ISDAT Programmers Guide
3. Clients

Gunnar Holmgren and Anders Lundgren
Swedish Institute of Space Physics
Uppsala Division
S-755 91 Uppsala
Sweden

May 4, 1994

Document Status Sheet			
1. Document Title: **ISDAT Programmers Guide, Clients**			
2. Document Reference Number: **CWD-IPG-003**			
3. Issue	4. Revision	5. Date	6. Reason for Change
Draft	0	94 Feb 14	New document.

Contents

1 Introduction

1.1 Purpose of the document

The purpose of this document is to provide the information needed to write client code to be implemented in the ISDAT 1.0 system. The document is intended for the programmer. General guidelines for the coding are given in [Ref. 2]. An introduction to the ISDAT is given in [Ref. 1].

1.2 Scope of the software

The scope of the ISDAT software package is to provide a flexible tool for the analysis of scientific data. The clients constitute one component of the ISDAT (see [Ref. 1]). The scope of the client software is to provide data analysis and visualisation functions.

1.3 Definitions and acronyms

Acronym	Meaning
DBH	Data Base Handler
GKS	Graphics Kernel System
ISDAT	Interactive Science Data Analysis Tool
N/A	Not Applicable
PEX	PHIGS Extension to X11
PHIGS	Programmer's Hierarchical Graphics System
TBD	To Be defined
TBW	To Be Written

2 The Client Source Code

There are three classes of ISDAT clients

1. Specific clients
2. General clients
3. Time managers

You will be guided through one example of each these classes of clients in the following sections. A convenient tool for implementing graphical user interfaces is provided in ISDAT. A special section is devoted to the use of this tool.

2.1 Specific Clients

The class of clients that are intended for use within a special project or for a special instrument are called *specific clients*. We will describe the coding of a specific client by walking through a client that computes and displays the orbit number and time for the Freja spacecraft. A full listing of the C code is given in Appendix A. The code is also intended to be used as a template for other clients.

2.1.1 Include and define files

In most clients you will need defines given in the following include files:

```
#include <stdio.h>      /* Standard input output declarations */
#include <math.h>       /* Mathematical declarations          */
#include <Is.h>         /* ISDAT declarations                 */
#include <Ui.h>         /* User interface declarations        */
#include <Db.h>         /* Data base declarations             */
```

The stdio.h is recognised as a standard C header file. The following three are ISDAT specific include files used by the IsLib, IsutilLib, UiLib, and DbLib libraries. The ISDAT system do know where to find them so you do not need to specify the full path. The corresponding calls are described in the appropriate on line manuals.

In addition to the standard definitions, it is good practice to use define statements also for the client specific flags in order to improve the readability of the code. In this example we use two define statements:

```
#define GH_LATER 0
#define GH_NOW 1
```

It is important to remember that the defines should be unique. In the example the initials of the author have been used to ensure the uniqueness of the names. Also note that upper case and underscore have been used. In more complex clients also the client specific defines should be collected in separate header files.

2.1.2 User interface definitions

In order to make coding easier, a built-in string interpreter is included in the ISDAT system that translates certain well defined strings to motif structures. It is further described in section 4. We could have chosen to include the corresponding functions directly in motif. In the example, we have constructed a string that defines the display window and associates it with the necessary user interface parts.

```
static char *basicLayoutString = "\
    layout 'forbit' basicLayoutPart=. {           \n \
        frame 'Freja orbit' {                     \n \
            text '' -c24 -r2 orbitWindowTextPart=.\n \
```

```
        }                                               \n \
    }                                                     \
";
```

Here, the first line declares the string and its identifier *basicLayoutString*. Then the string defines a outer frame labelled *forbit*, the client name in this case, and given the identifier *basicLayoutPart*. Next line defines an inner frame *Freja orbit* that consists of a *text* window with no visible text initially, 24 columns, 2 rows, and a part identifier *orbitWindowTextPart*. The (backslash)n's are not necessary, but define linefeeds that are used by the interpreter error handling to identify syntax errors. In addition to defining the window, we have to declare the parts used in the string above:

```
static UiPart basicLayoutPart;          /* top level part */
static UiPart orbitWindowTextPart;      /* text part      */
```

Then we have to enter them into an identifier map, shown below, for the information needed by the ISDAT system in the compiling phase:

```
UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(basicLayoutPart)
    UiID_MAP_ENTRY(orbitWindowTextPart)
    UiID_MAP_END
};
```

2.1.3 Main process

The function of the main process is to initialise the processing and then enter into the main loop and remain there pending callbacks. The main call utilises the standard C arguments argc and argv:

```
main(argc, argv)
int argc;
char **argv;
{
}
```

which may be used to communicate information about the desired display or other flags. A description of useful ISDAT-specific flags is given in section 3. Example: *forbit -display myscreen:0*. If no info is supplied, the display given by the environment variable $DISPLAY is used.

The declarations needed are:

```
    Display *display;
    void UpdateTimeCB();
    Database *db;
```

Note the convention to use lower case for variables and upper case lead letter for functions. The Display and Database declarations are ISDAT specific.

the first step is to initialise the user interface and isdat libraries:

```
    display = UiInitialize(argc, argv);
    IsInitialize(argc, argv, display);
```

These calls are described in the UiLib and IsLib on line man pages.

The next step is to create the user interface layout:

```
    if (!UiCreateLayout(NULL, basicLayoutString, idMap, 0))
        IsError("Cannot create the basic layout");
```

These calls are also described in the UiLib and IsLib on line man pages respectively. IsError stops execution when called. There is also an IsWarning that only warns via the *stm* and prints the string given as argument without stopping execution. However, if the user interface cannot be created, it is hardly worth continuing execution.

Next, we try to open a connection to a data base handler:

```
if ((db = DbOpen(NULL, argc, argv)) == NULL)
    IsError("Cannot connect to database %s", DbName(NULL));
```

The DbOpen call is defined in the DbLib on line manual. in this example we try to connect to the data base handler defined by the environment variable $DATABASE. This is an ISDAT defined environment variable.

So far nothing has become visible on our screen. We have to make the created window visible by the call:

```
UiMapLayouts();
```

This call is described in the DbLib on-line manual pages.

The final initialisation is to tell the *time manager* where to direct the time event callbacks:

```
IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);
```

This call is described in the IsLib on line manual pages. In this example we direct the time callbacks to the function UpdateTimeCB (see below).

Now we have done all necessary initialisation, so we just have to enter the main loop and wait for callbacks from the time manager.

```
IsMainLoop();
```

This call is described in the IsLib on line manual pages. We will now stay within the main loop until the end of the execution.

2.1.4 Time event callbacks

In the main process, we told the connected time manager to send all time event callbacks to process UpdateTimeCB(). Time events are typically when the time manager update the time or change the time interval. When the time manager sends the callback, it also provides some information about the data that it can provide for the time interval in question. This is done via arguments to the call:

```
void UpdateTimeCB(reason, tmInfo, closure)
int reason;
IsTmInfo *tmInfo;
IsPointer closure;
{
}
```

where *reason* is set to IsCR_TM_INFO when called as a result of a time manager action, tmInfo is a structure described in the IsLib on line manual pages, and closure is the value of the third argument to IsAddCallback (NULL in our example).

In a big client, the UpdateTimeCB() should only forward a request to some other function. However, in this short example, we format the whole output here. First define our variables and we make sure that we are really working with the Freja project since this is a *special Freja client.*

```
int FrejaOrbit();
char string1[64], string2[64];
int column=0, row=0;

if(tmInfo->project != DbFREJA)
    IsError("This is not the Freja project");
```

The project info, we find in the tmInfo structure provided by the associated time manager. if it is not the Freja project, we stop execution. The DbFREJA definition we find in the Db on line manual pages.

Next step is to remove old text from the text window:

```
    UiTextClear(orbitWindowTextPart,GH_LATER);
```

We do not need to update the screen now, so we use the GH_LATER flag defined above.

Now we can update the text in the text window, and we start by updating the orbit number:

```
    sprintf(string1,"orbit: %d",FrejaOrbit(tmInfo));
    UiTextChange(orbitWindowTextPart,GH_LATER,column,row++,string1);
```

The actual orbit number is computed by the function FrejaOrbit() described in section 2.1.5. We still do not need to update the screen, only the memory, so we flag GH_LATER. When specifying the text row, we take the opportunity to prepare for the next row by incrementing the row within the argument list by setting row++.

The second text line contains time info.:

```
    IsTime2YmdHms(tmInfo->start,string2);
    sprintf(string1,"time:  ");
    strcat(string1,string2);
    UiTextChange(orbitWindowTextPart,GH_NOW,column,row,string1);
```

Here we first use an ISDAT call to convert from internal ISDAT time to a readable string and put the text "time:" in front. Then we update the second text row. This is the last row, so we use the GH_NOW flag to visualise the updated text.

