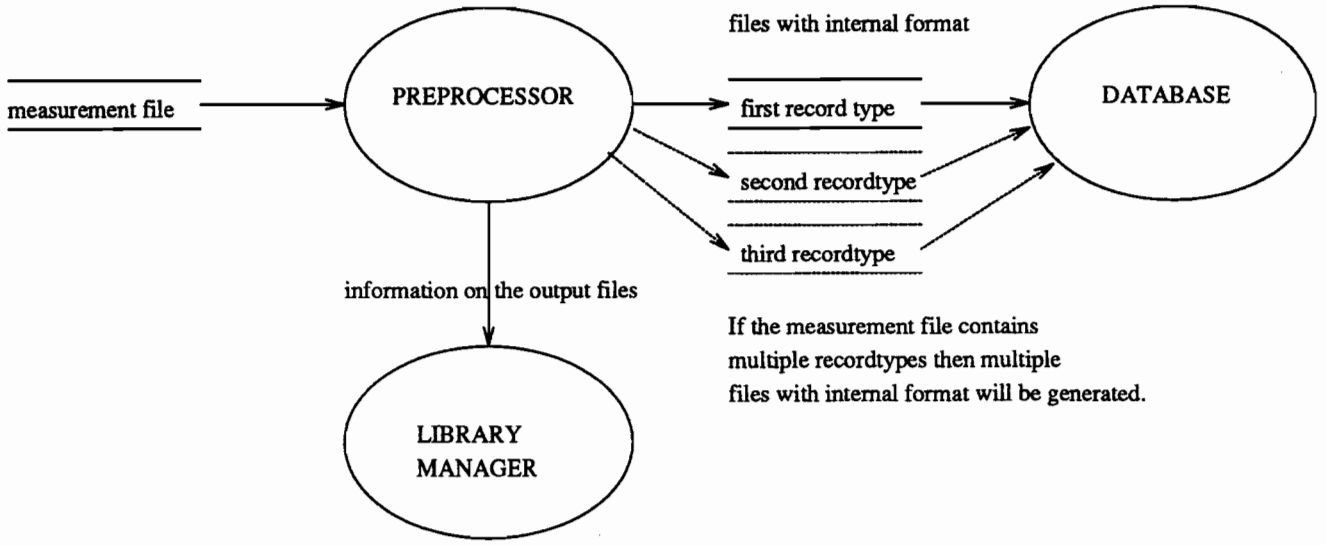


Figure 3-1: Input and output of the preprocessor

3.3 Reading the different file formats

Specification :

The different formats have to be processed and a windowing function must be implemented.

Analysis :

This would become complicated if it had to be done all at the same time. For instance in case of incremental time, the absolute time of the previous record is required to calculate the new record. If a window is empty there is no previous record.

To do this more easily we decided to write a preprocessor which converts all kinds of files into one standard format. This also has the advantage that if new file formats are used which can not be read by the current preprocessor, then only the preprocessor has to be changed. The file format generated by the preprocessor will be called the internal file format. This internal format is binary because that is usually more compact than ASCII files. Besides this it can also be processed faster, because the data in the file can be copied almost directly to memory. Another part of the program will read those preprocessed files and send them to Matlab. This part will be called the database because it has some funtions in common with databases like working with records, selecting records etc.

The prepocessor converts the contents of a measurement file into a format which can be read by the database. To do this the preprocessor needs information on the file. The easiest way is to add a header to each file. From the header the program can determine how to convert the file. Not every measurement program will, however,

output this header. Therefore the preprocessor can also get it's information from the user, if there is no header in the datafile.

When the file is converted, the information on that file will be send to the library manager in order to be stored in a library file. This library file contains all the information on the files which are part of the same experiment. Lines which result in an error when converting, are assumed to be documentation and will be copied to a documentation file.

The files which will be read into memory, have been preprocessed, so they will be binary and they will contain an absolute time in every record. To be able to analyse the data on another computer than it was generated, the database therefore has to be able to do endian conversion. No other conversions are required for preprocessed files.

Chapter 4

Tcl/Tk general interfacing

Generally speaking one can say that every application has some form of a commando language. There are various reasons to choose an exsisting commando language rather than designing your own. Tcl provides the normal expected features of a commando language, such as variables, conditional- and looping commands, procedures, lists, arrays, expressions, and filemanipulation. Tcl is built as an interpreter, the main advantage of this is that there is no need to compile Tcl programs. The disadvantage of it all is that Tcl's performance is slow. All the code is written in C, so inclusion of Tcl in user applications is fairly easy.

Due to the complexity of most of the applications of today, most of the applications work with easy to use graphical user interfaces. (GUI's) Tk provides an extension of Tcl which implements most of the basic GUI functions needed. Tk is build on top of the windowing system X11, which is the most commonly known windowing System on Unix. architectures.

X11 uses the client-server model, see 4-1. The client and server are connected to eachother via TCP/IP sockets. The client side of X11 is the side to which applications connect. The server side takes care of the actual interaction between the keyboard, mouse, display, other I/O devices and the user. The way X11, and most of the windowing systems, work implies that a program using X11, constantly needs to 'watch' for events. A normal gui will actually block on an event-querry until an event occurs. TclTk is implemented using events, to handle events correctly there an eventloop is required. Which means one's own code will actually run under control of TclTk. (Which runs itself under control of X11, and X11 runs under control of the user.)

To interface a program to TclTk one should make it available as a command. Since TclTk runs in an eventloop the commands need to return within reasonable time. Otherwise the application command 'hangs' the TclTk eventloop. If TclTk hangs on an application command it acts as if the application got stuck. This specifically shows one of the drawbacks of TclTk, and most of the other GUI languages, it's not really useable for realtime applications. Realtime applications generally need their

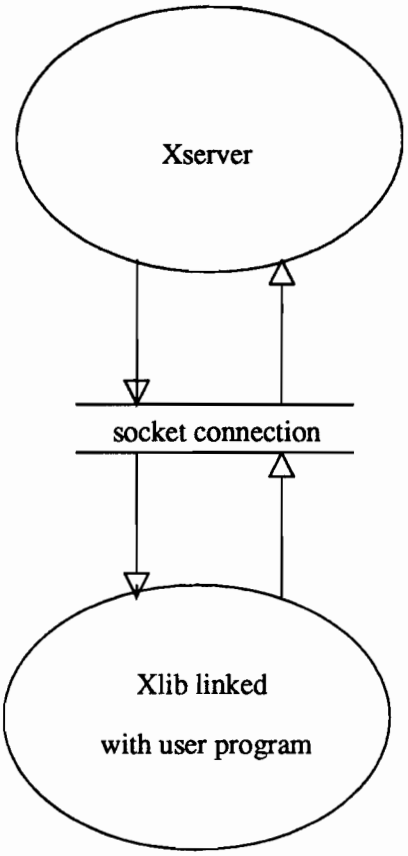


Figure 4-1: X11 setup

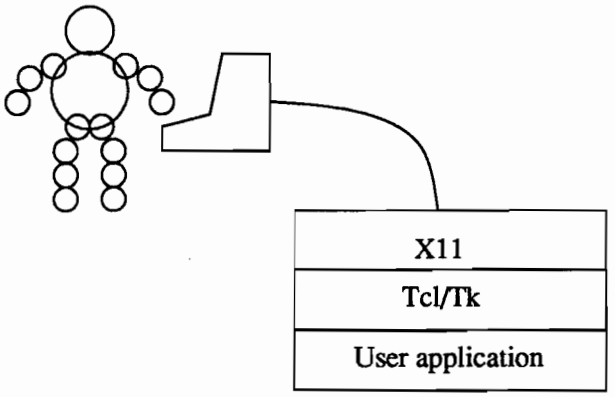


Figure 4-2: layerd model of user, X11, TclTk, application

own event loop.

4.1 Design of the application to TclTk interface

The first goal of the interface designphase was to design an interface which was useable for any application, not just for Netalyse.

Just like Tk is hooked up to Tcl your own application should hook up to TclTk. The way it is done is as follows: At startup Tcl_AppInit is called, this function performs all the initializations needed for the application. Since 'our' application needs to hook the commands into Tcl, 'our' application should be initialized at this point also. So we have to write functions which initializes the specific parts of 'our' application. Just to make insertion of new initfunctions easy this is done with the help of an array of functionpointers.

```
/*
 *  typedef ptr 2 an init function
 */

typedef int (*ptr2InitFunction_t) (Tcl_Interp*);
  .
  .
  ptr2InitFunction_t anInitFunction[] = {

    Tcl_Init,
    Tk_Init,
    Own_Init_Function,
    Own_Init_Function,
    ....,
    NULL
```

```
    interp->result = exportUserError();
    return TCL_ERROR;

  }

  return TCL_OK;

}
```

To keep the interfacemodules uniform, th above described function calls an evaluation function which is the actual implementation of the command. If the command consists of subcommands, this function calls the subfunctions related to the subcommand which was invoked.

```
int Eval_Command (Tcl_Interp *interp, int argc, char *argv[]) {
  /* input   : interp, the interpreter in which the command is
   *         invoked.
   *           argc, number of arguments parsed along with the
   *           invocation.
   *           argv, array of characterstrings containing the
   *           arguments.
   * returns : OK if no errors, NOK if an error occured
   * output  : -
   * desc    : Handles database commands.
   */

   if (strcmp (argv[0], "Sub_Command_1") == 0)
     return subCommand1 (interp, argc-1, &argv[1]);

   if (strcmp (argv[0], "Sub_Command_2") == 0)
     return subCommand2 (interp, argc-1, &argv[1]);

   .
   .
   .

   if (strcmp (argv[0], "Sub_Command_n") == 0)
     return subCommandn (interp, argc-1, &argv[1]);

   userError ("unknown subcommand %s\n", argv[0]);
   return NOK;

}
```

The subfunction can implement any functionality needed for the application

As one can see to all the functions a structure called Tcl_Interp is parsed. This structure contains some handy gadgets which can be used for various purposes. This structure is an overlay of a much bigger structure which contains all the references to the interpreter, since the bigger structure only contains specific information for the interpreter itself, that structure is not discussed here.

```
typedef struct Tcl_Interp {
    char *result;          /* Points to result string returned by last
                            * command.
  */
    void (*freeProc) _ANSI_ARGS_((char *blockPtr));
                           /* Zero means result is statically allocated.
                            * If non-zero, gives address of procedure
                            * to invoke to free the result.  Must be
                            * freed by Tcl_Eval before executing next
                            * command.
          */
    int errorLine;         /* When TCL_ERROR is returned, this gives
                            * the line number within the command where
                            * the error occurred (1 means first line).
  */
} Tcl_Interp;
```

The result field points to an array of 200 bytes, which can be used to parse messages to Tcl/Tk. This array is allocated statically. It's feasible that there are occasions that one wants to parse messages bigger than 200 bytes. In this case one can assign a pointer to a own allocated area of memory which is big enough to hold the data. That area can either be statically or dynamically allocated. In case it's allocated dynamically (using ie malloc) that area needs to be freed after parsing the message. The freeing of the message has to be done after it is read by the interpreter, this can be achieved by parsing a function pointer to the interpreter which points to a function which does the apropriate freeing of the memory. This is what the freeProc field is used for. The use of this field can also extended to gracely exit functions. If a function opens several resources and has multiple exits, its handy that the resources will be closed properly by the call to the function which is pointed to by the freeProc field. The errorLine field allows assignment of an integer which indicates in which line in the source the error occured. In the current interface this option is not used, but in the second revision of the interfacetemplate this will be used also, to conform completely to the interface specification.

Note : All this stuff is barely documented in the Tcl/Tk manual and all together it took quite a while to figure out all the stuff.

Chapter 5

Matlab interfacing

Matlab is a commercial software package meant for generic numerical analysis. It mainly consists of two parts, a numerical computation unit and a graphical display unit. Matlab has an easy to use command-language (just like TclTk) which has most of the normally expected programming capabilties. Next to that there are several matrix specific and calculation specific programming constructions. (vectorized calculations) The command language allows creation of complex scripts. In this way more complex calculations can be represented in simple scripts.

5.1 Analysis

Matlab is not really made to function as a part of another application, although they provide an extension which forks a matlab process. The Matlabprocess is connected via pipes to the process which uses the interface. The interface allows parsing of plain Matlab commands. All the interfacing code is contained in a library and in a headerfile The calls needed for forking off a matlab engine, controlling and transfering matrices are included in the library, so writing the code is fairly easy. The interfacing code written on top of the code provided by matlab takes care of errors and encapsulates the data structures that have no need to be seen by the user.

Figure 5-1: matlab interproces communication

5.2 Matlab interface design

All the calls to fork the engine etc are included in the library, so the only thing one has to do is call them in the correct order. The pipes which connect the processes are not even visible, since they are encapsulated in functions which do the actual passing to the pipes etc. The disadvantage of the encapsulation shows when it comes down setting up the reversepipe. (From matlab to one's own proces.) One has to set up a big buffer to hold the messages Matlab sends back. If the buffer is not big enough to hold the message, it's truncated. The functionality needed to interface any application to Matalb consists of the following.

- Creation of a Matlab environment.
- Destruction of a Matlab environment.
- Evaluation of commandstrings.
- Transfer of matrices from an application to Matlab.
- Transfer of matrices from Matlab to an application.
- Freeing matrices in Matlab.
- Export matlab buffer.

Since Matlab consumes many resources on a system it's not really useful to implement a core which can handle multiple instantiations of Matlab engines. All the data which is not needed by the user should be abstracted by encapsulating it. All the error checking should be done within the procedures.

5.2.1 Datatypes

There is not really a need for datatypes within the matlab module, the only one needed is the structure which binds logical components together. The only two components which can be bound together are the outputbuffer and a pointer to a stucture which refers to the Matlab engine.

5.2.2 Function specifications

name : createMatLabEngine
input : -
output : -

name : destroyMatLabEngine

input : -
output : -

name : evalMatLabCommand
input : matlabcommand
output : -

name : getMatrix
input : -
output : matrix

name : putMatrix
input : MatrixName, length, width, matrix
output : -

name : freeMatrix
input : matrix
output : -

name : getMatLabBuffer
input : -
ouput : Matlab output buffer

5.3 Interfacing Matlab to Tcl/Tk

Within Tcl/Tk a command Matlab should be made available which parses plain Matlab-commands to the Matlab engine. Since the Matlab interface is setup in such a way that it's easy to interface to c, this makes it also easy to interface to Tcl/Tk. The same functions are used as used for a general interface to Tcl/Tk, except the internals of the functions differ slightly from a 'normal' setup. To parse the outputbuffer to Tcl/Tk an extra function has to be created. The extra should implement the parsing from the Matlab output buffer to the Tcl/Tk domain. As mentioned before in the Tcl/Tk interface description, the resultpointer is used for parsing the buffer. Next to that Matlab has not any provisions for printing times allong the x-axis therefore also an extra function is needed. This function should be able to automatically determine the correct scale and number of values to put on the x-axis. The function which makes the Matlab command available to Tcl/Tk also creates a Matlab engine, so the user doesn't have to invoke a separate routine which forks the Matlab engine.