2.1.5 Application computations

The actual computations of the orbit number takes place in process FrejaOrbit():

```
/* FrejaOrbit returns the Freja orbit number                    *
 * If time corresponds to a time pre-launch, a negative orbit *
 * number is returned                                           */

int FrejaOrbit(tmInfo)
IsTmInfo *tmInfo;
{
    IsTime launchTime, timeInOrbit;
    double orbitalPeriodMinutes = 108.95764;
    int firstOrbitNo = 1;

    IsYmdHms2Time("921006 090939.0",&launchTime);
    timeInOrbit = tmInfo->start;
    IsSubTime(&timeInOrbit,&launchTime);
    return((timeInOrbit.s + timeInOrbit.ns * 1.0e-9) /
        60.0 / orbitalPeriodMinutes + firstOrbitNo);
}
```

Note that the time information is supplied by the time manager via the tmInfo structure. We use ISDAT calls to make time conversions. The internal ISDAT time is given to nano-seconds accuracy and in two structure members s and ns. It is recommended to use self-describing variable names at the expense of short source code.

2.1.6 Client documentation

All clients should be accompanied by a short on line manual. The format used for the on line manual follows the UNIX standard *nroff* format using man macros. For more complex clients it may also be necessary to write more extensive off line documents. For our *forbit* client the manual source code is named *forbit.1* and looks like this:

```
.TH forbit 1Sci "1.0" "ISDAT" "science clients"
.SH NAME
forbit \- computes and displays the orbit number for Freja
.SH DESCRIPTION
\fIforbit\fR computes the Freja orbit number based on the ISDAT time
returned from the time manager. It also converts the ISDAT start
time to UT and displays both alphanumerically in a separate window.
If the time corresponds to a pre-launch time, a negative orbit
number is shown.
\fIforbit\fR can be started from the Freja entry of the time manager
clients menu or from a separate window. It is stopped by closing the
forbit window.
.SH DIAGNOSTICS
If \fIforbit\fR is started under a project that is not Freja,
execution is stopped.
.SH SEE ALSO
ISDAT 1.0 Programmers Guide 3. Clients
.SH AUTHOR
Gunnar Holmgren gh@irfu.se 24 February 1994
```

The formatted page is shown in Figure 1. The formatting is made by the command *nroff -man forbit.1*.

```
forbit(1Sci)            science clients            forbit(1Sci)


NAME
     forbit - computes and displays the orbit number for Freja

DESCRIPTION
     forbit computes the Freja orbit number based on the ISDAT
     time returned from the time manager. It also converts the
     ISDAT start time to UT and displays both alphanumerically in
     a separate window. If the time corresponds to a pre-launch
     time, a negative orbit number is shown. forbit can be
     started from the Freja entry of the time manager clients
     menu or from a separate window. It is stopped by closing the
     forbit window.

DIAGNOSTICS
     If forbit is started under a project that is not Freja, exe-
     cution is stopped.

SEE ALSO
     ISDAT 1.0 Programmers Guide 3. Clients

AUTHOR
     Gunnar Holmgren gh@irfu.se 24 February 1994
```

Figure 1: Formatted manual page

2.2 General Clients

A *general client* usually has to query the data base handler about available data sets. In order to be useful this information also has to be presented to the user and the user should be given means to specify what he wants. We will illustrate this by writing a client that reads data and displays the returned specification structure. We will use the same basic structure as for the client *forbit* and will therefore limit our comments to new features.

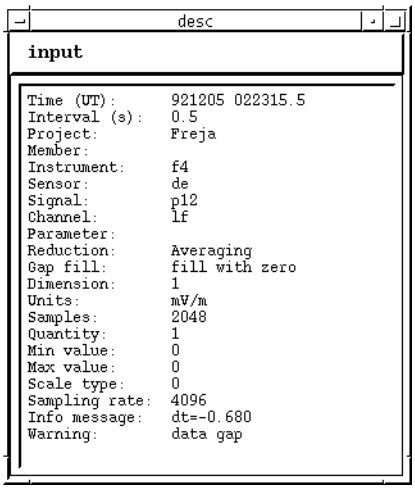

Figure 2: desc window

2.2.1 Declarations

We will use approximately the same layout of our window with the addition of a menu bar at the top that will contain an interactive menu describing available data. The string defining our window now is:

```
static char *basicLayoutString = "                    \
    layout 'desc' basicLayoutPart=. {            \n \
        menubar 'Menu' menubarPart=.;            \n \
        frame 'data structure'                   \n \
        text '' -c40 -r22 descWindowTextPart=.   \n \
    }                                            \n \
";
```

The resulting window is shown in Figure 2. The only new thing in the layout string is that we have added a menubar.

We next have to declare the DataChannelCB and enter it into a mapping table shown below:

```
/* forward declaration of callback functions */
void DataChannelCB();

/* fill in callback mapping table */
UiCallbackMap cbMap[] = {
    UiCB_MAP_ENTRY(DataChannelCB)
    UiCB_MAP_END
};
```

2.2.2 Main process

New functions of the main process is that we have to ask the DBH what the current project and member is.

```
    info = IsGetTmInfo();
```

This information is then needed when creating the input menu string:

```
inputMenuString = BuildQueryTree(db, DbQUERY_ALL, info->project,
    info->member, "input", "DataChannelCB");
```

The menu string is built using information supplied by the data base handler and communicated to us via the DbQuery call in the CreateLevel process called from the BuildQueryTree call where the menu bar is actually created.

```
if (!UiCreateMenu(menubarPart, inputMenuString, idMap, cbMap))
    IsError("Cannot create project specific menu");
```

All necessary source code to build and create the menu in supplied in appendices B, B.2 and B.3. However, we will not discuss them here.

Tell the time manager where to direct the time event callbacks:

```
IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);
```

2.2.3 Time callback

In the UpdateTimeCB we are going to request data. We therefore have to make the necessary declarations:

```
void FillDataSpecStruct(), ShowDataSpec();
float *data;
int errorCode;
```

The data description structure *dataDesc* has been declared as a global structure above as:

```
DbDataDesc dataDesc;
```

In addition to the data desc structure we need a pointer to the returned data array **data*. The DBH will allocate memory for it so we just need the pointer here. We will also need an integer errorCode.

Before requesting data, we need to specify what data we desire. For a description of the structure, see Appendix D. We do that in:

```
FillDataSpecStruct(tmInfo);
```

and by pointing at the appropriate entry of the menu. That will make a callback to DataChannelCB and set the appropriate data channel.

Now we are ready to request data:

```
errorCode = DbGetData(db,&dataDesc,&data);
```

We will save the errorCode for printout in the ShowDataSpec process if needed. The DbGetData call is the most primitive version of the ISDAT data requests. Normally you need the DbGetTimeTaggedData or the DbGetSegmentedData calls. They are described in the DbGetData on line manual.

Next step is to visualize the data descriptions returned by the DbGetData. This is done with the ShowDataSpec call.

Finally it is our responsibility to free the used data memory when no longer needed:

```
DbFree(data);
```

2.3 Time Managers

2.3.1 Time manager functions

The *time managers* are responsible for coordination of the performance of a group of clients. One or many time managers can be active at one time. A time manager should perform the following functions:

- Provide the group of associated clients with a group identity.

- Select *project* and *member*[1].
- Select time and interval for analysis[2]
- Start up other clients.

Optionally, the time managers may also include more functions like:

- Handle error and warning messages from the associated clients.
- Visualisation of available data sets.
- Start of new time managers.
- Use other data selection criteria than time.
- Automatic time stepping.

2.3.2 Time manager coding

[to be written]

3 Development, Execution and Maintenance

3.1 Installation and debugging

When developing a client, you should first, after consulting the ISDAT guru, select an appropriate source code directory in the ISDAT tree,

The Unix tool *make* should always be used for building the executable file. make depends on a *Makefile* for its execution. However, you should never try to edit the Makefile. Instead, You should write an *Imakefile*. The Imakefile is then used by *make* to build its own Makefile. In this way, all system dependencies can be taken into account automatically. For our *forbit* program the Imakefile looks like this:

```
        DEPLIBS = $(DEPUILIB) $(DEPDBLIB) $(DEPISLIB)
LOCAL_LIBRARIES = $(UILIB) $(DBLIB) $(ISLIB) \
                  $(MOTIFLIB) $(XTLIB) $(X11LIB) \
                  $(STDLIB)
       INCLUDES = -I. -I$(INCLUDESRC) $(MOTIFINC) $(XTINC) $(X11INC)

           SRCS = main.c
           OBJS = main.o

ComplexProgramTarget(forbit)
RegisterClient(forbit, $(SCIBINDIR), Freja)
```

For a new program the development sequence is the following:

1. Create the Imakefile
2. Type *imkmf* to create the *Makefile*
3. Type *make* to compile and load the program

During the client development phase, there are a few useful flags that might be helpful:

-dump to make a core dump when an error appear.

[1] This may change in future releases to fulfill the requirement to analyze several members or projects simultaneously
[2] This may actually also be done from an ordinary client in special cases

3.2 Execution

After a successful *make* you have an executable *forbit* file in an appropriate directory. The execution can now be started from the directory manually. Alternatively, since *forbit* will appear under *Freja* in a time manager menu, it can also be started via the time manager menu. In the first case, the program will search for an active *time manager* and associate itself to the first time manager found. In the second case, the client will become a child of the time manager from which it has been started.

3.3 Maintenance

For an already existing program, the updating sequence is the following:

1. Edit the source file
2. Edit the *Imakefile* if necessary.
3. Type *make Makefile* ir *imkmf* to update the *Makefile* if necessary.
4. Type *make* to complie the program.

4 User Interface Building Tool

A special tool to simplify building of Motif based user interfaces has been developped within the ISDAT system. It also connects the callbacks to the user interface parts. The use of this tool is described in this section.

Examples of implemented panels are shown in Figure 3. The corresponding source code is listed in Appendix C.

4.1 General syntax

[to be written]

4.2 Functions provided

[to be written]

4.2.1 Menu bar

4.2.2 Text window

4.2.3 Radio buttons

4.2.4 Toggle buttons

4.2.5 Scroll window

4.2.6 Graphics window

4.2.7 Edit field

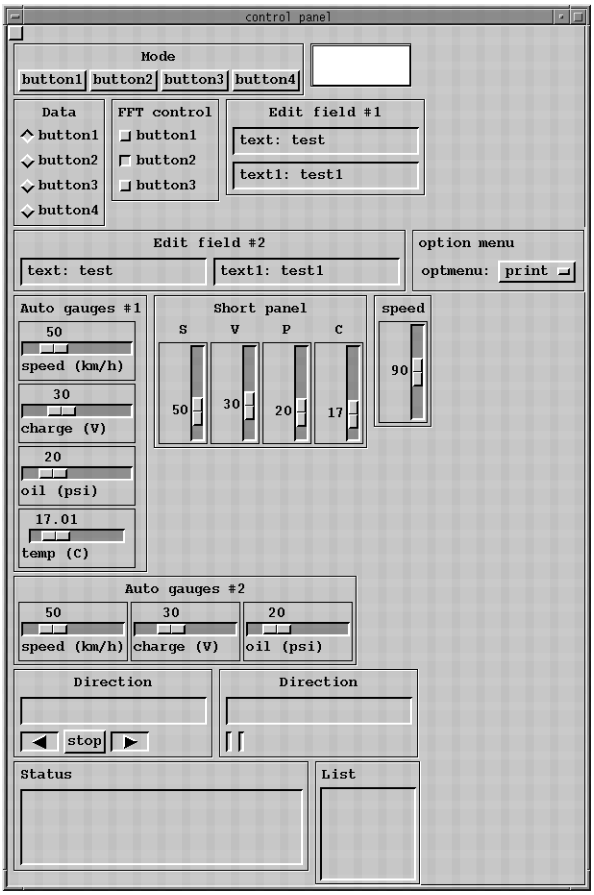

Figure 3: Illustration of panels that can be created

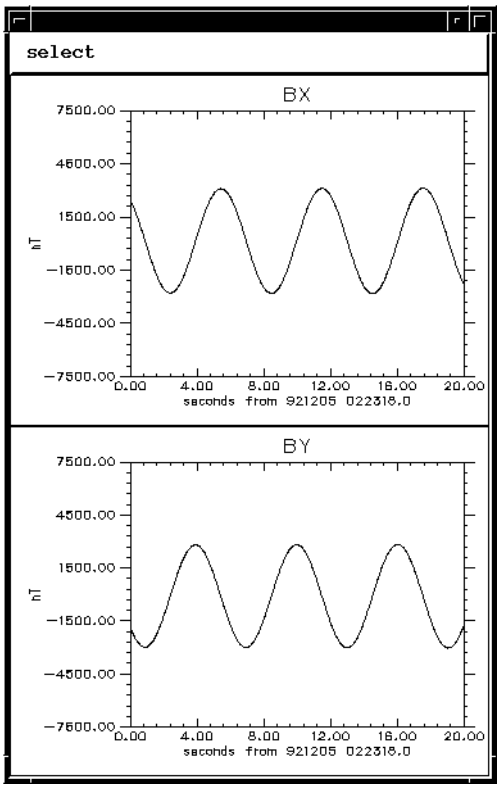

Figure 4: The output from the bf (XGKS) client

5 Graphics

A convienient way of visualising the ISDAT output is to use external software like IDL or MatLab. However, you can also write your own graphics client. You can use any locally available graphics tools, by loading the appropriate libraries. In general, your client will then not be portable. It is recommended to write ISDAT graphics clients in PHIGS/PEX for maximum portability and functionality. However, there are also ISDAT clients written in XGKS. There is an ISDAT library in support of XGKS (GiLib), see on line manuals. We will show examples of simple graphics clients that display the three magnetic field components of Viking. The first example is written in XGKS, and the second in PHIGS.

5.1 XGKS client

All necessary libraries required for writing XGKS graphics is provided in the ftp distribution of ISDAT.

The output from the *bf* XGKS client shown in Figure 4. The corresponding source code is listed in Appendix E.

5.2 PHIGS/PEX client

To compile and run a PHIGS/PEX client, PHIGS and PEX libraries are required. They are not provided with the ftp distribution of ISDAT.

The source code for a "bf" client written in PHIGS/PEX is listed in Appendix F.

6 Special features

The ISDAT clients may include some special features that are described in the following sections.

6.1 System menu

Every graphics window (or identifiable part) may be associated with an *ISDAT system menu*. At present, the system menu has two entries: redraw and filter. It is planned to incude more entries. It is activated by pressing the $< ctrl >$right mouse button inside the window.

6.1.1 Redraw entry

This results in an update of the associated panel without updating the whole family of clients.

6.1.2 Filter entry

This so called *pipe mechanism* is implemented in the bf code listed in Appendix E. We use that source code as an example.To enable the system menu you use the call

```
for(i=0; i < DRAWING_COUNT; i++) {
    IsCreateSystemMenu(UiDrawingWidget(drawings[i].part));
}
```

where a system menu is enabled for each drawing panel.

Many pipe entry/exit points may be implanted in the code. They are all associated with a unike name. Before being used they have to be registred by the call:

```
for(i = 0; i < DRAWING_COUNT; i++ ) {
    IsRegisterPipe(UiDrawingWidget(drawings[i].part, "data");
    IsRegisterPipe(UiDrawingWidget(drawings[i].part, "plot");
}
```

where we have registred two pipes *data* and *plot*.

In the UpdateBfield process we declare the pipe structure

```
IsPipeDesc piped;
```

and we fill the structure and define the *data* pipe entry/exit point by the sequence:

```
piped.samples[0] = desc.samples[0];
piped.dimension = desc.dimension;
piped.type = IsPIPE_FLOAT;
IsCallPipe(UiDrawingWidget(pDrawing->part),"data",&piped,&data);
desc.samples[0] = piped.samples[0];
desc.dimension = piped.dimension;
if (piped.type != IsPIPE_FLOAT)
IsError("Can't handle data type %d", piped.type);
```

6.2 Error handling

ISDAT provides an error and warnings handling mechanism via the IsError() and IsWarning() calls. Using them, the time manager takes care of displaying the client error messages. The calls are described in the on-line manuals.

6.3 Filters

Filters provide the user with a runtime modification function of the data stream. The programming of filters is described in [Ref. 3].

7 On line manuals

For an ISDAT client programmer, access to the current on-line manuals is indispensable. The on-line manuals ar organised according to the general Unix scheme. I.e. for programming purposes, we are interested in the *subroutine library* which is stored under manual chapter 3. The *user commands* under manual chapter 1, in the ISDAT implementation contains the client manuals, and are not listed below. There is one group of libraries that are related to the ISDAT system, and another group of libraries intended as a support for the scientific analysis of data.