5.3.1 Matlab commands in Tcl/Tk specifications

name : matlab
input : matlabcommand
output : -

name : getMatlabOutput
input : -
output : matlab output buffer

name : createTimeAxis
input : begin time, begin date, end time, end date, matlab axis handle
output : -
note : This function is not implemented mainly due to a lack of time.

Chapter 6

Basic routines needed for Netalyse

6.1 General analysis

Two basic tools are needed , a generic linked list utility, and a generic file-I/O utility which is capable of handling windows on files. There are several design approaches to such generic tools, in this case a choice is made for an object oriented approch. The objects that need to be handeld are seen as abstract data types on which a number of functions operate. This implies that the objects operated on can not be seen directly by the user. Only functions which work on the object can modify the object, ie the object is not directly accesible by the user.

6.2 General Design

Since the basic-routines need to be implemented in plain-C, this will create some constraints on the design.

- Abstracting the user from any data which is not neccessary for performing a certain function. (ie. if one uses the fread systemcall on a file, one should also parse the filepointer to that function. This pointer is neccessary for the fread function to determine which systembuffer to use, but is of absolutely no interest to the user.)
- Abstract the user from messing around with systemcalls which can be complex.
- Relieve the user from the normal error-checking overhead on systemcalls.
- Extend functionality of standard libraries (i.e. linked lists are not implemented in standard libraries.)
- Encapsulate important data in modules in such a way that the user can not access that data. (Static declared variables are always accessible by declaring

in another module the same static, the linker solves this and makes it reference the same memoryspace. Conclusion is that if the user really wants to access data there is always a way to do that. Although this should be considered to be 'tricky' programming)

- If encapsulated data needs to be accessed, that data should be accessed through a function, which can do additional checks on the passed data.

An approach like this has it's disadvantages :

- There is a need for a selection-method in case the functions have to operate on multiple instantiations of objects (files, linked lists etc.) This selection algorithm needs a key, otherwise there is no bases on which objects can be selected. This selector needs to be parsed to the user.

- There is a need for administering the objects. Since generic code is meant to be as general as possible, the number of objects should not be bounded to a maximum, so the allocation of objects has to be done dynamically. This can either be done by means of dynamic arrays or by using a linked list.

Adminstration of objects could be done either using a linked list or a dynamic array. A dynamic array has the disadvantage that when it needs to grow there is a chance that the whole array has to be copied in memory, since it won't fit at the current location. An advantage of a dynamic array is that compared to a linked list the access speed is faster since one can index directly into it. It's clear that the choice is arbitrary. In this case the choice is made in favour of a linked list, because that is also a part of the basic-I/O routines.

There also has to be a mechanism which preserves the state on a current level in a application. This is achieved by using the dynamic array structure, since the total amount of memory used is small, rough estimation a maximum of 20 quadwords, this should not pose a problem. The array is organised as a lifo (stack) so as one goes down a level the current state is pushed onto stack, and as one goes up the current state is poped of stack. This can be made clear with an example : Suppose the user selects a listobject on toplevel, the user would expect that from that moment on the listobject stays selected, but there could be a layer below the toplevel that also uses listobjects, i.e. if the toplevel calls a function which also uses listobjects. In that case the state of the top level has to be preserved by the previous described mechanism. If not the function would return to top level leaving the state of the core managing the list objects modified, that this would be a major problem speaks for itself.

6.3 Linked-List Design

The linked list engine is as mentioned before based on a selectionalgorithm. That makes the core behave as an infinite statemachine. The different states in which

the linked list core can exist are numereous. Due to that there is a need for state preserving functions. State preservation is achieved by using a combination of a stack which stores the current list object and some variables mentioned in the datastructure.

Primairy functionality

- Create a list object.
- Destroy a list object.
- Insert an element.
- Delete an element.
- Read an element. (get)
- Write an element. (set)
- Selection of a list on basis of an ID, generated while the list was Created.
- Saving of current context.
- Restoring of current context.

Secondairy functionality

- Moving through the list.
- Rewind to the beginning of the list. (BOL)
- Wind to the end of the list. (EOL)
- Size of the list in number of allocated elements.
- Status of the list.
- Searching through the list.
- Element Id tagging function.
- Element Id search function.

Debugging functionality

- Print out of the internal links. (the listobjects)
- Print out of a specified list object.
- ID & Name (if given) of the list.

The internal housekeeping of the linked list-utility also consists of a linked list which contains the list-objects themselves. Each listobject is identified by a unique identifier. This identifier can be obtained in several ways. By far the easiest way is by tagging the objects with their sequence number in the list. The disadvantage of this method is that the objects need to be inserted in a sorted manner. So each time a new object is created the list has to be run down from the beginning either until the end or until an available identifier is found. Identifiers become available when objects are deleted from the list.

To achieve a consistent movement there are several solutions, one could take care of every exception that could occur, this results in a lot of code. Another option is to create a state machine. In this case the choice is made for a state machine. The state machine reflects the current status of the linked list. The state machine is on a per object basis, so if one changes from one list to another also the current state changes. The state of a linked list is stored in the object itself.

6.3.1 Data types

Each listobject is a datastructure which itself is a element of a linked list. This implies that the datastructure needs pointers which maintain the linked list of which the object is a part of. To speed up the selection algorithm a dual chained linked list is convenient since we can go back and forth in the list of objects. The structure also needs to contain a pointer which point to the list. For faster access a head and a tail pointer are implemented. The sizeOfElement field describes the default size which is allocated per element. The ptr2CurrentEntry field is a pointer which points to the last accessed element.

type	name
pointer	ptr2Next
pointer	ptr2Prev
integer	listIdentifier
string	nameOfList
pointer	ptr2HeadOfList
pointer	ptr2TailOfList
integer	sizeOfElement
integer	numberOfElements
integer	stateOfLastMove
pointer	ptr2CurrentEntry

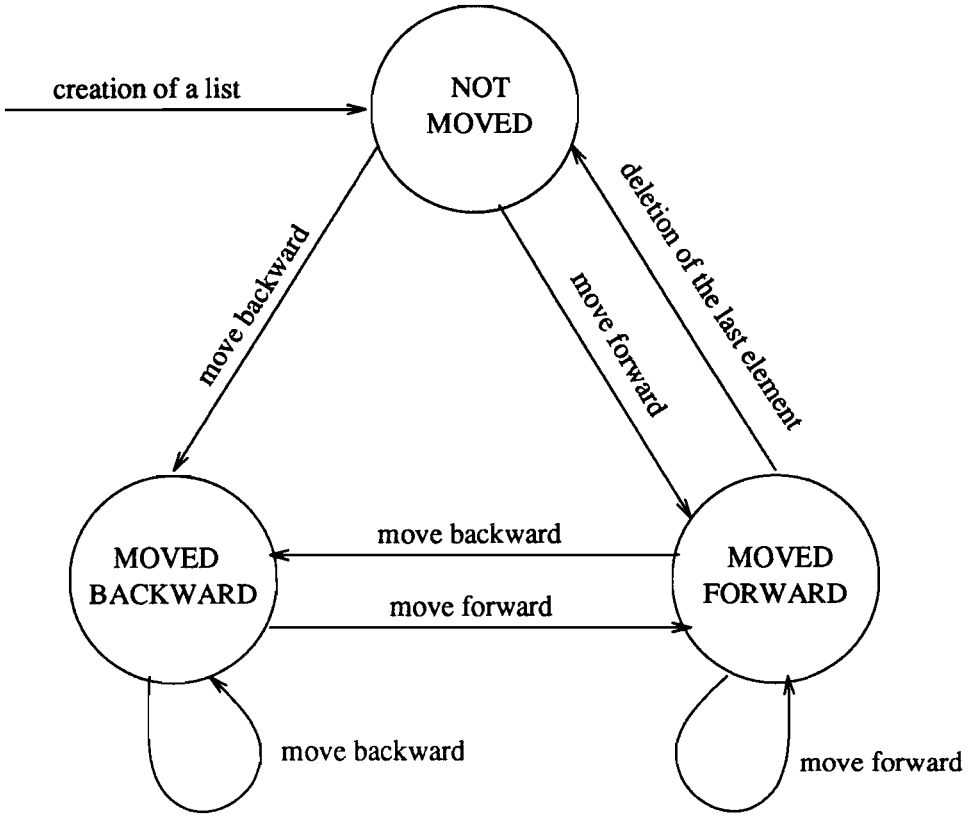


Figure 6-1: linked list state machine

Each element is also contained in a data structure, this structure should at least contain a pointer to the next element and a pointer to the previous element. Since we want to be able to cope with elements with varying sizes within one list, also in each element the size needs to be included.

type	name
pointer	ptr2Next
pointer	ptr2Prev
integer	elementSize
pointer	ptr2Data

The reason why there is also a pointer for the data itself, is that it makes the linked list even more flexible, so there is also a option to assign the pointer to an allocated area of memory. Normally the functions provided should take care of allocating memory. During the development of file-I/O it became clear that there was a need to have direct acces to the data pointer. The alternative was to have two copies of the same information in memory. To keep the two copies consistent, every change had to be copied to the other and so on. This could be solved by having direct access to the datapointer. So some extra functionality was needed.

Datapointer functionality

- Get data pointer
- Set data pointer

The last function mentioned is currently not neccessary but may be used by future extensions of the core. It allows to set the datapointer to a user specified memory area, this implies that the core shouldn't use the pointer in this case to allocate memory.

6.3.2 Function specifications

Primary functionality

name : createList
input : sizeOfAnElement, nameOfList
output : listIdentifier
note : the nameOfList is an optional argument, it can be handy in case of debugging otherwise the user has to match up the listid's which is not easy at all.

name : destroyList

input : -
output : -

name : insertElement
input : anElement
output : -

name : deleteElement
input : -
output : -

name : getElement
input : aDirection
output : anElement

name : setElement
input : aDirection, anElement
output : -

name : selectList
input : listId
output : -

name : saveListContext
input : -
output : -

name : restoreListContext
input : -
output : -

Secondary functionality

name : moveList
input : direction, repeatCount
output : -

name : rewindList
input : -
output : -

name : windList
input : -
output : -

name : getListSize
input : -
output : listSize

name : getListStatus input
output : listStatus

name : getFreeDataIdentifier
input : offset
output : dataIdentifier

name : searchDataIdentifier
input : offset
output : -

Debugging functionality

name : printControlLinks
input : -
output : -

name : printListLinks
input : -
output : -

name : getListId
input :
output : listName
note : The name is optional, by passing a null to this function no name is parsed back.

Datapointer functionality

name : getPtr2Data

input : direction
output :

name : setPtr2Data
input : -
output : -

6.4 File-I/O Design

Since there is a need for reading files and marking windows within a file there is implicitly also a need for reading backwards in a file. The reason why there was a need for windows within a file was that because of the size it is not possible to read it into memory all at once. (file sizes $\approx$ 300Mb) Also there is no reasonable way to calculate offsets into files, in such a way that there is no need anymore for reading at every different access the whole file again.

Elementary functionality

- Open a file
- Close a file
- Select a file
- Read ASCII records
- Write ASCII records
- Read binairy records
- Write binary records
- Saving of current context
- Restoring of current context

Secondary funtionality

- Goto end of file (EOF)
- Goto begin of file (BOF)

Windowing functionality

- Open a window on a file
- Close a window on a file
- Select a window
- Mark begin of a window
- Mark end of a window
- Goto begin of the window
- Goto end of window

The file-I/O utility is build on top of the linked list core to do the internal housekeeping. Most of the functions are easy to implement. Every file is represented by an element in a filelist, it contains the information concerning one file. Every time a file is opened, an entry in the list is created, and accordingly on closing th corresponding entry is removed.

6.4.1 Data types

There is a need for two datatypes within the fileio core, none of which should be visible for the user. The first one should contain all the properties neccessary for the file itself, the second on should contain all the properties neccessary for the windowdata. The datatypes used are actually elements in linked lists. But since the linked list core is used the datatypes conceal the typical pointer hassle. The file object should also preserve the state in which the window list was, because there is no other way of preserving that state. Every fileobject has one associated linked list which controls a list object which contains all the windowobjects.

type	name
integer	fileIdentifier
long	previousOffset
int	windowListIdentifier
pointer	ptr2Window
pointer	ptr2FileDescriptor
int	recordSize

type	name
integer	windowIdentifier
long	windowStart
long	windowEnd

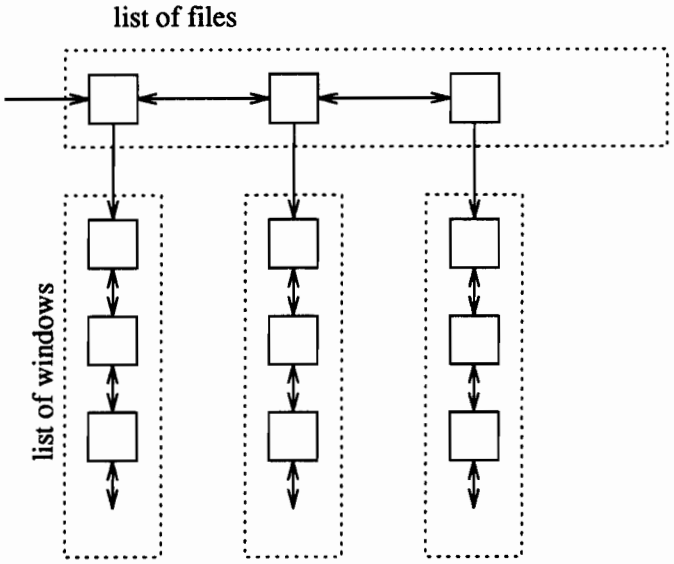


Figure 6-2: linked list structure

Within the fileobject a pointer is contained which maintains the current selected window. The data stored in a windowobject is only the begin and the end of a window within the file. The begin and the endpoint can be obtained by requesting the offset relative to the beginning of the file. One can jump to those offsets with certain systemcalls.

6.4.2 Function specifications

Primairy functionality

name : openFile
input : filename, filemode, recordsize
output : fileIdentifier
note : recordsize is used for binairy mode reading

name : closefile
input : -
output : -

name : readLine
input : direction
output : line

name : writeLine

input : line
output : -

name : readRecord
input : direction
output : record

name : writeRecord
input : record
output : -

name : saveFileContext
input : -
output : -

name : restoreFileContext
input : -
output : -

Secondairy functionality

name : go2Eof
input : -
output : -

name : go2Bof
input : -
output : -

Windowing functionality

name : openWindow
input : -
output : -

name : closeWindow
input : -

output : -

name : markBow
input : -
output : -

name : markEow
input : -
output : -

name : go2Eow
input : -
output : -

name : go2Bow
input : -
output : -

name : selectWindow
input : -
output : -

Chapter 7

Database design

7.1 Record definitions

The user will specify the types of records in a file. We have assumed this file won't change often so the file is read when the program begins and remain unchanged during execution. The record definition file can be modified with any text editor. The disadvantage of this approach is, that this is not the easiest way for the user to change the recordtypes he has available, but again the user won't have to change this file often.

This record definition file contains the record definitions and have some explanation of the format of the file. The definition of a record looks like this:

recordname; fieldname fieldtype; fieldname fieldtype ...