7.1 ISDAT system associated libraries

7.1.1 The data base library

Purpose: To handle the client/server (client/DBH) communications.
The DbGetData is the most important manual page, since it contains the definition of the whole data request structure. It is included as Appendix D. The DbLib includes the following calls:

```
DbClose.3              DbControl.3          DbDownload.3
DbErrorString.3        DbFree.3             DbGetContent.3
DbGetData.3            DbGetInfo.3          DbName.3
DbName2Spec.3          DbOpen.3             DbPrepareData.3
DbQuantityString.3     DbQuery.3            DbSpec2Name.3
DbUnitString.3         DbUpload.3           DbWarningString.3
```

7.1.2 The user interface library

Purpose: To provide functions to build a client user interface based on Motif.
The user interface library includes the following calls:

```
UiAddCallback.3        UiAssignWsId.3       UiCallCallbacks.3
UiCreateLayout.3       UiCreateMenu.3       UiDestroyPart.3
UiDrawingWidget.3      UiEditorSetValue.3   UiFileSetDir.3
UiFreePartName.3       UiGetLabel.3         UiInitialize.3
UiListAddItem.3        UiListDeleteItem.3   UiMainLoop.3
UiMapLayouts.3         UiMapPart.3          UiNameToPart.3
UiOptionMenuSelect.3   UiPartToName.3       UiPartToWidget.3
UiPotChange.3          UiRelativeToPart.3   UiSetLabel.3
UiSetSensitivity.3     UiTextChange.3       UiTextClear.3
UiUnmapPart.3          UiWS_ID.3            UiWidget.3
```

7.1.3 The ISDAT library

Purpose: To provide the client/time manager interactions.
The IsLib includes the following calls:

```
IsAddCallback.3        IsCallCallbacks.3    IsCallPipe.3
IsChangeTime.3         IsClientExec.3       IsClientNotify.3
IsClientPath.3         IsCreateSystemMenu.3 IsExec.3
IsGetTmInfo.3          IsInitialize.3       IsMainLoop.3
IsManager.3            IsPipeRead.3         IsPipeWrite.3
IsRedrawMe.3           IsRegisterPipe.3     IsSetTmInfo.3
```

7.1.4 The ISDAT utilities library

Purpose: To provide time conversion and some miscellaneous multi dimensional memory allocation functions.
The IsutilLib includes the following calls:

```
IsAddTime.3             IsCmpTime.3          IsDivTime.3
IsDumpCore.3            IsMjd2Time.3         IsMulTime.3
IsSeconds2Time.3        IsSubTime.3          IsTime2Hms.3
IsTime2Mjd.3            IsTime2Seconds.3     IsTime2VikStw.3
IsTime2YmdHms.3         IsTimeGm.3           IsVikStw2Time.3
IsYmdHms2Time.3
```

7.1.5 The graphics interface library

Purpose: To provide some high level XGKS functions.
NOTE: This is a NON-SUPPORTED library, intended for internal IRF-U use.
The GiLib includes the following calls:

```
[to be written]
```

7.1.6 The GKS interface library

The GkLib includes the following calls:

```
GkOpenWs.3
```

7.1.7 The PHIGS interface library

The PhLib includes the following calls:

```
PhOpenWs.3
```

7.2 ISDAT analysis support libraries

7.2.1 Time series analysis library

The time series analysis library includes the following calls:

```
Ts1FFT.3                Ts2FFT.3             TsARMAAutocorr.3
TsARMACholesky.3        TsARMAPSD.3          TsAutoCorrel.3
TsCQ2CohPhase.3         TsCheckARMA.3        TsCheckFFT.3
TsCheckMEM.3            TsErrorToString.3    TsEvlARMA.3
TsEvlMEM.3              TsInitWindow.3       TsLsmywe.3
TsMEMCof.3              TsMEMPSD.3           TsMayneFirooza.3
TsNrCorrel.3            TsRealFFT.3          TsSlowCorrel.3
TsSpect.3               TsWindowData.3       TsXCorrel.3
TsXSpect.3              TsXSpectCQ.3
```

References

[1] G. Holmgren and A. Lundgren. Isdat 1.0 interactive scientific analysis tool. an introduction. Technical report, IRF-U, February 1994.

[2] A. Lundgren and G. Holmgren. Isdat 1.0 programmers guide. 1. overview and general guidelines. Technical report, IRF-U, February 1994.

[3] A. Lundgren and G. Holmgren. Isdat 1.0 programmers guide. 5. filters. Technical report, IRF-U, February 1994. not yet written.

A forbit source code

```
/*************************************************************
 * Compute and show Freja orbit number.                      *
 * Author: Gunnar Holmgren                                   *
 * Written: 17 Feb 1994                                      *
 *************************************************************/
#include <stdio.h>      /* Standard input output declarations */
#include <Is.h>         /* Isdat declarations                 */
#include <Ui.h>         /* User interface declarations        */
#include <Db.h>         /* Data base declarations             */

#define GH_LATER 0
#define GH_NOW 1

/* String that defines the window */
static char *basicLayoutString = "\
    layout 'Freja orbit' basicLayoutPart=. {      \n \
        frame 'Freja orbit' {                     \n \
            text '' -c24 -r2 orbitWindowTextPart= \n \
        }                                         \n \
    }                                                \
";

/* define the part identifiers we need before UiIdentifierMap*/
static UiPart basicLayoutPart;          /* top level part */
static UiPart orbitWindowTextPart;      /* text part      */

/* fill in identifier mapping table */
UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(basicLayoutPart)
    UiID_MAP_ENTRY(orbitWindowTextPart)
    UiID_MAP_END
};

main(argc, argv)
int argc;
char **argv;
{
    Display *display;
    void UpdateTimeCB();
    IsTmInfo *info;
    Database *db;


    /* initialize user interface library */
    display = UiInitialize(argc, argv);

    /* initialize isdat library          */
    IsInitialize(argc, argv, display);

    /* create the basic layout */
    if (!UiCreateLayout(NULL, basicLayoutString, idMap, 0))
        IsError("Cannot create the basic layout");

    /* Get in touch with data base handler */
```

```
    if ((db = DbOpen(NULL, argc, argv)) == NULL)
        IsError("Cannot connect to database %s", DbName(NULL));

    UiMapLayouts();            /* make layout visible */

    /* callback for orbit manager info */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    /* enter main loop and do never return */
    IsMainLoop();
}

/* UpdateTimeCB()                                    *
 * Callback that is called when the orbit manager    *
 * has sent us new time information.                 */

void UpdateTimeCB(reason, tmInfo, closure)
int reason;
IsTmInfo *tmInfo;
IsPointer closure;
{
    int FrejaOrbit();
    char string1[64], string2[64];
    int column=0, row=0;

    if(tmInfo->project != DbFREJA)
        IsError("This is not the Freja project");
    UiTextClear(orbitWindowTextPart,GH_LATER);
    sprintf(string1,"orbit: %d",FrejaOrbit(tmInfo));
    UiTextChange(orbitWindowTextPart,GH_LATER,column,row++,string1);
    IsTime2YmdHms(tmInfo->start,string2);
    sprintf(string1,"time:  ");
    strcat(string1,string2);
    UiTextChange(orbitWindowTextPart,GH_NOW,column,row,string1);
}

/* FrejaOrbit returns the Freja orbit number                  *
 * If time corresponds to a time pre-lanunch, a negative orbit *
 * number is returned                                         */

int FrejaOrbit(tmInfo)
IsTmInfo *tmInfo;
{
    IsTime launchTime, timeInOrbit;
    double orbitalPeriodMinutes = 108.95764;
    int firstOrbitNo = 1;

    IsYmdHms2Time("921006 090939.0",&launchTime);
    timeInOrbit = tmInfo->start;
    IsSubTime(&timeInOrbit,&launchTime);
    return((timeInOrbit.s + timeInOrbit.ns * 1.0e-9) /
        60.0 / orbitalPeriodMinutes + firstOrbitNo);
}
```

B desc source code

B.1 File main.c

```
/*************************************************************
 * Show the ISDAT data request structure                     *
 * Author: Gunnar Holmgren                                   *
 * Written: 28 Feb 1994                                      *
 *************************************************************/
#include <stdio.h>        /* Standard input output declarations */
#include <Is.h>           /* Isdat declarations */
#include <Ui.h>           /* User interface declarations */
#include <Db.h>           /* Data base declarations */
#include "query.h"        /* query declarations */

#define GH_LATER 0
#define GH_NOW 1
#define GH_MAX_CHAR 50

/* String that defines the window */
static char *basicLayoutString = "                          \
    layout 'desc' basicLayoutPart=. {                   \n \
        menubar 'Menu' menubarPart=.;                   \n \
            frame 'data structure'                      \n \
                text '' -c40 -r22 descWindowTextPart=.  \n \
            }                                           \n \
    }                                                       \
";

/* define the part identifiers we need before UiIdentifierMap*/
static UiPart basicLayoutPart;          /* top level part */
static UiPart menubarPart;              /* menubar part */
static UiPart descWindowTextPart;       /* test part      */

/* forward declaration of callback functions */
void DataChannelCB();

/* fill in callback mapping table */
UiCallbackMap cbMap[] = {
    UiCB_MAP_ENTRY(DataChannelCB)
    UiCB_MAP_END
};

/* fill in identifier mapping table */
UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(basicLayoutPart)
    UiID_MAP_ENTRY(menubarPart)
    UiID_MAP_ENTRY(descWindowTextPart)
    UiID_MAP_END
};

/* structure that accommodates strings related to default settings */
typedef struct _GhNames {
    char reduction[GH_MAX_CHAR];
    char gapFill[GH_MAX_CHAR];
```

```
} GhNames;

DbDataDesc dataDesc;
GhNames dataName;
Database *db;

/* The purpose of the main process is to initialize all settings *
 * and then enter the main loom to stay there pending callbacke  */
main(argc, argv)
int argc;
char **argv;
{
    Display *display;
    void UpdateTimeCB();
    IsTmInfo *info;
    UiPart part;
    char *inputMenuString;
    char *BuildQueryTree();

    /* initialize user interface library */
    display = UiInitialize(argc, argv);
    /* initialize isdat library          */
    IsInitialize(argc, argv, display);

    /* create the basic layout */
    if (!UiCreateLayout(NULL, basicLayoutString, idMap, 0))
        IsError("Cannot create the basic layout");

    /* Get in touch with data base handler */
    if ((db = DbOpen(NULL, argc, argv)) == NULL)
        IsError("Cannot connect to database %s", DbName(NULL));

    /* get manager supplied project and member */
    info = IsGetTmInfo();

    /* Create a string that defines the menu used to specify *
     * the desired data channel. The menu string is built    *
     * using information supplied by the data base handler   *
     * and communicated to us via the DbQuery call in the    *
     * CreateLevel process called from the BuildQueryTree    *
     * call below.                                           */
    inputMenuString = BuildQueryTree(db, DbQUERY_ALL, info->project,
        info->member, "input", "DataChannelCB");

    /* Use the inputMenuString to actually create the menu   */
    if (!UiCreateMenu(menubarPart, inputMenuString, idMap, cbMap))
        IsError("Cannot create project specific menu");

    UiMapLayouts();              /* make layout visible */

    /* callback for orbit manager info */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    IsMainLoop();                /* enter main loop and do never return */
}
```

```
/* UpdateTimeCB()                                    *
 * Callback that is called when the orbit manager    *
 * has sent us new time information.                 */

void UpdateTimeCB(reason, tmInfo, closure)
int reason;
IsTmInfo *tmInfo;
IsPointer closure;
{
    void FillDataSpecStruct(), ShowDataSpec();
    float *data;
    int errorCode;

    FillDataSpecStruct(tmInfo);         /* set default values */
    /* request data       */
    errorCode = DbGetData(db,&dataDesc,&data);
    /* display the returned spec */
    ShowDataSpec(&dataDesc, errorCode);
    /* free the memory space allocated by the DbGetData call*/
    DbFree(data);
}

void ShowDataSpec(desc,errorCode)
DbDataDesc *desc;
int errorCode;
{
    int column=0, row=0;
    char string1[GH_MAX_CHAR], string2[GH_MAX_CHAR];
    DbSpecName specName;
    void AddLine();

    UiTextClear(descWindowTextPart,GH_LATER);
    IsTime2YmdHms(desc->start,string2);
    AddLine("Time (UT):",string2,row++,GH_LATER);
    IsTime2Seconds(desc->interval,string2);
    AddLine("Interval (s):",string2,row++,GH_LATER);
    DbSpec2Name(db,&dataDesc.spec,&specName);
    AddLine("Project:",specName.project,row++,GH_LATER);
    AddLine("Member:",specName.member,row++,GH_LATER);
    AddLine("Instrument:",specName.instrument,row++,GH_LATER);
    AddLine("Sensor:",specName.sensor,row++,GH_LATER);
    AddLine("Signal:",specName.signal,row++,GH_LATER);
    AddLine("Channel:",specName.channel,row++,GH_LATER);
    AddLine("Parameter:",specName.parameter,row++,GH_LATER);
    AddLine("Reduction:",dataName.reduction,row++,GH_LATER);
    AddLine("Gap fill:",dataName.gapFill,row++,GH_LATER);
    if(errorCode == DbSUCCESS) {
        sprintf(string1,"%d",desc->dimension);
        AddLine("Dimension:",string1,row++,GH_LATER);
        AddLine("Units:",DbUnitString(desc->units[0]),row++,GH_LATER);
        sprintf(string1,"%d",desc->samples[0]);
        AddLine("Samples:",string1,row++,GH_LATER);
        sprintf(string1,"%d",dataDesc.info.quantity[0]);
        AddLine("Quantity:",string1,row++,GH_LATER);
        sprintf(string1,"%g",dataDesc.info.minValue[0]);
        AddLine("Min value:",string1,row++,GH_LATER);
```

```
        sprintf(string1,"%g",dataDesc.info.maxValue[0]);
        AddLine("Max value:",string1,row++,GH_LATER);
        sprintf(string1,"%d",dataDesc.info.scaleType[0]);
        AddLine("Scale type:",string1,row++,GH_LATER);
        sprintf(string1,"%g",dataDesc.info.samplingFreq[0]);
        AddLine("Sampling rate:",string1,row++,GH_NOW);
        if(strlen(dataDesc.info.message) > 1) {
            AddLine("Info message:",dataDesc.info.message,row++,GH_NOW);
        }
        strcpy(string1,DbWarningString(dataDesc.warning));
        if(strlen(string1) > 1) {
            AddLine("Warning:",string1,row++,GH_NOW);
        }
    }
    else {
        sprintf(string1,"DbGetData returned the message:");
        AddLine("DbGetData returned the message:","",row++,GH_NOW);
        AddLine(DbErrorString(errorCode),"",row++,GH_NOW);
    }
}

void AddLine(s1,s2,row,when)
char *s1, *s2;
int row, when;
{
    int col;

    col = 0;  UiTextChange(descWindowTextPart,when,col,row,s1);
    col = 16; UiTextChange(descWindowTextPart,when,col,row,s2);
}

void FillDataSpecStruct(tmInfo)
IsTmInfo *tmInfo;
{
    dataDesc.start = tmInfo->start;
    dataDesc.interval = tmInfo->interval;
    dataDesc.spec.project = tmInfo->project;
    dataDesc.spec.member = tmInfo->member;
    dataDesc.units[0] = DbUN_PHYS;
    dataDesc.samples[0] = 100000;
    dataDesc.reduction[0] = DbRED_AVERAGE;
    sprintf(dataName.reduction,"Averaging");
    dataDesc.gapFill[0] = DbGAP_ZERO;
    sprintf(dataName.gapFill,"fill with zero");
}

/* Selection of data input */
static void DataChannelCB(part, reason, state, id)
UiPart part; /* drawing part */
int reason; /* callback reason */
int *state; /* set to one if enable */
int id; /* which node */
{
    Tree *last;
    Tree *p;
```

```
    last = (Tree *)id;

    for (p = last; p->prev; p = p->prev)
    {
        switch (p->level) {
        case DbLEVEL_INSTRUMENT:
            dataDesc.spec.instrument = p->entry->value; break;
        case DbLEVEL_SENSOR:
            dataDesc.spec.sensor = p->entry->value; break;
        case DbLEVEL_SIGNAL:
            dataDesc.spec.signal = p->entry->value; break;
        case DbLEVEL_CHANNEL:
            dataDesc.spec.channel = p->entry->value; break;
        case DbLEVEL_PARAMETER:
            dataDesc.spec.parameter = p->entry->value; break;
        }
    }
}
```