Fieldtype will be something like integer, floatingpoint or time The recordname and fieldname can be anything the user wants. The recordname is used to identify the record type so it has to be unique. Comment are supported in the file. Comment lines will start with a special marker : "*" When this file is read it produces the following datastructures:

- a list of record structures
- containing per record :
  - the name of the record, and a list of field structures
- containing per field :
  - the field type as a number and the field name.

When processing the record definitions file, three levels can be distiguished: the file level, the line level and the field level. The file level will open files and select definition lines.

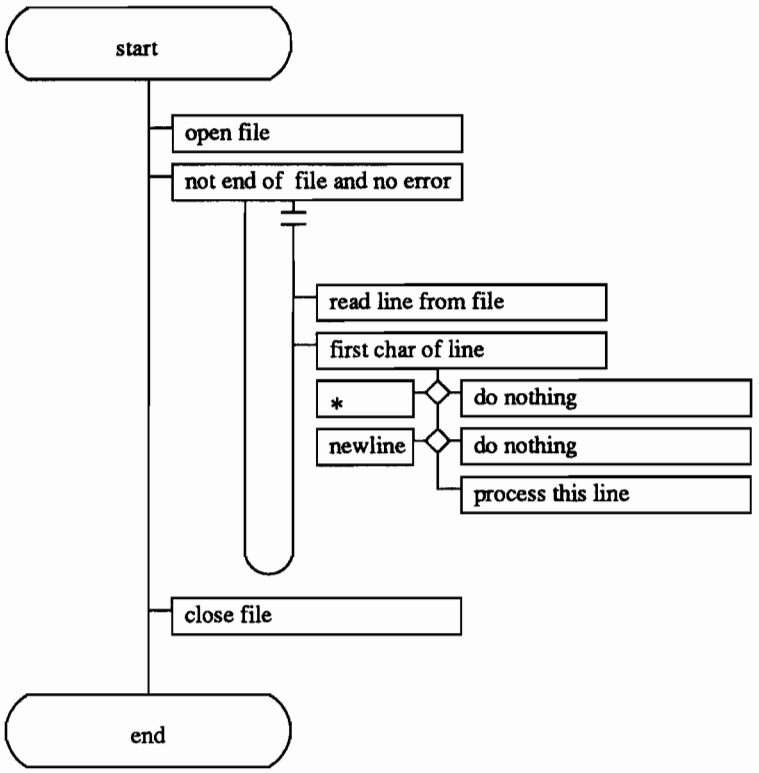


Figure 7-1: readconfig

name : readconfig
input : filename of record definitions file
output : the record definitions structure filled in result indicating whether an error occured or not

In the second level, one line will be converted to a record definition.

name : processline
input : a line with a record definition
output : one record definition result indication whether an error occured or not

In the third level a pair of words is converted to a field definition

name : string to field
input : a string containing a fieldname and a fieldtype
output : a field definition result indication whether an error occured or not

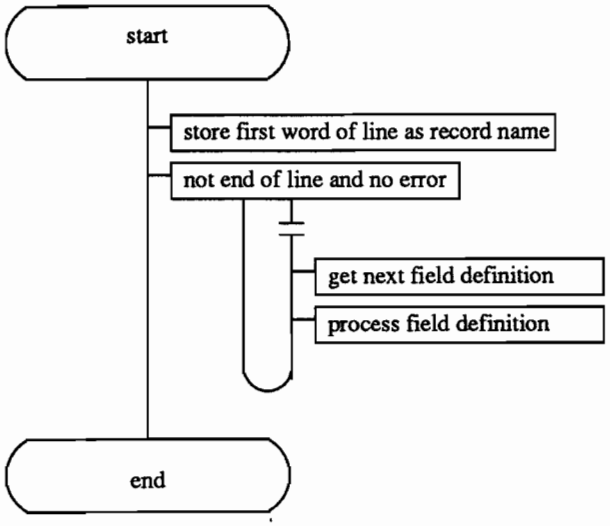


Figure 7-2: processline

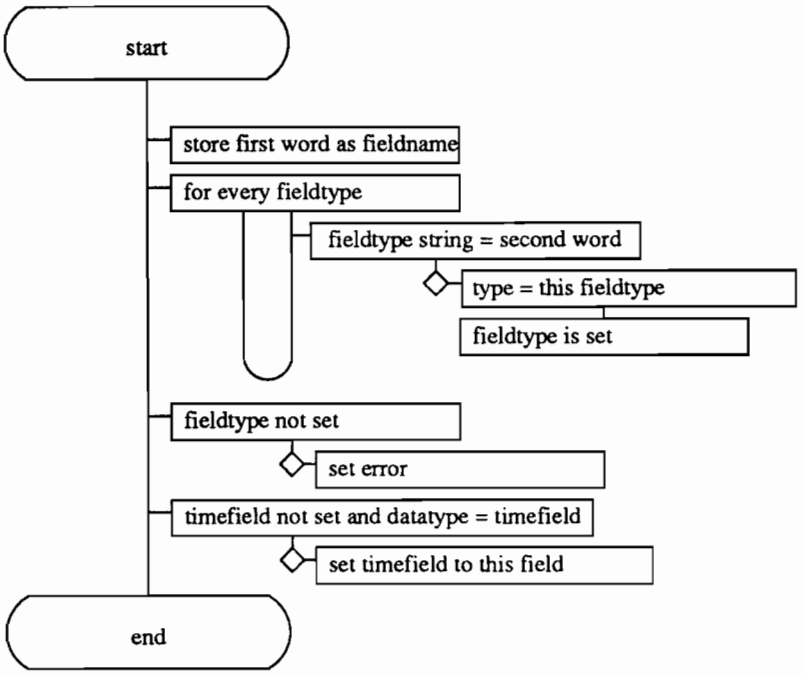


Figure 7-3: string to field

7.2 Field Type handling

Field type definition ?

A record is composed of several fields. How a record is processed depends on the type of the field. The actions taking place in case of an ASCII-file are typically:

- Read a line from a file
- determine the type of the record
- Take the first field and convert it to an internal representation
- store this converted field in an internal record
- take the second.....
- and so on until the last field of the record.

For converting time and date to an internal representation of time there are two fields required which can not be converted seperately because they convert to one internal structure. The definition of a fieldtype must therefore depend on the internal type (time in this case) and not on the whitespace seperated fields of the record in the file.

A fieldtype gives the type of a field in an internal record, and specifies the operations on a field. Absolute time, for instance, will be a fieldtype which can be calculated from two fields in the input record : date and time.

The conversion from string to internal representation, which is required for reading ASCII files, depends on the fieldtype. Also, when reading a binary file the conversion depends on the type. The conversion from incremental time to absolute time for instance, is an example of a field dependent conversion which has to be done when reading a binary file. For debugging and displaying of data it is necessary to convert the internal representation to a string. To select data, a function which can compare two fields of the same type is required. Because the type of a field can be a structure, the endian-conversion depends on the field type.

7.3 Working with fieldtypes

Fieldtypes defined in the program have to be extended when new measurement tools use new formats. This means that the functionality which is related to the fieldtypes must be seperate from the rest of the program. Also, the number of fieldtypes defined may not influence the part of the program which works with the field types.

For instance a routine which has to read from a file needs a different conversion function for every fieldtype it has to handle. To make this routine independent of the fieldtypes which are defined, it has to call a function which converts a string of

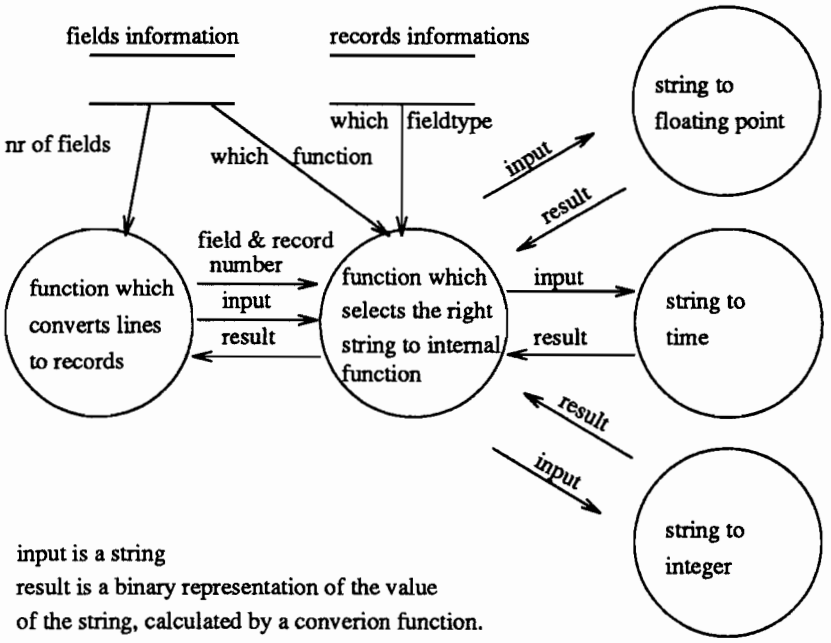


Figure 7-4: converting a string with fieldtypes

any type to an internal representation. This function converts a string to an internal representation depending on the fieldtype. This is illustrated in figure 7-4

Example :

Assume, the field types absolute_time, integer and float are defined. These three fields together make up one record, which could look like this :

06/20/94 12:30:05 123 10.5

This record has to be converted to an internal representation. The record type is given. The fieldtype can be derived from the recordtype and the fieldnumber. The first fieldtype is time. So the conversion routine for converting a string to an internal representation of time must be called. The function which is converting the whole record will call the conversion function which converts strings to internal representation with the recordtype and the fieldnumber as parameters. This general string to internal representation function will call a routine which will convert the string to the internally used representation of time.

For every operation which is required on a fieldtype there has to be general function which will call the function specific to the fieldtype must exist.

The functionality which is required for the field types depends on the needs of the program. Every time the program needs a new operation on a fieldtype, it can easily be added by writing that function for every fieldtype available and another function

which provides this new function to the rest of the program. These functions are discussed in paragraph

For each fieldtype the followin has to be stored :

- A name used to identify the type in the record definitions file
- The size in fields for conversion from records
- The size in bytes when stored in memory or preprocessed file
- The size in bytes used in a unpreprocessed measurement file
- a function which converts a string to a fieldtype
- a function which converts a binary input to a fieldtype
- a function which performs a correction calculation on a field
- a function which converts an internal file to an internal field
- a function which converts a field to a type readable for Matlab
- a function which converts an fieldtype to a string
- a function which compares two fieldtype elements

7.3.1 Defined fieldtypes

The fieldtypes we've defined sofar have the following characteristics :

- Integer :
  - String :
    Must be a positive or negative number with in the range of a 32 bit integer.
  - Binary external format :
    Little or big endian 32-bit integer.
  - Correction calculation :
    None
- Float :
  - String :
    Must be a positive or negative number with in the range of a 32 bit IEEE floatingpoint.

  - Binary external format :
    Little or big endian 32-bit IEEE floatingpoint.
  - Correction calculation :
    none

- Double :

  - String :
    Must be a positive or negative number with in the range of a 64 bit IEEE floatingpoint.
  - Binary external format :
    Little or big endian 64-bit IEEE floatingpoint.
  - Correction calculation :
    none

- Time :

  - String :
    a date in the format mm/dd/yy followed by the time in the format hh:mm[:ss[.uuuuuu]]
  - Binary external format :
    GMT time in seconds since 01/01/1970 in 32 bit plus microseconds 32 bit unsigned
  - Correction calculation : none

- Incrementaltime :

  - String :
    a timestep in microseconds not bigger than the max value of a 32 unsigned number
  - Binary external format :
    GMT time in seconds since 01/01/1970 in 32 bit plus microseconds 32 bit unsigned
  - Correction calculation :
    The incremental time is calculated by adding the timestep to the time in the previous record

- implicittime

- String :
  empty
- Binary external format :
  0 bytes
- Correction calculation :
  The implicit time is calculated by adding the value in the constant record to the time in the previous record

7.3.2 Defined functions which work with fields

name : str2Type
input : record type, fieldnumber, the field as a string
output : the field in internal representation

name : cmpType
input : record type, fieldnumber, field1 and field2 in internal representation
output : -1 if field1 < field2, 0 if field1 = field2, 1 if field1 > field2

name : type2Str
input : record type, fieldnumber, a field in internal representation
output : a string representing the input field

name : type2Double
input : record type, fieldnumber, a field in internal representation
output : the field represented in double format

name : ext2Int
input : record type, fieldnumber, a field in external representation
output : the field represented in internal representation

name : file2int
input : record type, fieldnumber, a field in internal file format
output : the field represented in internal representation

name : correctType
input : record type, fieldnumber, the constant record, the previous record
the current record
output : the specified field will contain the calculated value

The functionality of all the routines above is the same since the conversion is done by fieldtype specific functions. The difference between them are the parameters with which they are called. For each of the functions listed above there are functions specific to the fieldtype. Some fieldtypes share functions for the same operation. The fieldtype specific functions have a limited complexity so they will not be discussed in further detail. The functionality of most of the conversion routines is limited. The routines providing functions are also very simple: just look up the fieldtype with the given record and field number and call the right function.

7.4 Working with records

Records have to be copied, stored and deleted and fields have to be stored in and retrieved from records. For storing the records the size of the record has to be calculated. This size can be calculated by adding the sizes of the fields comprising the record. To retrieve the field from a record, the size of the field and the offset have to be known. The size of a field depends on the fieldtype and is specified with that field type. The offset of a field in a record can be calculated after the record definition file is read. The size of a record can be calculated at the same time.

The data structure :

- max size of any record for buffers
- a list of size/offset structures
- containing per record
  - the size in memory
  - the size on disk
  - a list offsets
  - containing per field
    - an offset

To fill this structure, for every field in every recordtype the size is requested and the offset calculated

name : record type init
input : record type definitions and the field type definitions
output : the records sizes and offsets structure filled in.

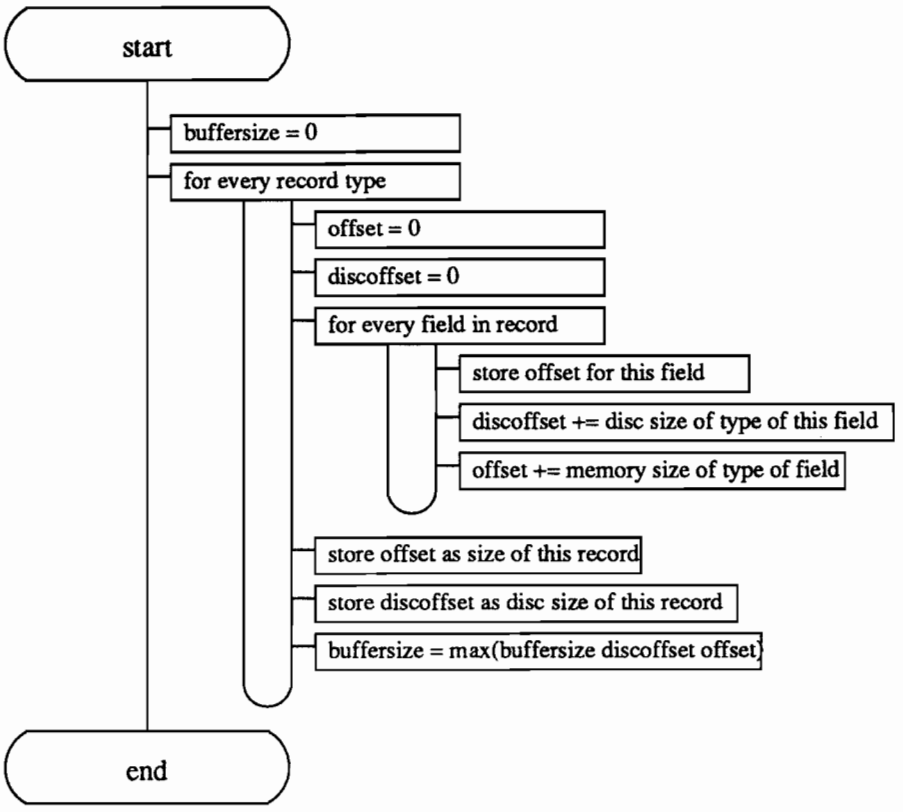


Figure 7-5: record type init

The function to extract a field from a record is simple since the offset is stored. The function uses the record type and field number to find the offset and the size of the field. The it copies it out of the record.

name : getField
input : a record, the record type and field number
output : the selected field

In the same way a field is extracted from a record containing it.

name : store field
input : a record, the record type and field number
output : the modified record

With the functions defined on fields, the same functions can be performed for records. A line of text can be converted to a record. The line could have a record identifier as the first word. A record from an external binary file can be converted to an internal binary record. An record in internal file format can be converted so it has the right endian format. Also the correction calculation can be done per record. This requires three new functions:

name : str2Rec
input : the list of record types, a string containing a line, the untyped flag
output : the record type in this line, the result indication succes or failure

name : ext2Rec
input : the record type, a binary record in extern format
output : the binary record in internal format

name : int2Rec
input : the record type, a binary record in internal file format
output : a record in internal file format after endian correction

name : correctRec
input : the record type, a binary record in internal file format
output : a record on which the correction calculations are executed

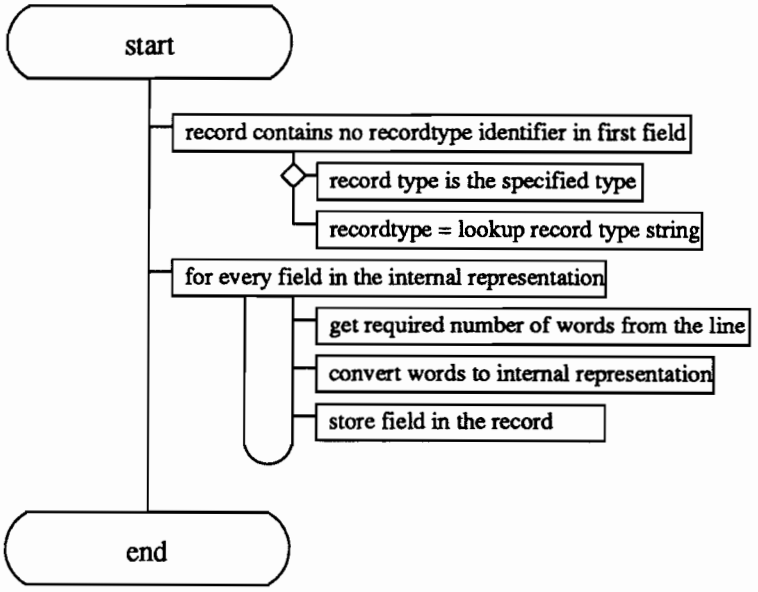


Figure 7-6: str2Rec

7.4.1 Reading preprocessed files into memory

The following are the requirements for the database part of the program which will read prepocessed files into memory:

- Endian conversion
- Multiple files must be open at the same time
- data from multiple files will be in memory at the same time
- The data in memory has to be treated as a window on a file which can be
- resized and moved across the file.
- The window is specified by a starttime and an endtime
- multiple windows can be opened on a file.

Multiple files means a list of files must be maintained. A request for data will be called a query as for a database. A new query opens a new window on a file. For each file there will be a list of queries. Each query results in data in a list of records read into memory.

For each file a fileID needs to be stored, which can be used by TCL/TK to indicate which file it is. The identifier for the list in which the data is stored, information on how to read the file, the record type and a boolean indication big of little endian. Also the beginning and end of the window on the file need to be stored.

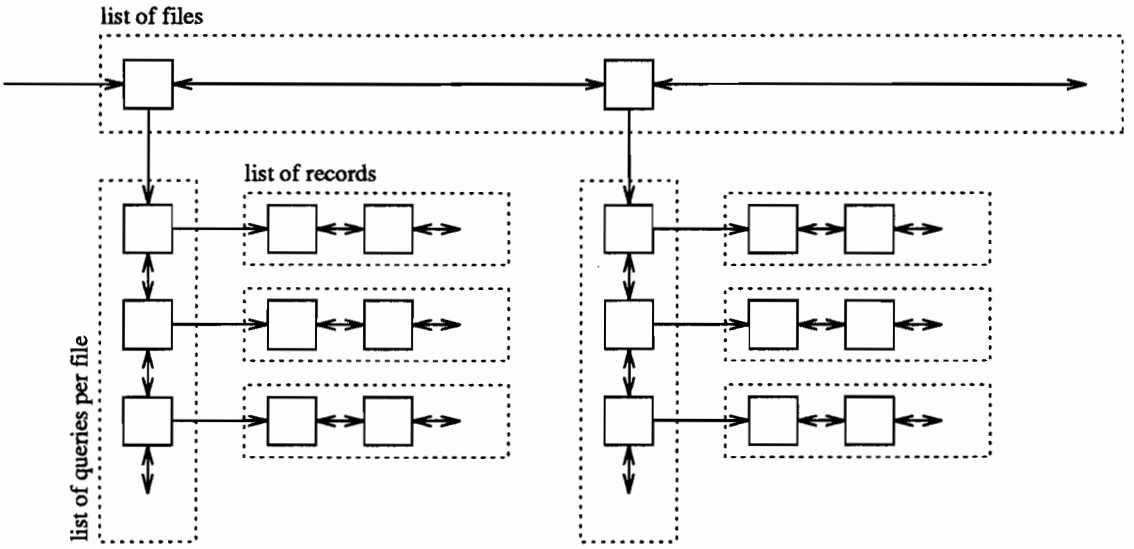


Figure 7-7: The main datastructure of the program

Per query the following needs to be stored: the number of the list in which the data is stored, an identifier for TCL/TK to refer to this query.

This data will be very dynamic so it must be stored in linked lists. With this a linked list data can easily be added at both the beginning and the end of a window. The structure of the required lists is illustrated by figure 7-7

- a list of opened files
- containing per file:
  - an file ID
  - fileinfo needed to read the file
  - a query list
  - containing per query:
    - a filewindow ID
    - the record type of the file
    - a list containing the record

The list structure shown in figure 7-7, can be modified in a couple of ways. When a file is opened, an element is added to the file list with information on that file. When a query is opened, a query element is added to the query list then the empty data list is created. When a query is closed the datalist and the related query element are deleted. When a file is closed all the related queries will be closed.

name : openfile
input : filename, little endian flag
output : a file identifier

name : open query
input : none
output : a query identifier

name : close query
input : query identifier
output : none

name : close file
input : file identifier
output : none

7.5 Querying data

With a query data can be read from an opened file. The user specifies the time interval he wants to read. For a new query he just specifies the file to read from. For a modification of a query the user has to specify which query he wants to modify. If the modification is a reduction, then the list has to be reduced and with the list the window on the file has to be made smaller. When when a window is increased and more data has to be read, it can be necessary to read from both the start and the end of the window. This is handled by the following algorithm: first reduce the list by deleting all the records from the start of the list that are not required. Then remove all the records from the end of the list that are not required. If no records were deleted from the start then data has to be added by reading back from the start of the window towards the start of the file. If no records were deleted from the end of the file then data has to be read from the end of the window towards the end of the file.

This results in the functions this means two functions are required: one for reduciing the list if necessary and one for reading data from the file. Both functions will provide a forward and a backward option.

name : modify query
input : query identifier, starttime and endtime
output : the query will be modified

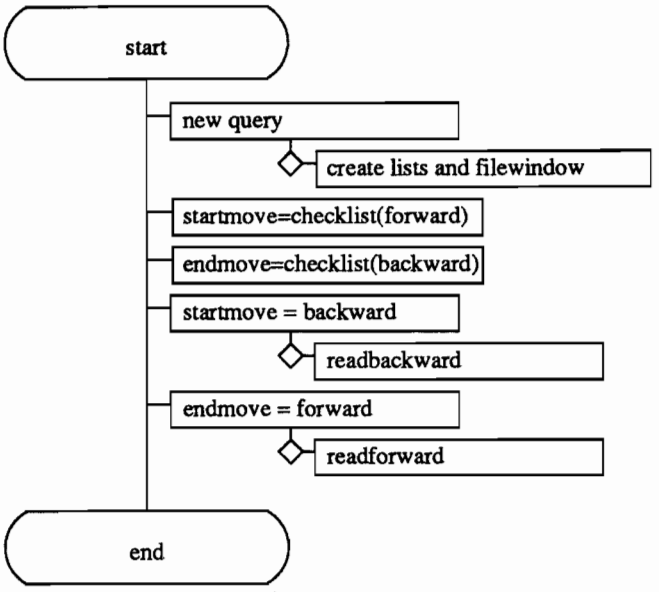


Figure 7-8: modify query

name : checklist
input : query identifier, direction, starttime and endtime
output : the list will be reduced if necessary and result will
indicate whether to read on in the specified direction

name : readforward/backward
input : query identifier, file identifier, starttime and endtime
output : the list will be expanded with data from the file

7.6 Sending data to Matlab

Matlab expects the input data to be arrays of doubles. Data in a query is a list of records. From this list, lists of fields can be created. The fields must be converted to doubles and put in an array. Which can then be send to Matlab. Since one array of fields is send at a time, the query and the field must be specified.

name : listField2Array

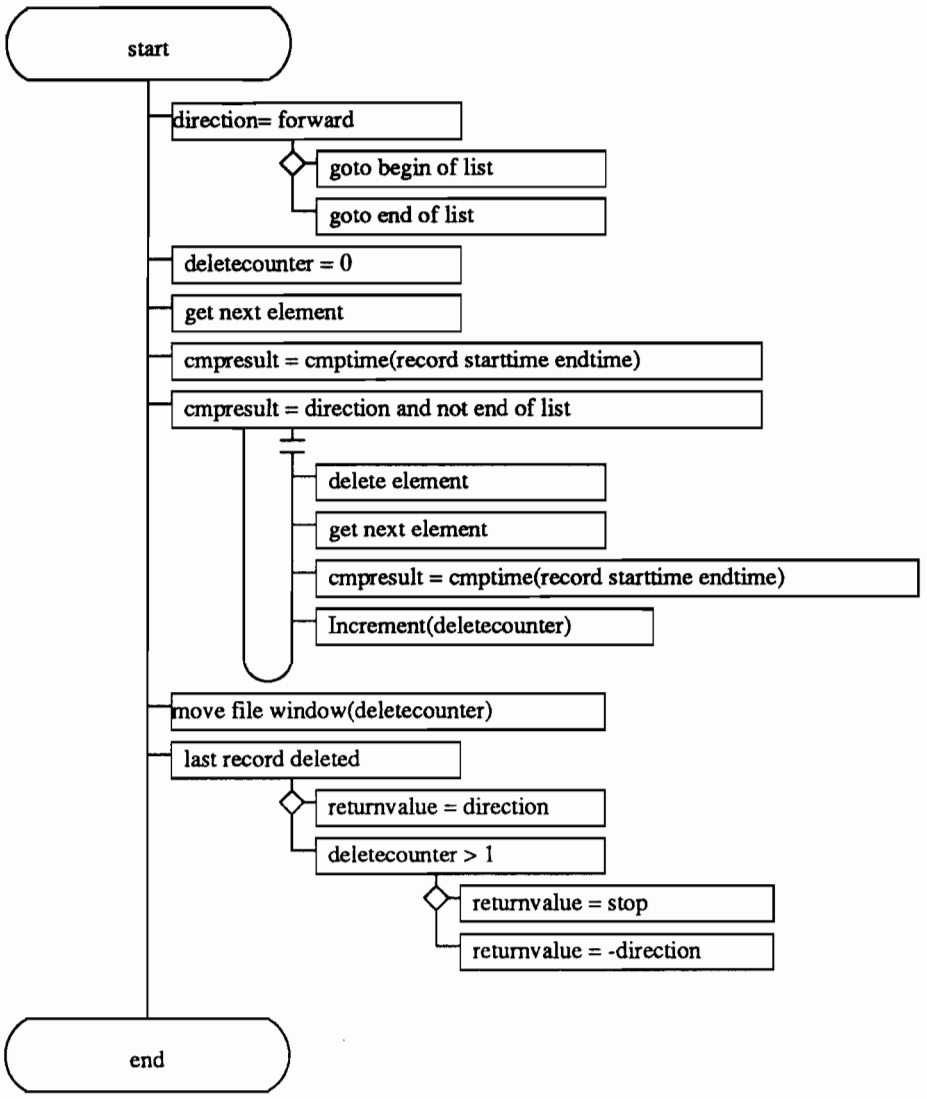


Figure 7-9: checklist

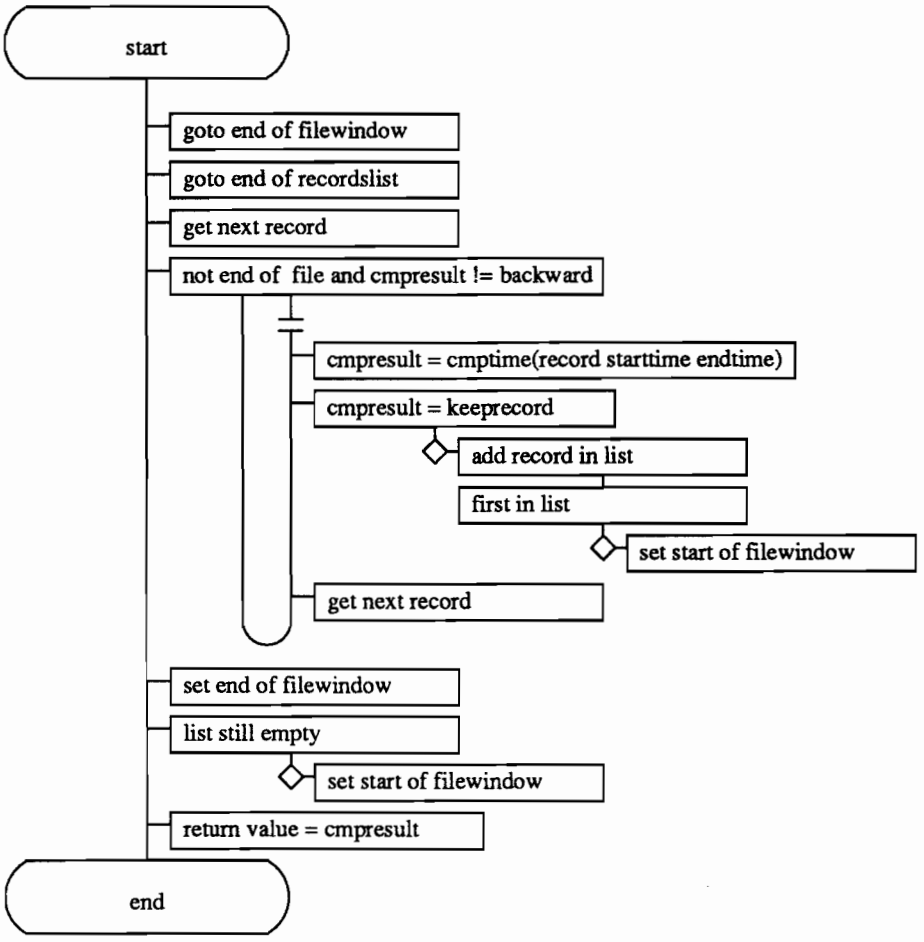


Figure 7-10: readforward / readbackward

input : query identifier and fieldnumber
output : a pointer to the array and the length of the array

7.7 The commands added to TCL for the database

The functionality of the database and preprocessor described in the previous pages is made available in TCL by adding commands. These commands are all preceded by the word database, so it is easy to seperate the commands from ordinary TCL/TK commands. The commands are:

dbinit When this function is called, the file with the record type definitions will be read, also all the data which can be derived from this file (like record size and offset) will be calculated.

openfile This function opens a data file.

query This function will read a part of an opened file.

closefile This function closes a file all the queries related to this file will be closed as well.

closequery this function closes a query and frees all the memory occupied by the data related to that query.

tomatlab From every record read in the query, the specified field is send to Matlab.