B.2 File query.c

```
#include <stdio.h>
#include <math.h>
#include <Is.h>                 /* Isdat declarations */
#include <Ui.h>                 /* User interface declarations */
#include <Db.h>                 /* Data base declarations */
#include "query.h"

#define PROLOG \
    if (p != query) { \
        tp->link = NewTreeElement(); \
        tp = tp->link; \
    } \
    tp->prev = parent; \
    tp->entry = p; \
    tp->level = d.level;

static Tree treeHead;

static void Concat();

char *BuildQueryTree(db, mode, project, member, menuName, CBName)
Database *db;
int mode;
int project;
int member;
char *menuName;
char *CBName;
{
    char *str = NULL;
    DbQueryDesc d;
    char tmp[256];
    static void CreateLevel();
    static void PrintTree();
    static void CreateMenuString();
```

```
    treeHead.next = NULL;
    treeHead.link = NULL;
    treeHead.prev = NULL;
    treeHead.tmp = NULL;
    treeHead.entry = NULL;
    d.mode = mode;
    d.spec.project = project;
    d.spec.member = member;
    CreateLevel(&treeHead, 0, db, &d);
    sprintf(tmp, "menu '%s' {\n", menuName);
    Concat(&str, tmp);
    CreateMenuString(treeHead.next, &str, CBName);
    sprintf(tmp, "}");
    Concat(&str, tmp);
    return str;
}

static void CreateLevel(parent, level, db, desc)
Tree *parent;
int level;
Database *db;
DbQueryDesc *desc;
{
    DbQueryData *p;
    DbQueryData *query;
    DbQueryDesc d;
    Tree *tp;
    static Tree *NewTreeElement();

    d = *desc;
    switch (level) {
    case 0:
        d.level = DbLEVEL_INSTRUMENT;
        DbQuery(db, &d, &query);
        if (!query) break;
        tp = parent->next = NewTreeElement();
        for (p = query; p && p->name; p++) {
            PROLOG
            d.spec.instrument = p->value;
            CreateLevel(tp, 1, db, &d);
        }
        break;
    case 1:
        d.level = DbLEVEL_SENSOR;
        DbQuery(db, &d, &query);
        if (!query) break;
        tp = parent->next = NewTreeElement();
        for (p = query; p && p->name; p++) {
            PROLOG
            d.spec.sensor = p->value;
            CreateLevel(tp, 2, db, &d);
        }
        break;
    case 2:
        d.level = DbLEVEL_SIGNAL;
```

```
        DbQuery(db, &d, &query);
        if (!query) break;
        tp = parent->next = NewTreeElement();
        for (p = query; p && p->name; p++) {
            PROLOG
            d.spec.signal = p->value;
            CreateLevel(tp, 3, db, &d);
        }
        break;
    case 3:
        d.level = DbLEVEL_CHANNEL;
        DbQuery(db, &d, &query);
        if (!query) break;
        tp = parent->next = NewTreeElement();
        for (p = query; p && p->name; p++) {
            PROLOG
            d.spec.signal = p->value;
            CreateLevel(tp, 4, db, &d);
        }
        break;
    case 4:
        d.level = DbLEVEL_PARAMETER;
        DbQuery(db, &d, &query)
        if (!query) break;
        tp = parent->next = NewTreeElement();
        for (p = query; p && p->name; p++) {
            PROLOG
        }
        break;
    }
}

static Tree *NewTreeElement()
{
    Tree *p;

    p = IsNew(Tree);
    p->next = NULL;
    p->link = NULL;
    p->prev = NULL;
    p->tmp = NULL;
    p->entry = NULL;
    return p;
}

static void PrintTree(node)
Tree *node;
{
    Tree *p;
    static void PrintPath();

    for (p = node; p; p = p->link) {
        if (p->next) {
            PrintTree(p->next);
        } else {
            PrintPath(p);
```

```
        }
    }
}

static void PrintPath(last)
Tree *last;
{
    Tree *p;

    for (p = last; p->prev; p = p->prev) p->prev->tmp = p;
    for (p = p->tmp; p; p = p->tmp) printf("%s ", p->entry->name);
    printf("\n");
}

static void CreateMenuString(node, str, CBName)
Tree *node;
char **str;
char *CBName;
{
    Tree *p;
    static void MenuPath();

    for (p = node; p; p = p->link) {
        if (p->next) {
            CreateMenuString(p->next, str, CBName);
        } else {
            MenuPath(p, str, CBName);
        }
    }
}

static void MenuPath(last, str, CBName)
Tree *last;
char **str;
char *CBName;
{
    Tree *p;
    char tmp[256];

    for (p = last; p->prev; p = p->prev) p->prev->tmp = p;
    for (p = p->tmp; p; p = p->tmp) {
        sprintf(tmp, "'%s' ", p->entry->name);
        Concat(str, tmp);
    }
    sprintf(tmp, "%s(%d);\n", CBName, last);
    Concat(str, tmp);
}

#define CHUNK 10240

static void Concat(str, new)
char **str;
char *new;
{
    int len;
    static int index;
```

```
    static int total;

    len = strlen(new);
    if (*str) {
        if (total - index <= len) {
            total += CHUNK;
            *str = UiRealloc(*str, total + 1);
        }
    } else {
        *str = UiMalloc(CHUNK + 1);
        index = 0;
        total = CHUNK;
    }
    strcpy(*str + index, new);
    index += len;
}
```

B.3 File query.h

```
\begin{verbatim}
typedef struct _Tree {
    struct _Tree *next;
    struct _Tree *link;
    struct _Tree *prev;
    struct _Tree *tmp;
    DbQueryData *entry;
    int level;
} Tree;
```