For debugging :

printlist prints the control structures related to the indicated linked list

printlinks prints the control structures per list. It shows all the lists created, what files are open, which queries are made etc.

The parameters of Tcl commands are checked by a function. This function will generate error messages to Tcl if a parameter is not correct . If all the parameters are correct then the function which actually does the work is called. Sometimes a small part of the functionality of a command is also implemented. The checking is as strict as possible to be able to detect bugs more easily. This prevents as much as possible that a wrong parameter generated by a bug in a Tcl script generates an during the execution of the code. Al these functions are in the module database which provides, in this way, the functionality of the database to Tcl.

Chapter 8

Preprocessor design

The purpose of the preprocessor is to convert many formats into a simple standard format. For this the following functionality is required:

- conversion from ASCII to binary
- perform an endian conversion on the input files, if necessary
- convert time to the internal format of seconds and microseconds
- split a file with multiple recordtypes into multiple file.

The preprocessor will be able to read any file which has records that meet the specifications in the record definition file. When reading from a file there are several cases as indicated below:

- ASCII files
  - with record identifier in first field
    - multiple recordtypes
    - one record type
  - no record identifier in first field
- binary files
  - little endian
  - big endian

Before the program can read a file it must know the following: a list of record types, whether the file is binary, whether the file little endian or big endian, whether the first field of a record is used as a record identifier and the timestep if the file has impicit time.

This information might be specified by the user. Another option is to add a header to the file, which is easier for the user and prevents mistakes. However not all measurement programs will write a header. If a file contains no header, the user will have to describe what is in the file.

The preprocessor uses the fieldtype definitions to perform the conversion. With the following three conversion routines most conversions should be possible:

- A conversion routine to convert a string to a an internal field
- A conversion routine to convert a group of bytes to an internal field
- A correction calculation function.

The conversion from string to an internal field is obvious. It is required when reading from an ASCII file.

The conversion from a group of bytes to an internal field is required for endian conversion and type conversion when reading from a binary file. Type conversion is required to convert an incremental time which is stored as an integer number into a time type.

The correction calculation applies to fields which depend on other fields. For instance, implicit time, the timestamp of the current record has to be derived from the previous record and a constant timestep. For correction calculation, the previous record, the current record and a constant record are available. The constant record is the record which holds constant parameters which are not in the current or previous record, like the timestep used for incremental time.

When the conversions are done, the converted record can be written to a file. Which output file it is written to depends on the recordtype since each output file can only contain one record type.

The data structure which is needed to preprocess a measurement file

- header present or not
- name of doc file
- name of input file
- start time
- end time
- list of record types in the file
- time type: implicit, incremental or ordinary
  - timestep (in case of implicit time)
- binary or ascii file

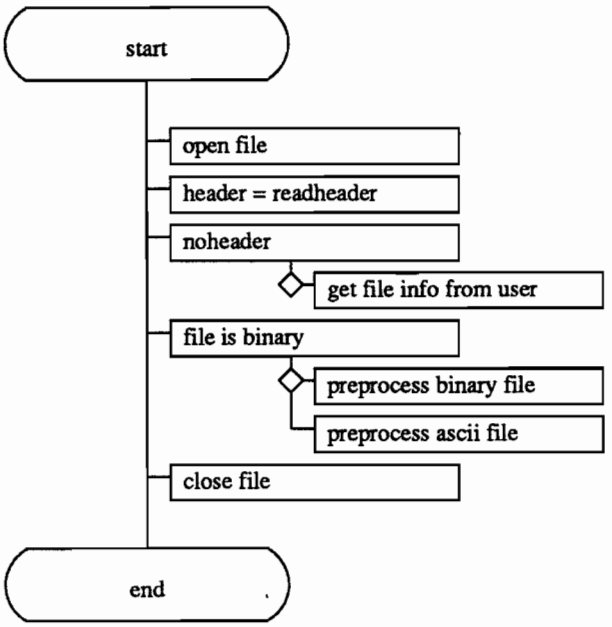


Figure 8-1: preprocess

- unTyped (in case of ascii file whether or not the line starts with a record type)
- endian big or little

name : preprocess
input : input filename, output directory
output : output files are created, file info is send to the library
manager and result indiciates whether the process was successful

name : readheader
input : a file
output : information on that file

name : preprocessAscii
input : input file, output directory input fileinfo
output : output files are created, file info is send to the library
manager and result indiciates whether the process was successful

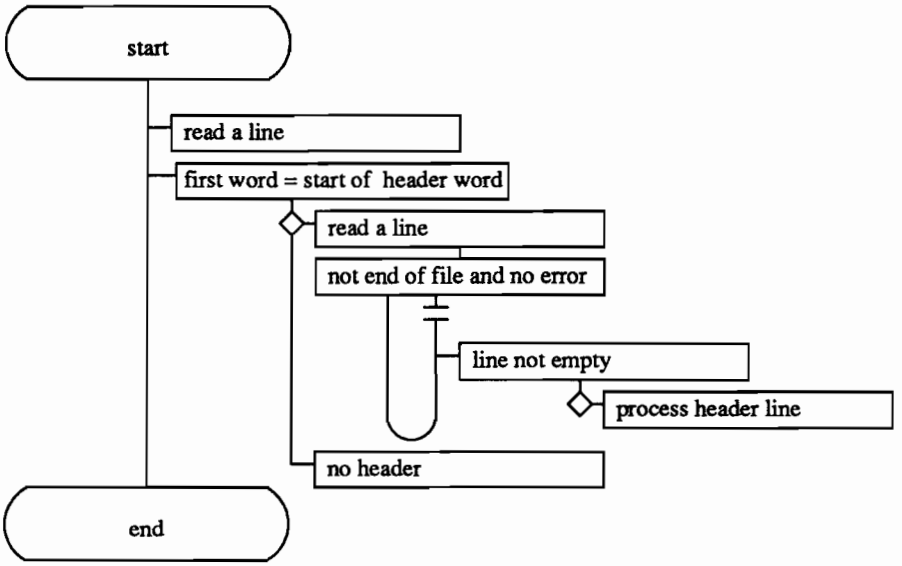


Figure 8-2: readheader

name : preprocessBinary
input : input file, output directory input fileinfo
output : output file is created, file info is send to the library
manager and result indiciates whether the process was successful

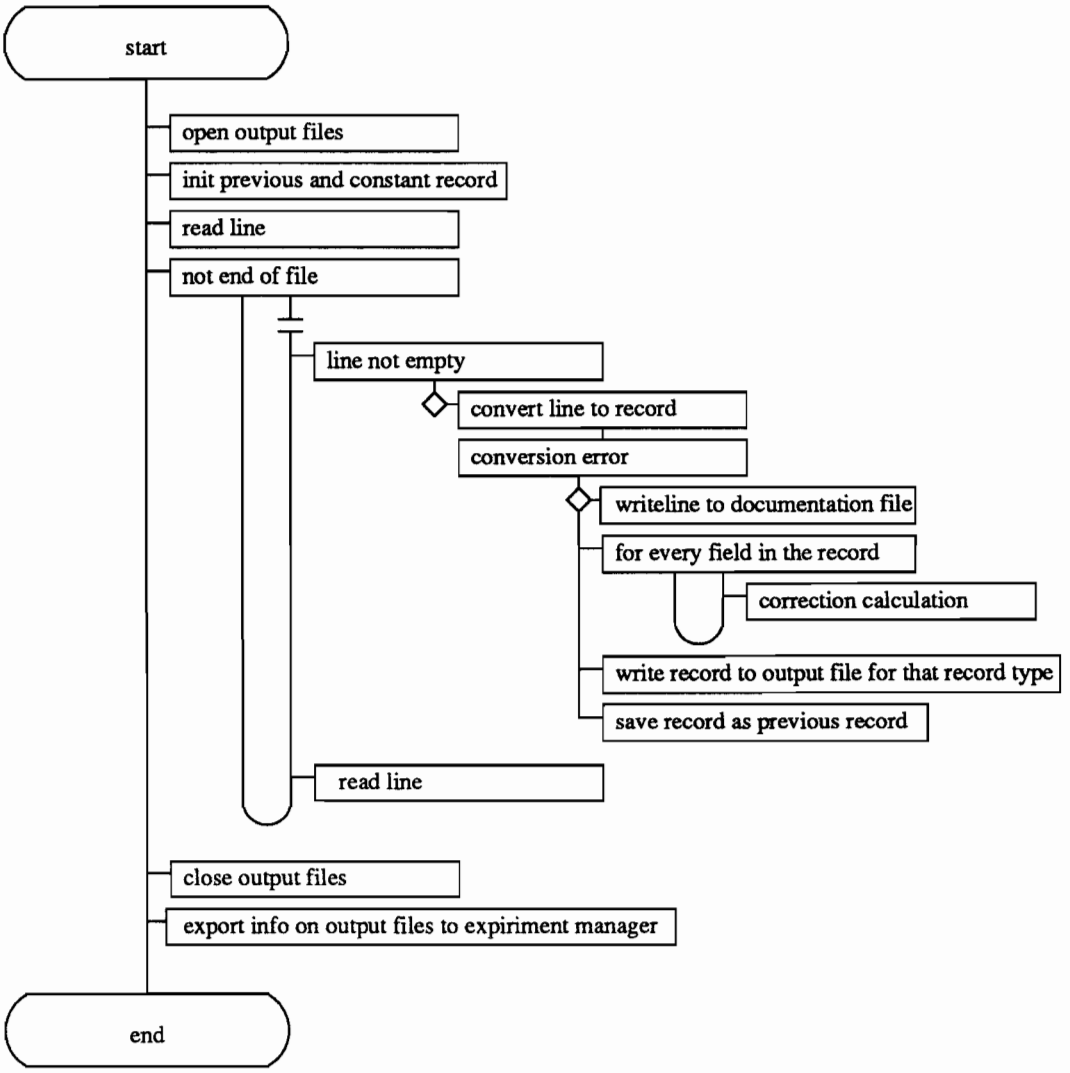


Figure 8-3: preprocessor

Chapter 9

Experiment Manager

Since there is a need for clustering databases together, which belong to one experiment, there is a need for a tool which takes care of the administration. If an extra tool is clued into Netalyse anyway it also would be nice if one could store a documentation file which describes the experiment. This last option is in perticulair very important because when one does an experiment the files that are generated are numerous. It's also nice if there is a possibility to generate a documentation file per file which is converted to internal database format.

9.1 Analysis

Resulting from the specifications there is a need for some global experiment control functionality, documentation file control, and database file control. The basic goal is to parse the information related to an opened experiment to the gui. There should also be a couple of functions available to the preprocessor which allow the preprocessor to write an experiment.

9.2 Design

Each experiment gets it's own description-file and it's own dirtectory. The experimentmanager should manage description files which contain all the information which is contained in an experiment directory. Each data-file gets it's own information block within the experiment file. Since the preprocessor creates any new data-file which belongs to an experiment, the preprocessor should have access to a function which appends the information of that file to the experiment-file. The only funcionality needed by the preprocessor is a function which takes care of that :

- Write datafile info to a experiment-file

The GUI needs access to the following fairly obvious functions :

- create an experiment
- destroy an experiment
- open an experiment
- close an experiment
- get the documentation file on the experiment
- get documentation files on a specified data-file
- get data files contained in an experiment
- get info on specific data file

Chapter 10

Various utilities needed in Tcl/Tk

10.1 Analysis

There are several routines which are not easy to implement directly in Tcl/Tk for several reasons. There was a need for two routines which could be implemented easier in c than in Tcl/Tk because the subroutines needed for the routines were already available in c.

10.2 Design

To simplify the scale in Tcl/Tk which is used for selecting, the scale is made to work with percentages instead of real date and time variables. This makes the displaying in Tcl/Tk easier. Since a lot of conversion routines already were needed for the database-core it was easier to write this routine in C. It can make use of the existing routines. The only adapation required is the incorporation of the data-types used by the date-time conversion routines. Another utility which is placed in the utility extension is the reading of the user-specified defaults. Since the tool has to be multi-user this also lead to a slight extension of the file-io utitlity. There was a need for tilde-expansion (The ~ is used to indicate homedir references ~rlaan = homedir of rlaan, ~/pathname = own homedir.) This part is build into file-io without adding any extra funcntionality for the user. If one opens a file, the filename is automatically scanned for a tilde. When file-io was 'fixed' the function itself only had to read the file and set the variables used in the tcl-domain accordingly. The last function could also be implemented as a part of the experiment manager, because it's mainly concerning stuff related to the experiment manager.

functionality

- Reading of the default places to search for files.

- percentage to time conversion.

10.2.1 Function specifications

name : readDefaults
input : -
output : -
note : this routine sets certain fixed variables in the Tcl/Tk domain.
(names are hardcoded)

name : prc2Time
input : beginTime, beginDate, endTime, endDate, percentage,
TclVariableName4Time, TclVariableName4Date
output : -
note : this routine sets the specified variables in the Tcl-domain.

10.2.2 Tcl/Tk command specifications

name : util readdefaults
input : -
output : -
note : this routine is called during initialization, normally a user doesn't have to bother.

name : util prc2Time
input : beginTime, beginDate, endTime, endDate, percentage, TclVariableName4Time,
TclVariableName4Date
output : -
note : specified variables are set accordingly..

Chapter 11

The Grapical User Interface GUI

11.1 Designing GUI's

Designing a GUI is different from designing an ordinary program. First of all there are very many ways in which the program can be used. The user will enter data, go back, change things and wants to see the results immediately. Furthermore the user can make mistakes or perform all kinds of unexpected operations, which may not result in errors.

The GUI design has to depend on the anticipated use of the program. So the first thing to do is to define possibilities the user interface will give the user and the way the user will usually work with the program. The standard operations have to be very easy to perform. This can be hard for a program which is completely new and very interactive. For instance, a first there was no event plot, which is actually just a special case of a xy-plot. It could be generated by an inventive user but it was not easy. Since event plots appeared to be important and were used often, an event plot was added as a graph type. Now it can be generated with a click of a button.

The way the dialog with the user works is also important. It is possible, for instance, to pop-up an message window every time an exception occurs, but this is very tiring so it has to be reserved for serious errors which need the intervention of the user. Ordinary messages can be displayed on a status line.

11.2 Programming in Tcl/Tk

Programming in Tcl/Tk is not much different from programming in any other interpreter language. But since it is used for GUI's some things are unique.

11.2.1 Lay out

A GUI written in Tcl/Tk is made up of basic elements called widgets. Some examples of widgets are buttons, listboxes, scrollbars, entry boxes, menu bars, text boxes etc.

These elements can be grouped using frames for lay-out purposes. Within the frame the lay-out is done relatively to the frame. A widget can be put at the top, bottom, left side or right side of the frame it belong in. The frame it self can be part of another frame or the main window. The frame system makes it very easy to make comlicated lay-outs. Designing the lay-out in Tcl/Tk is very easy. After a couple of drawings on paper you can type the code for the lay-out and almost interactively modify the lay-out until it looks good. Coding the functionality is bit more complicated.

11.2.2 Functionality

The widgets all have predefined functionality. The standard functionality of an entry widget for instance are entering and modifying text, selecting text and copying text from other windows, scrolling in case the text gets to big to fit completely in the entry. Besides this an application can bind it's own functionality. For instance when the user enters the return key in the entry widget, a function can be called which processes the input. When multiple entries must be filled in, the tab key can be bound to a function that causes the the cursor to go to the next entry.

A Tcl/Tk program can be seen as a group of eventhandling routines. Each time the user causes an event (move the mouse, click on a button etc.) one of these eventhandlers is called. When multiple event handling routines use the same data, this data has to be accessible to all of them. It's tempting to let every eventhandler access the data it needs directly, but this will lead to unexpected side effects and unsystematic code.

If there are several defined functions which work on the same data but can be called from several event bound functions, then it is best to make a module which performs the defined functions on an internally defined global variable. Which may not be accessed outside that module. This way you can prevent unexpected modifications to the global variable, because the operations on the global variable are always done by one of the functions in the module. It also makes it easier to modify the data structure as long as the existing functions don't change. For instance, if the global variable was a list but it would be faster if implemented with an array. The provided functions can stay the same but the internal structure is changed.

It is possible to pass the name of a variable to a function. This way the previously described module can act upon multiple instances of a datastructure. Passing a name can be compared to passing a pointer in C though there are some small differences. In Tcl the function has to know at what level the variable was defined. The function can access the variable with that name on global level, but also in the scope of the calling routine. It can access any variable in between the current level and the global level.

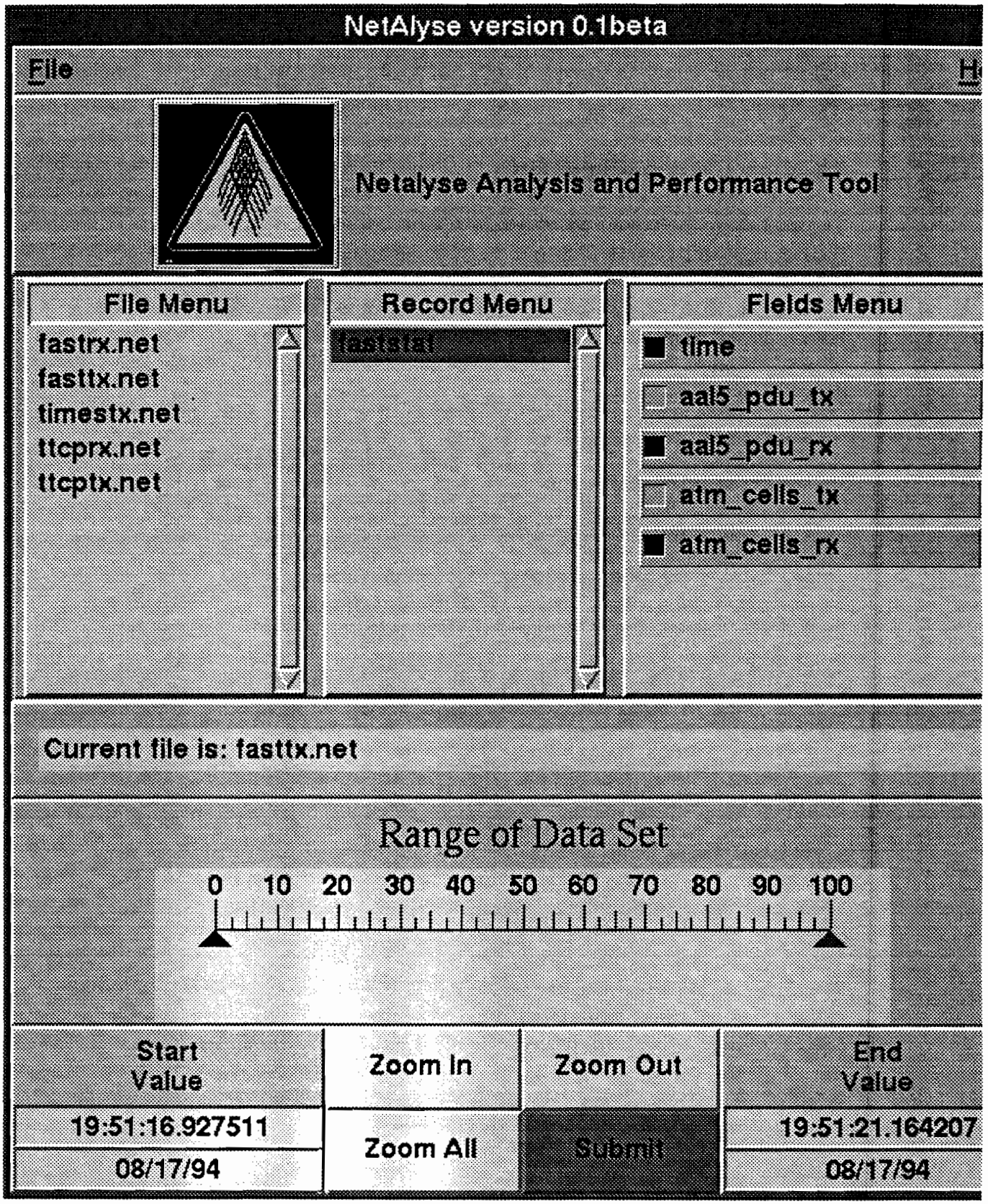
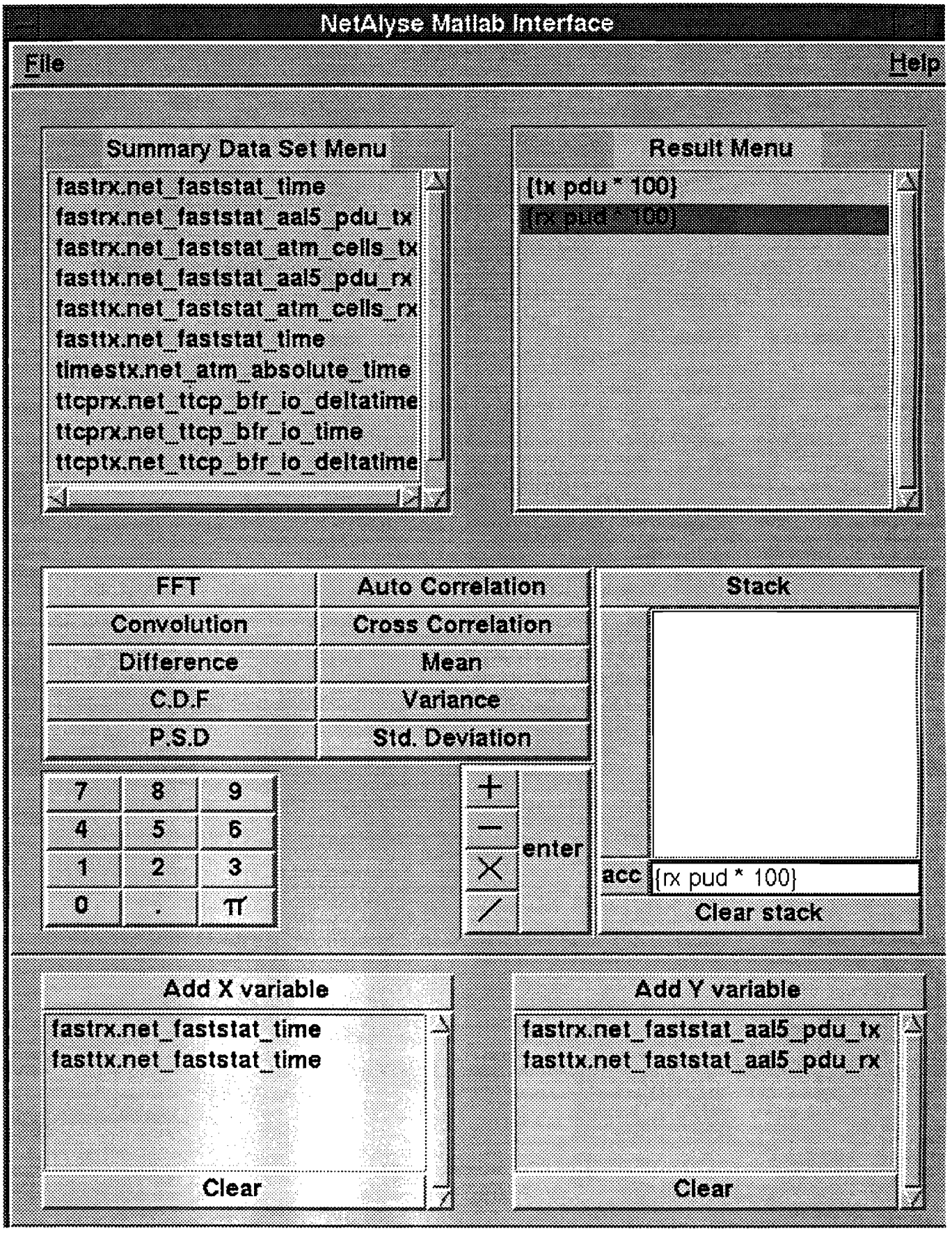


Figure 11-1: database selection gui


NetAlyse Matlab Interface
File
Help
Summary Data Set Menu
fastrx.net_faststat_time
fastrx.net_faststat_aal5_pdu_tx
fastrx.net_faststat_atm_cells_tx
fasttx.net_faststat_aal5_pdu_rx
fasttx.net_faststat_atm_cells_rx
fasttx.net_faststat_time
timestx.net_atm_absolute_time
ttcprx.net_ttcp_bfr_io_deltatime
ttcprx.net_ttcp_bfr_io_time
ttcptx.net_ttcp_bfr_io_deltatime
Result Menu
{tx pdu * 100}
{rx pud * 100}
FFT
Auto Correlation
Convolution
Cross Correlation
Difference
Mean
C.D.F
Variance
P.S.D
Std. Deviation
Stack
7
8
9
4
5
6
1
2
3
0
.
π
+
−
×
/
enter
acc
{rx pud * 100}
Clear stack
Add X variable
fastrx.net_faststat_time
fasttx.net_faststat_time
Clear
Add Y variable
fastrx.net_faststat_aal5_pdu_tx
fasttx.net_faststat_aal5_pdu_rx
Clear

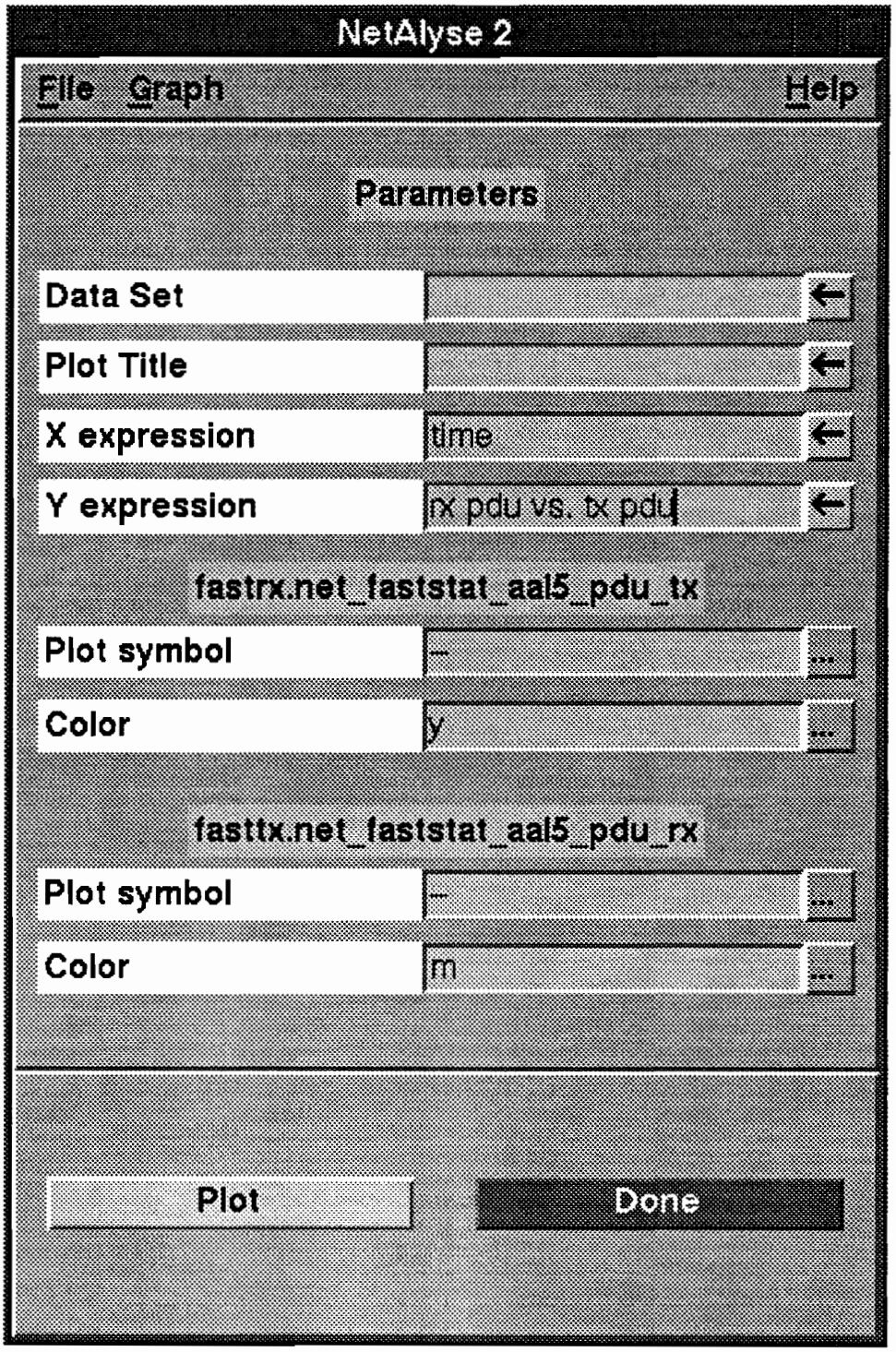


Figure 11-3: plot control gui

- the type (result or input)
- the formula if the variable is a result

The five display lists will contain:

- the descriptive name
- the index to the Matlab list.