C panels.c source code

```
#include <math.h>
#include <Is.h>
#include <Ui.h>

/*
* define layouts
*/
static char *drawingString = " \
    layout 'drawing area' { \n \
drawing '0' drawing '1' drawing '2'; \n \
drawing '3'; \n \
drawing '4'; \n \
drawing '5'; \n \
    } \
";

static char *rdrawingString = " \
    layout 'relative drawing area' { \n \
drawing '' -c0,0,.5,.1) drawing '' -c.5,0,.75,.1 \n \
    drawing '' -c.75,0,1.0,.1; \n \
drawing '' -c0,.1,1.0,.2; \n \
drawing '' -c0,.2,1.0,.4; \n \
drawing '' -c0,.4,0.7,1.0 \n \
frame '' -c-1.,-1.,1.0,1.0 { button 'hej' button 'san' }; \n \
    } \
";

static char *layoutArr = " \
    layout 'control panel' -f { \n \
menubar 'Menu' -l; \n \
frame 'Mode' -l { \n \
    button 'button1' -s Button1CB('button 1 pressed') \n \
    button 'button2' Button2CB(2) \n \
    button 'button3' -s button 'button4' \n \
} \n \
drawing 'graph'; \n \
frame 'Data' -r -l { \n \
    button 'button1' -s; button 'button2'; \n \
    button 'button3'; button 'button4' \n \
} \n \
frame 'FFT control' -l { \n \
    button 'button1' -t; button 'button2' -t -s; \n \
    button 'button3' -t; \n \
} \n \
frame 'Edit field #1' -l { \n \
    editor 'text: ' 'test'; \n \
    editor 'text1: ' 'test1'; \n \
}; \n \
frame 'Edit field #2' -l { \n \
    editor 'text: ' 'test' \n \
    editor 'text1: ' 'test1' \n \
} \n \
frame 'option menu' -l { \n \
    optmenu 'test' \n \
```

```
}; \n \
frame 'Auto gauges #1' -l { \n \
    pot 'speed (km/h)' 10 200 50 speedPart=. -d PotCB('moved'); \n \
    pot 'charge (V)' 0 100 30; \n \
    pot 'oil (psi)' 0 100 20; \n \
    pot 'temp (C)' 0 100.33 17.01; \n \
} \n \
frame 'Short panel' -l { \n \
    pot 'S' 10 200 50 -v \n \
    pot 'V' 0 100 30 -v \n \
    pot 'P' 0 100 20 -v \n \
    pot 'C' 0 100 17 -v \n \
} \n \
frame '' { \n \
    pot 'speed' 80 100 90 -v \n \
}; \n \
frame 'Auto gauges #2' -l { \n \
    pot 'speed (km/h)' 10 200 50 \n \
    pot 'charge (V)' 0 100 30 \n \
    pot 'oil (psi)' 0 100 20 \n \
}; \n \
frame 'Direction' -l { \n \
    arrow 'left' -dleft button 'stop' arrow 'right' -dright \n \
} \n \
frame 'Direction' -l { \n \
    arrow 'left' -dleft arrow 'right' -dright \n \
}; \n \
frame 'Status' -l { \n \
    text '' -c40 -r5 statusPart=. \n \
} \n \
frame 'List' -l { \n \
    list '' -c10 -r5 -ms ListSelectionCB() listPart=. \n \
}; \n \
    } \
";

UiPart statusPart;
UiPart listPart;
UiPart speedPart;

UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(statusPart)
    UiID_MAP_ENTRY(listPart)
    UiID_MAP_ENTRY(speedPart)
    UiID_MAP_END
};

static void Button1CB();
static void Button2CB();
static void PotCB();
static void ListSelectionCB();

UiCallbackMap cbMap[] = {
    UiCB_MAP_ENTRY(Button1CB)
    UiCB_MAP_ENTRY(Button2CB)
    UiCB_MAP_ENTRY(PotCB)
```

```
    UiCB_MAP_ENTRY(ListSelectionCB)
    UiCB_MAP_END
};

static char *menuArr = {"\
menu 'popup menu title' { \n \
  'print'; \n \
  'print' 'lp' 'do'; \n \
  'print' 'lp' 'cancel'; \n \
  'print' 'pr' 'do'; \n \
  'print' 'pr' 'cancel'; \n \
  'plot' 'laser'; \n \
  'plot' 'term'; \n \
  'strange !@#$%^&*() label'; \n \
} \n \
"};

static char *optmenuArr = {"\
menu 'optmenu:' { \n \
  'print'; \n \
  'pr'; \n \
  'cancel'; \n \
  'laser'; \n \
  'term'; \n \
} \n \
"};

static void Button1CB(part, reason, state, str)
UiPart part;
int reason;
int *state;
char *str;
{
printf("Button1CB(%s)\n", str);
    UiTextClear(statusPart, 0);
    UiTextChange(statusPart, 0, 0, 0, "first row");
    UiTextChange(statusPart, 0, 20, 1, "second row with wrap at end of line");
    UiTextChange(statusPart, 0, 30, 2, "third row");
    UiTextChange(statusPart, 0, 15, 3, "fourth row");
    UiTextChange(statusPart, 1, 0, 4, "fifth row");
    UiPotChange(speedPart, 78.0);
}

static void Button2CB(part, reason, state, id)
UiPart part;
int reason;
int *state;
int id;
{
int i;
char text[256];

printf("Button2CB(%d)\n", id);
    for (i = 0; i < 100; i++) {
UiTextChange(statusPart, 0, 0, 0, "first row          ");
UiTextChange(statusPart, 0, 20, 1, "second row with wrap at end of line");
```

```
sprintf(text, "fourth row %.3d", i);
UiTextChange(statusPart, 0, 15, 3, text);
UiTextChange(statusPart, 1, 0, 4, "fifth row 0000");

UiTextChange(statusPart, 0, 0, 0, "          first row");
UiTextChange(statusPart, 0, 20, 1, "second row with      at end of line");
UiTextChange(statusPart, 1, 0, 4, "fifth row 1111");
    }
}

static void button3(part, reason, data, closure)
UiPart part;
int reason;
UiPointer data;
UiPointer closure;
{
    int *state;
    char *str;

    state = (int *)data;
    str = (char *)closure;

    UiSetLabel(part, "new label");
    UiListAddItem(listPart, "item 0");
    UiListAddItem(listPart, "item 1");
    UiListAddItem(listPart, "item 2");
    UiListAddItem(listPart, "item 3");
    UiListAddItem(listPart, "item 4");
    UiListAddItem(listPart, "item 5");
    UiListAddItem(listPart, "item 6");
    UiListAddItem(listPart, "item 7");
    UiListAddItem(listPart, "item 8");
    UiListAddItem(listPart, "item 9");
}

static void button4(part, reason, data, closure)
UiPart part;
int reason;
UiPointer data;
UiPointer closure;
{
    int *state;
    char *str;

    state = (int *)data;
    str = (char *)closure;
    UiListDeleteItem(listPart, "item 0");
    UiListDeleteItem(listPart, "item 1");
    UiListDeleteItem(listPart, "item 2");
    UiListDeleteItem(listPart, "item 3");
    UiListDeleteItem(listPart, "item 4");
    UiListDeleteItem(listPart, "item 5");
    UiListDeleteItem(listPart, "item 6");
    UiListDeleteItem(listPart, "item 7");
    UiListDeleteItem(listPart, "item 8");
    UiListDeleteItem(listPart, "item 9");
```

```
    UiDestroyPart(part);
}

static void toggle1(part, reason, data, closure)
UiPart part;
int reason;
UiPointer data;
UiPointer closure;
{
    int *state;
    char *str;
    UiPart butt2part;

    state = (int *)data;
    str = (char *)closure;

    butt2part = UiNameToPart("control panel", "Mode", NULL);
    printf("toggle(%d)\n", *state);
    if (*state) {
UiUnmapPart(butt2part);
    } else {
UiMapPart(butt2part);
    }
}

static void PotCB(part, reason, position, str)
UiPart part;
int reason;
double *position;
char *str;
{
    printf("pot(%s) position = %f\n", str, *position);
}

static void ListSelectionCB(part, reason, info, closure)
UiPart part;
int reason;
UiListCBS *info;
UiPointer closure;
{
    printf("list item = (%s) %d\n", info->item, info->position);
}

static void ListPosCB(part, reason, position, closure)
UiPart part;
int reason;
int position;
UiPointer closure;
{
    printf("list position = %d\n", position);
}

main(argc, argv)
int argc;
char **argv;
{
```

```
    Display *dpy;
    UiPart part;

    dpy = UiInitialize(argc, argv);
    IsInitialize(argc, argv, dpy);

    UiCreateLayout(NULL, drawingString, idMap, cbMap);
    UiCreateLayout(NULL, rdrawingString, idMap, cbMap);
    UiCreateLayout(NULL, layoutArr, idMap, cbMap);

#ifdef KOKO
    part = UiNameToPart("control panel", "option menu", NULL);
#endif
    part = UiNameToPart("control panel", "option menu", "test", NULL);
    UiCreateMenu(part, optmenuArr, NULL, NULL);
/*
    part = UiNameToPart("control panel", "Mode", "button1", NULL);
    UiAddCallback(part, UiCR_BUTTON, Button1CB, "button 1 pressed");

    part = UiNameToPart("control panel", "Mode", "button2", NULL);
    UiAddCallback(part, UiCR_BUTTON, Button2CB, "button 2 pressed");
*/

    part = UiNameToPart("control panel", "Mode", "button3", NULL);
    UiAddCallback(part, UiCR_BUTTON, button3, "button 3 pressed");

    part = UiNameToPart("control panel", "Mode", "button4", NULL);
    UiAddCallback(part, UiCR_BUTTON, button4, "button 4 pressed");

    part = UiNameToPart("control panel", "FFT control", "button1", NULL);
    UiAddCallback(part, UiCR_BUTTON, toggle1, "button 1 pressed");

/*
    part = UiNameToPart("control panel", "Auto gauges #1", "speed (km/h)", NULL);
    UiAddCallback(part, UiCR_POT_DRAG, PotCB, "pot moved");

    statusPart = UiNameToPart("control panel", "Status", "", NULL);

    listPart = UiNameToPart("control panel", "List", "", NULL);
    UiAddCallback(listPart, UiCR_LIST_SELECTION, ListSelectionCB, NULL);
*/

    UiMapLayouts();
    IsMainLoop();
}
```

D DbGetData on line manual

```
DbGetData(3Db)            Db LIBRARY            DbGetData(3Db)


NAME
     DbGetData - get specified data from the database

SYNOPSIS
     #include "Isutil.h"
     #include "Db.h"

     int DbGetData({db, desc, data})
     Database *{db};
     DbDataDesc *{desc};
     float **{data};

     int DbGetSegmentedData({db, desc, seg, data})
     Database *db;
     DbDataDesc *desc;
     DbDataSegment **seg;
     float **data;

     int DbGetTimeTaggedData(db, desc, seg, data, time)
     Database *db;
     DbDataDesc *desc;
     DbDataSegment **seg;
     float **data;
     IsTime **time;

ARGUMENTS
     db        Pointer to an open database.

     desc      Pointer to a data description structure.

     seg       Specifies a pointer that will point to  a  segment
               table  on  return.   Storage for the data is allo-
               cated by this  function  and  it  is  the  callers
               responsibility  to  free  the  segment table using
               DbFree(3Db) when it is no longer needed.

     data      Specifies  a  pointer  that  will  point  to   the
               requested data on return.  Storage for the data is
               allocated by this function and it is  the  callers
               responsibility  to free the data using DbFree(3Db)
               when the data is no longer needed.

     time      Specifies  a  pointer  that  will  point  to   the
               requested  time  array on return.  Storage for the
               data is allocated by this function and it  is  the
               callers  responsibility  to  free  the  data using
               DbFree(3Db) when the data is no longer needed.
```

DESCRIPTION
Gets data from the database.
DbGetData(3Db) returns a contiguous data array, if a data gap or drop is present it will be filled according to the

specified gap fill strategy.
DbGetSegmentedData(3Db) returns a segment for each contiguous section of the data array, eg. if a drop is present in the data two segments will be returned.
DbGetTimeTaggedData(3Db) differs from DbGetSegmentedData(3Db) in that a time array is returned that gives the exact time of each returned data sample.