A couple of standard functions can be defined for working with the Matlab variables list: A function which returns a new unique name which can be used for a variable in Matlab

name: getNewMatlabName

input : -

output : a unique name for a variable in Matlab

A function which lets the user add an input to the Matlab list

name : addInputToList

input : information on input variable

output : reference for this variable and the Matlab name

A function which lets the user add a result to the Matlab list

name : AddResultToList

input : the Matlab name and the formula

output : the reference for this variable and the Matlab name

A function which returns the Matlab name

name : getMatlabName

input : reference to the variable

output : the name of the variable in Matlab

A function which returns the information on an input variable

name : getUserData

input : reference to the variable

output : the extra data the calling function needed on the variable

11.4.1 functionality of the calculator

The calculator will consist of a numeric keypad, function buttons, operator buttons, a stack and an accumulator in which data can be enterd. When a number key is hit the corresponding digit must be stored in the accumulator just like in an ordinary calculator. When a constant or variable was previously entered, it is not usefull to add a digit to the string. Instead the accumulator is cleared. To do this a flag is required which indicates the type of data in the accumulator.

name : enterDigit

input : name of entrybox, a digit

output : the digit will be added to the accumulator

The variable in the accumulator can be entered on the stack when the user hits the enter button. The stack is a list with a descriptive name and a reference to the Matlab variables list. Numbers and constants however do not have a reference to this list. These stack entries can be used directly in a formula. To indicate these entries the reference to the Matlab variables list is set to a reserved value.

When an operator button is hit, a string like "var1 + var2" must be created. Var2 will be the first variable. Var1 will be the second variable on the stack. This is the way real reverse polish notation calculators work.

name : matlabOperator

input : the operator, the number of inputs and the list of inputs

output : the formula as a string

When a function button is hit the top n variables of the stack are popped and put in a function expression in reverse order. For a function taking two variables as arguments this will be: "functionname(var1, var2)" Var2 is the first variable popped from stack and Var1 is the second variable popped from stack.

name : matlabFunction

input : the functionname, the number of inputs and the list of inputs

output : the formula as a string

Just the string for the formula is not enough. First it has to be checked if the stack holds enough variables. Then the formula can be generated. With the formula and a new Matlabname an equation can be generated. The equation is send to Matlab. When no error returns the result can be added to the matlab variables list. The name of the result is the name of the function so the user can keep some track of the contents on the stack.

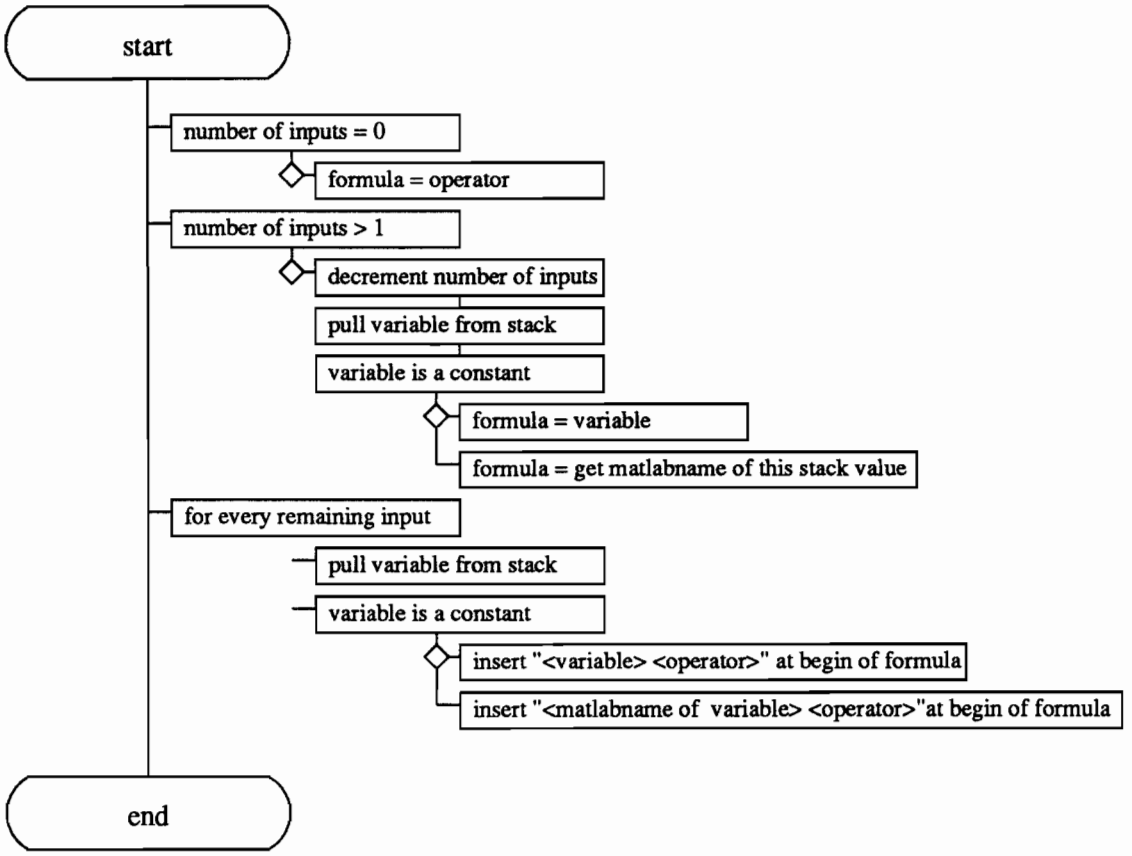


Figure 11-4: matlabOperator

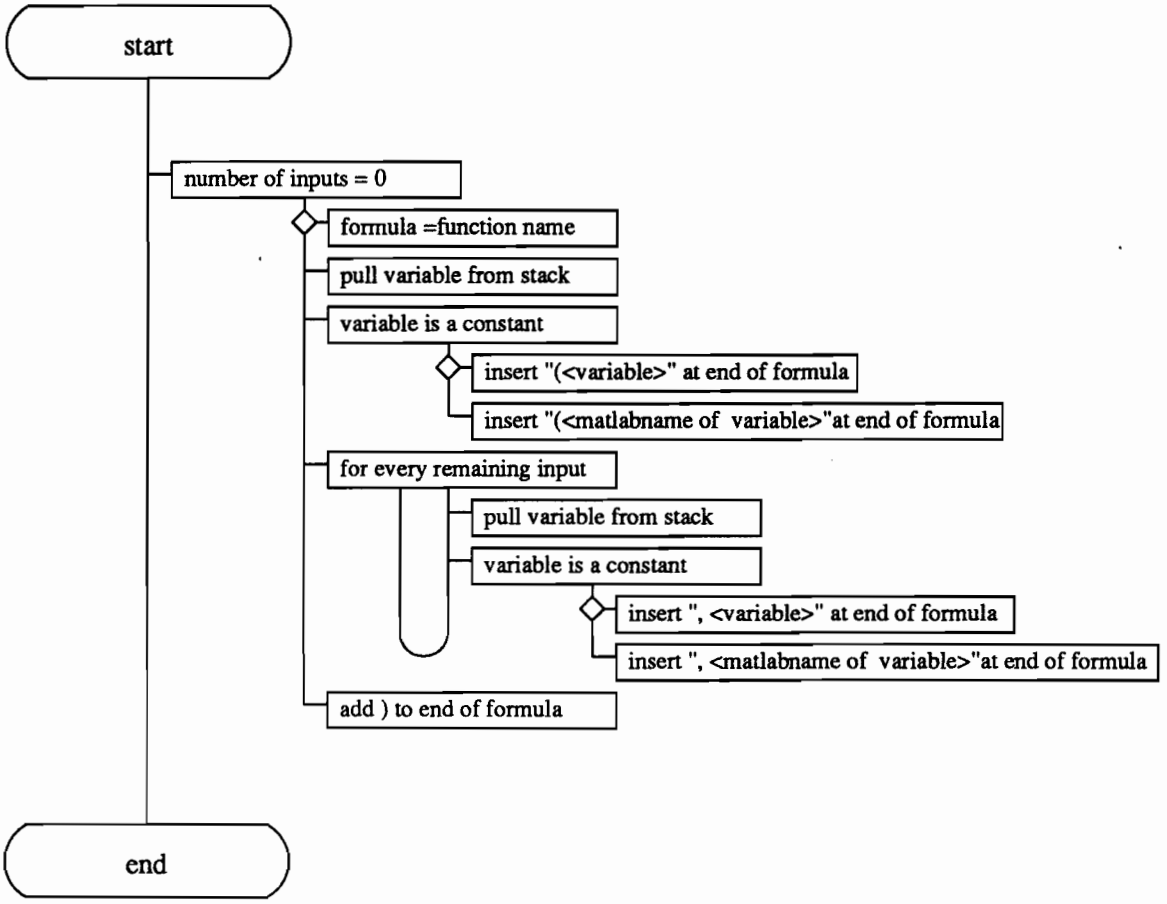


Figure 11-5: matlabFunction

name : operation

input : the stackname, the stack listbox name, the type of the operation either operator or function, the name of the operator/function, the number of inputs/operands, the list of output names.

output : a new variable will be created in matlab containing the result of the calculation.

If the user clicks on a stack variable an input window will be popped up which asks the user for the name of the result It defaults to the name on the stack which was the name of the operation.

name : setResult

input : the name of the stack, the name of the stacklistbox, the name of the result list the name of the result listbox

output : the current selection in the stack is stored as a result in the result list

To get a unique name the name has to be compared to the existing names and if there is a duplication then a new name must be entered by the user.

name : addUniqueToListbox

input : listboxname, the default name to add

output : the new name

11.4.2 Selecting a data set for plots

The user can select a data set for a plot in the plot x and plot y listboxes This is done when the user hits the x-add button. Then the current selection in the input or result listbox (by default one excludes the other) is copied to the x plot listbox. This works the same for adding data to the y plot listbox.

name : addvar

input : -

output : the current selection from either the input listbox or the output listbox

When the dataset has been selected the user can select the type of plot he wants. The same plot function is called for all the plot types. The difference is in the parameters with which they are called.

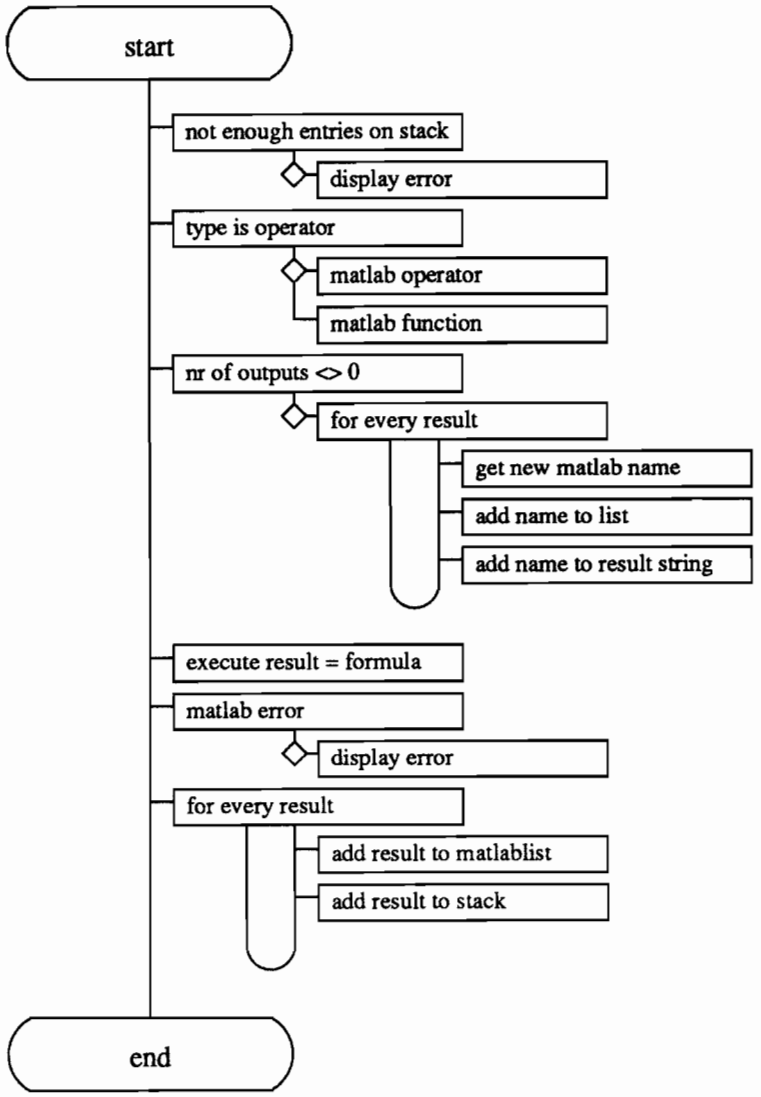


Figure 11-6: operation

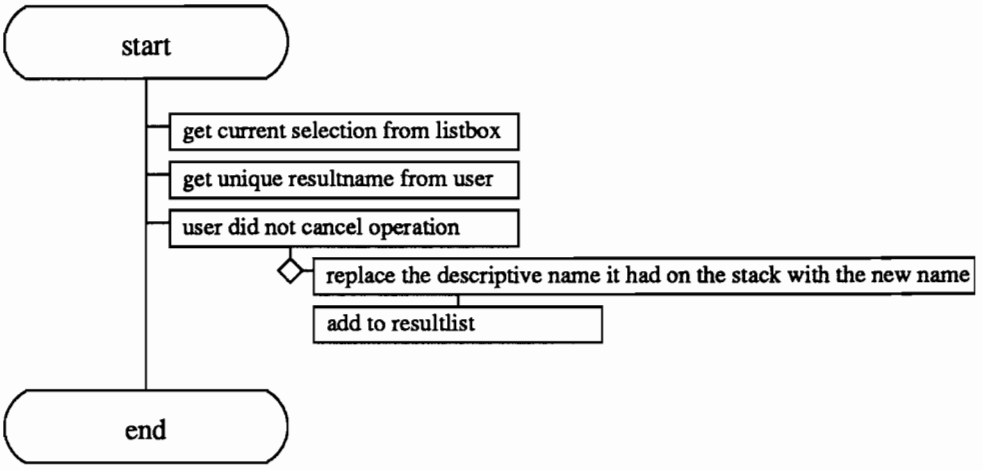


Figure 11-7: setResult

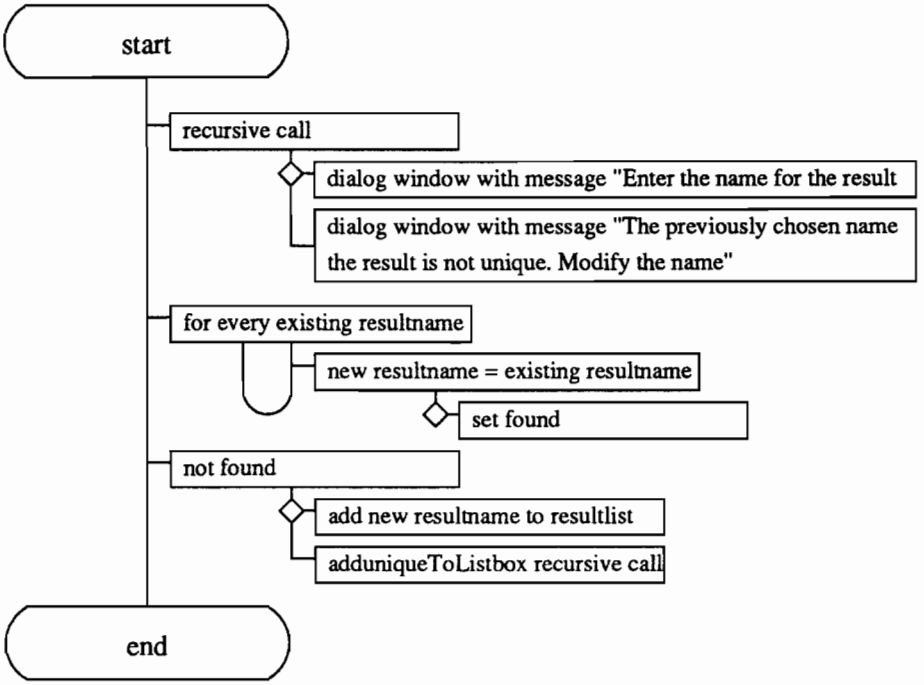


Figure 11-8: addunique

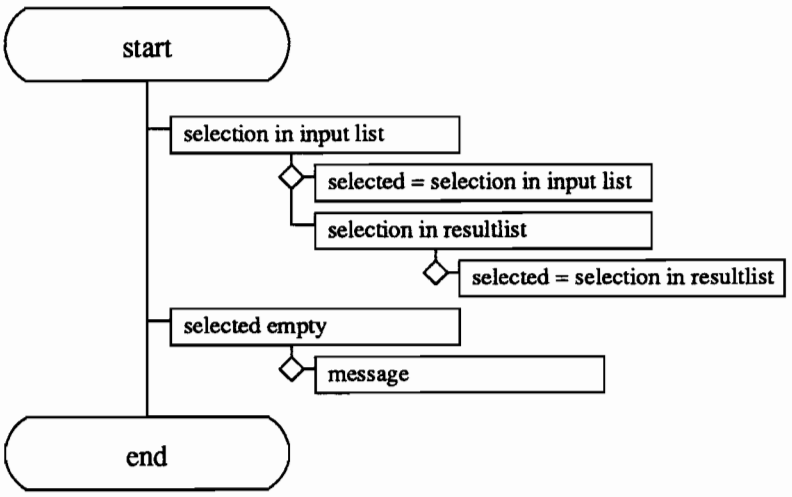


Figure 11-9: addvar

Chapter 12

Manual

The program is not extremely complicated and the GUI makes it easy to use. In this chapter the use of the program is described because the online help is not yet available. It will describe the existing version of the GUI, since the new version with support for experiments still has to be designed.

12.1 Selecting data

When the program starts up the listboxes will be empty. What the user should do then is click on the file menu and select the item "select database". This will pop up a window in which the user can select files by double clicking on them. When all the files are selected the database selection window is dismissed by pressing OK.

When this is done the file menu will show the selected database files. The user then must select each of the files and click on records and fields he is interested in. For the file the user selected he can specifiy a time interval by moving the triangles along the range selector. If the resolution of motion (limited by the size of the pixels) is not enough then the zoom in button can be pressed and the scale will be magnified. When the this is done for every file, the submit button can be pushed.

12.2 Calculation

The calculation window is popped up in response to the submit button. The selected data will be listed in the summary listbox. The data will have names that are composed of the filename, the recordtype and the fieldtype. The items in the summary listbox and other listboxes will be called variables.

A double click on any variable in the summary listbox will put that variable in the accumulator of the calculator. When the enter button of the calculator is pushed, the variable is copied from the accumulator to the stack. If the accumulator is empty, the first variable on the stack is copied. Calculations are made only on the data on

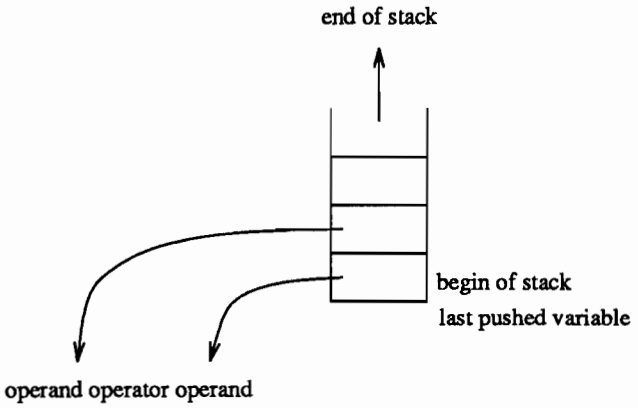


Figure 12-1: Order of operands, reverse polish notation

the stack, not on some data that might be just in the accumulator. When the desired variables are put on stack, a calculation can be selected. This can be either a function or an operator. The order of the operands used with an operator is illustrated in figure 12-1 The order of parameters for a function is the reverse of the order of the operands for an operator. The parameter last pushed on stack is the first in the parameter list of the function. The currently implemented operators are: + - x /

These operators work element by element. If one variable is a scalar and the other a vector, then the operation will be done with the scalar and every element. The currently implemented functions are:

- Fast Fourier Transform (FFT)

input : array of function values and an array with time stamps when these samples were taken

output : array with amplitudes and an array with frequencies

- Auto Correlation

input : the array of which to take the auto correlation

output : an array of correlation values and the lag to plot it against

- Cross Correlation

input : two arrays of the same size

output : an array containing the result of the cross correlation

- Convolution

input : two arrays of the same size

output : an array containing the result of the convolution

- Difference

input : array to take the difference of Xn = Xn+1 - Xn

output : the result of the difference, the last value is doubled to make the array the same size as the source.

- Mean

input : an array

output : a number that is the mean of the array

- Variance

input : an array

output : a number indicating the variance of the array

- Power Spectral Density

input : an array of function values and an array with time stamps when these samples were taken

output : array with the square of the amplitudes and an array with frequencies

Example of a calculation :

Dividing two input variables, variable1 / variable2, is done as follows:

- Double click on input variable1.
- Click on the enter button of the calculator.
- Do the same for the variable2.
- Click on the divide button and the answer called divide will replace the two input values on the stack.

To store this value double click on the stack variable divide. A window will pop up asking for a name for the variable. It will default to divide, but when multiple divide results must be stored, a unique name must be entered.

12.3 Selecting data for plots

Three types of plots can be generated:

- event plots
- histograms

- xy plots

The first two types only require x-data and ignore the y data. Which data is plotted in a graph is specified in the x and y plot lists. To add a variable in these boxes just select a variable in the summary or result list, then press the add button in the x or y plot list. When the desired dataset is selected, a plot can be made.

A histogram can be made by pressing on the histogram button. The number of bars is specified in the entry just above the histogram button.

12.4 Modifying the plot parameters

When a plot is chosen a window will appear which will let the user modify the graph. The data set generating the graph is specified, this can be a filename or expiriment description. The plot title, which indicates what the plot means can be entered. The description for the x-axis an y-axis can be also be specified. Per line in the plot the line style and color can be specified. When the plot button is hit, the plot will be drawn or redrawn after the specifications are changed. The Graph entry from the menubar gives the user the following options:

Print : print to the printer

Add : add text and a legend to the graph

View : modify the scale and the grid of the graph

The done button closes the graph and the controlling window.

Chapter 13

Todo list

Required modifications to Tcl to be up to date with the C-functionality

13.0.1 expiriment level

In the C-code support is added for an expiriment level. This expiriment level is not yet implemented in the Tcl part of the program. The expiriment layer is important because it will allow the user to keep better track of the datafiles. With the few files we have now it is already getting difficult to keep track of which file contains what. Also the environment in which the expiriment was taken must be descibed. This discription might be standardized. This will allow searches on specific expiriments. If it is not standardized, this can simply be implemented as a text file which the user can enter when he opens a new expiriment.

This will not change the GUI much. The file list will change in an expiriment list, the record list will change in the file list and the field list will stay the same. An open expiriment option has to be added to the file menu. This will result in a window which will let the user add files to the expiriment.

13.0.2 preprocessor support

The files will be added to an expiriment after they have been preprocessed. The preprocessing has been seperated from the reading of different files for performance reasons and a flexibility. The Tcl part does not support the separate preprocessor yet. When preprocessing a file it will be opened and the header will be read. If no header is read, from the C-code a window can be called which asks the user for information on the file.

13.1 Possible improvements

13.1.1 In the C-code

The C-part of the preprocessor may be extended with a standard text parsing utility which solves problems like skip the n-th line, combine two lines together to get one record, skip all the lines which contain a specified string. These functions are available from standard text parsing functions and this would greatly extend the ability of the program to read irregular file formats.

13.1.2 Tcl in general

Colors

Change colors so the current selection is readable on a black and white monitor. Now on a black and white monitor the selected menu item and the selection in a listbox are not readable.

Style

The system used for relief is not the same for every window of the program.

Variable names

The naming of variables might be changed to a better system and the user might want to specify a name for the variable. The variable names are now made up out of the combination file_record_field. The records and fields also contain underscores. Better readability can easily be achieved by replacing the underscores by other characters.

13.1.3 Selection GUI

Reducing the mouse clicks required With a double mouse click a file can be selected from the file menu, then a double click on a record gives the fields in that record. The double clicks can be replaced with single mouse clicks. When a file is selected the first record type can be selected and the fields in that record type can be displayed without other actions.

Deselection of data

With the current implementation it is not possible to deselect data To save memory space this might be needed, when large files become available.

Entering start/end time and date

Entering time and date in an entry to select what part of the file to read.

Multiple datasets from one file

The C-code supports multiple parts to be read from one file. This might be usefull if the user wants to compare a part of a big file with another part of the same file.

Reuse of previously read data

Currently the data from the files is read when the submit button is pressed. The second time the same data is read, the previous data is discarded and all the data is reread. This is not necessary. The queries can be recalled with the query number returned on the first call. This will make the performance better since no data is read if the range of data is not modified. When the range is modified only the part which does not overlap the previous range is read.

Reduce data in Matlab

When the submit button is hit, the data is send to Matlab. Every time new data is read it is put in different variables. This uses unnecessary memory, because the old data is stil in the memory of Matlab. When the data is put in the same variable, the old data will be discarded. When data is reused as described above, then the Matlab variable name can be linked to a query identifier, so data from the same query ends up in the same Matlab variable.

13.1.4 Calculation GUI

User specified functions

A function or operator in the program can be completely specified. The user might generate such a description with the help of the GUI. This way the user can use the calculator to work with his own scripts. It would be very convienient if the user defined functions could be stored in a configuration file.

Recalculation

When the user has done some calculations on data and then selects an other part of the same file he will probably see the result of the same calculations. If the points "reuse of previously read data" and "Reduce dat in Matlab" are designed, then a very simple recalculation function can be made which executes all the formula's in the matlab list.

Discarding data

When the user does not need certain results anymore he might want to discard them. When the result is used to calculate another result, then it may not be deleted. For

this the Matlab list has to be extended to contain dependencies so the program can figure out what result depends on which. Deleted variables in the Matlab list must be replaced by deleted entries. This cause the elements have to stay in the same position in the list so the reference given will stay correct. The list may be converted to an array ofcourse which does not have this problem. (in Tcl array elements can be deleted without interfering with other elements indices.)

13.1.5 The plot and calculation GUI

Remove plot boxes from calculation GUI The data for the plots is now selected in plotboxes. These boxes take up a lot of space and they are not necessary. In this alternative situation, after the plot type is chosen the plot window will pop up. Then the user can select the data he wants to plot, just by clicking on it. This is also more logical because not every application needs xy-pairs of data.

13.1.6 The plot GUI

readable time scale

The time is a number in seconds which indicates the time since the start of the day. Matlab only displays a four digits. This is clearly not enough. To solve this we generate our own time scale with 8 digits.

There are three disadvantages to this: Matlab doesn't take in account that the labels are bigger so they will overlap depending on the size of the graph. The format is not very easy to read. The third disadvantage is, when the user resizes the window containing the graph, Matlab doesn't inform Netalyse about it and will just make more labels on the line and print the old values at new positions along the axis. It will reuse the first labels when it runs out of labels.

The time can be made more easy to read with some more formatting etc. It is now done in a Matlab script so it shouldn't be hard to change. If a button is added to update the axis labels, then the labels can be regenerated after they became invalid due to a resize of the window. The tickmarks overlapping is not easily solvable. The user will just have to size the window so that the labels do not overlap. It is however with some C-programming possible to get control over the Matlab windows by using X11 calls. The size of the plot window could be set fixed and can then only be modified with the Tcl script. This script could then also regenerate the labels on the axis.

One plot control window

The program uses a plot control window for every seperate graph. It would be easier if there is just one control window which controls all the plot windows. This can be done by letting the user select the desired plot from a list of generated plots. A

nicer way would be to let the user point at a plot window and select that plot as the current. This can also only be implemented by using X11 calls directly from C.

icon names

The icon names of the plots are not descriptive, just fig1, fig2 etc. Since the user can generate plots very fast with this program, he will forget the plot number. With X11 calls it is possible to modify the icon name of a plot window. It could be set to the specified plot name for instance.

Chapter 14

conclusion

Netalyse works but it needs a lot of improvement. Suggestions are done in chapter 13 "Todo list". A couple of improvements which are not mentioned in the To-do list and might be considered are :

- Get netalyse interfaced to Octave instead of Matlab, it has almost the same capabilities as Matlab but it's shareware!
- Instead of the own build database, a consideration might be to incorporate a shareware database instead of our own. A suggestion might be "Ingres89" a relational database which can do much advanced queries. Since our own database can only do queries based on timestamps.

Originally this project started of as an Ethernet-analysis utility which was supposed to become a systemadministration-tool. Mainly to analyse network congestion and related problems. Since the specifications altered slightly it became a generic analysis tool with an emphasis on networks. The first experiments done on an 155 Mbit/s optical line were analysed with netalyse, and it showed that it was easy to use. (still buggy sometimes...) The first prototype of Netalyse needs to evolve into an extensive tool which is much more versatile than it is right now. But this needs a lot of user-input. (specifications)

Bibliography

[1] Morris L. Bolsky, "Handboek voor de C-programmeur", Prentice Hall (1988)

[2] Yourdon Inc., "Yourdon systems method", Yourdon Press (1993)

[3] Barclay, "C-leerboek", Prentice Hall (1993)

[4] Larisch, "UNIX", Data Becker Nderlands