STRUCTURES

```
typedef struct _DbDataDesc {
    IsTime start;       /* start time of requested data (input/output) */
    IsTime interval;    /* time interval of requested data (input/output) */
    DbDataSpec spec;    /* data hierarchy specification (input/output) */
    int segments;       /* number of segments (output) */
    int dimension;      /* signal vector dimension (output) */
    int samples[5];     /* number of data samples (input/output) */
    int units[5];       /* physical units of returned data (input/output) */
    int skip[5];        /* number of real data samples to skip (input) */
    int reduction[5];   /* type of data reduction (input) */
    int gapFill[5];     /* how to fill data gaps (input) */
    DbDataInfo info;    /* information about the returned data (output) */
    DbExtraInfo extraInfo; /* extra information about the returned
                  data, reserved for use by the Berkeley
                  SSL group (output) */
    unsigned int warning; /* gives caller a warning that the requested data is
                             returned but is corrupted in some way (output) */
} DbDataDesc;
```

start — Start time of the requested data. The value may be changed by the call.

interval — Interval time of the requested data. The value may be changed by the call.

spec — Data hierarchy specification.

segments — Number of segments returned.

dimension — On return this member defines the vector dimension of the returned data.

units — Defines the units of the returned data. Possible input values are: DbUN_TM, DbUN_CORR and DbUN_PHYS. Possible return values are: DbUN_V_PER_M, DbUN_MV_PER_M, DbUN_PROCENT, DbUN_MV_PER_M_SQR_PER_HZ, DbUN_MICRO_AMP, DbUN_NANO_TESLA and UN_DbDCB.

samples — If reduction is not set to DbNONE and samples is set to a value n, at most n samples will be returned, the number of samples will reduced according to the reduction parameter.

On return this member contains the number of samples returned.

skip — When requesting multi-dimensional data, skip is used to skip a number of samples at the beginning of the vector. The skip is applied before the requested reduction takes place.

reduction — Defines the data reduction strategy used. If set to DbRED_NONE all available samples are returned. Possible reduction algorithms are DbRED_AVERAGE, DbRED_SKIP, DbRED_MIN, DbRED_MAX and DbRED_RESAMPLE.

gapFill — Defines how data gaps will be represented in the returned data. Gaps can be filled with IEEE NaN values (DbGAP_NAN), filled with zero values (DbGAP_ZERO) and filled with interpolated values (DbGAP_INTERPOL). The value is not changed by the call. Only used in DbGetData(3Db).

info — A structure returning information about the returned data.

extraInfo — Extra information about the returned data, reserved for use by the Berkeley SSL group.

warning — On return this member is set to indicate in which way the returned data is corrupted. Each reason is coded as a bit mask and one call can result in several warning conditions to be set. The defined warnings are: the experiment mode matches the requested criteria only part of the requested interval (DbWARN_PART), a data drop occurred in the interval (DbWARN_DROP), a gap is present in the interval (DbWARN_GAP), some part of the interval is before the beginning of the file (DbWARN_BOF) and an end of file occurred somewhere in the requested interval (DbWARN_EOF). The gaps will be filled according to the gap fill strategy defined.

A drop is flagged when data is missing because of some error. A gap is flagged when the data set is designed with gaps in between data.

```
typedef struct _DbDataSpec {
    int project;        /* project specification (input) */
    int member;         /* project member (input) */
```



```
    int instrument;     /* project instrument (input) */
    int sensor;         /* instrument sensor (input) */
    int signal;         /* instrument signal (input) */
    int channel;        /* instrument channel (input/output) */
    int parameter;      /* instrument parameter (input/output) */
} DbDataSpec;
```

project — Project specification. Can be one of DbVIKING, DbFREJA, DbCLUSTER, DbPROTO and DbEISCAT. The value is not changed by the call.

member — Project member. This field is only used for the Cluster and Eiscat projects. Valid values are CLU_0, CLU_1, CLU_2 and CLU_3 for Cluster and DbEIS_TROMSO, DbEIS_KIRUNA and DbEIS_SODANKYLA for Eiscat. The value is not changed by the call.

```
     instrument      Project instrument. Viking instruments are
                     DbVIK_V2, DbVIK_V3, DbVIK_V4L and DbVIK_V4H.
                     Cluster instruments are DbCLU_EFW and
                     DbCLU_STAFF. Eiscat instruments are DbEIS_VHF
                     and DbEIS_UHF. The value is not changed by
                     the call.

     sensor          Instrument sensor. Viking V2 sensors are
                     DbVIK2_BX, DbVIK2_BY and DbVIK2_BZ. Viking
                     V3 sensors are DbVIK3_PISP1 and DbVIK3_PISP2.
                     Viking V4L sensors are DbVIK4_EX, DbVIK4_EY,
                     DbVIK4_EZ, DbVIK4_DBX, DbVIK4_N1 and
                     DbVIK4_N2. Proto sensors are DbPROTO_CH0,
                     DbPROTO_CH1, DbPROTO_CH2, DbPROTO_CH3,
                     DbPROTO_CH4 and DbPROTO_CH5. Eiscat sensors
                     are tbd. The value is not changed by the
                     call.

     signal          Instrument signal. Viking V4L signals are
                     DbVIK4_WF, DbVIK4_DFT and DbVIK4_FB. Viking
                     V4H signals are DbVIK4_FB. Eiscat signals
                     are tbd. The value is not changed by the
                     call.

     channel         Instrument channel. Viking V4L filter bank
                     channels are DbVIK4_500HZ, DbVIK4_1KHZ and
                     DbVIK4_2KHZ. Viking V4H filter bank channels
                     are DbVIK4_4KHZ, DbVIK4_8KHZ, DbVIK4_16KHZ,
                     DbVIK4_32KHZ, DbVIK4_64KHZ, DbVIK4_128KHZ,
                     DbVIK4_256KHZ or DbVIK4_512KHZ.

     parameter       Instrument parameter.



ISDAT                Last change: alpha                      4



DbGetData(3Db)            Db LIBRARY            DbGetData(3Db)



     typedef struct _DbDataSegment {
         IsTime start;
         IsTime interval;
         int offset;
         int samples;
     } DbDataSegment;
```

```
start          Time of the first sample in this segment.

interval       Time interval of this segment.

offset         Data array index of the first sample in  this
               segment.

samples        Number of samples in this segment.

typedef struct _DbDataInfo {
    int quantity[5];    /* quantity descriptor (output) */
    float minValue[5];  /* value corresponding to minimum index (output) */
    float maxValue[5];  /* value corresponding to maximum index (output) */
    int scaleType[5];   /* type of scale (output) */
    double samplingFreq[5];/* sampling frequency (output) */
    char message[64];   /* informative message (output) */
} DbDataInfo;


quantity       Description of quantities associated with
               each   dimension.   Possible   values   are
               DbQTY_FREQUENCY, DbQTY_POWER, DbQTY_COUNTS,
               DbQTY_ENERGY and DbQTY_ANGLE.

minValue       Physical value corresponding  to  the  lowest
               array index (zero).

maxValue       Physical value corresponding to  the  highest
               array index (samples - 1).

scaleType      Type of scale used  in  the  array  indexing.
               Possible    values    are   DbSCALE_LIN   and
               DbSCALE_LOG.

sampleFreq     Returns the sensor sample frequency  used  by
               the experiment for the returned data.

message        May contain some extra  informative  message,
               if not it is set to an empty string.

typedef struct _IsTime { /* define Isdat time (IsTime) */
    long s;             /* seconds since January 1, 1970 */
    long ns;            /* and nanoseconds */
} IsTime;
```

RETURN VALUE

Returns DbSUCCESS if no error occurred. If an error occurred no data is returned and an error code is returned.

ERRORS

If an error occurs one of the following error codes is returned:

DbBAD_TIME	Requested time is not found on the disc.
DbBAD_PROJECT	The requested project is not available during the requested interval.
DbBAD_MEMBER	The requested member is not available during the requested interval.
DbBAD_INSTRUMENT	The requested instrument is not available during the requested interval.
DbBAD_SENSOR	The requested sensor is not available during the requested interval.
DbBAD_SIGNAL	The requested signal is not available during the requested interval.
DbBAD_CHANNEL	The requested channel is not available during the requested interval.
DbBAD_PARAMETER	The requested parameter is not available during the requested interval.
DbBAD_UNITS	The requested units is not valid.
DbBAD_REDUCTION	The requested reduction is not valid.
DbBAD_GAPFILL	The requested gapfill is not valid.
DbBAD_ALLOC	Request couldn't be serviced because of memory limitations.
DbBAD_INTERNAL	Request couldn't be serviced because of some internal failure.
DbNOT_IMPLEMENTED	The requested operation is not yet implemented for the given project.

E bf (GKS) source code

E.1 File main.c

```
/************************************************************
 *
 *    main.c
 *
 *    Shows the three magnetic field components bx, by, bz.
 *    Each component can be selected individually by a menu.
 *    Each drawing has a system menu attached that is popuped
 *    by the <Ctrl>right mouse button.
 *
 *    Written Aug 3, 1990 by Anders Lundgren, IRFU
 *
 ************************************************************/
#include <stdio.h>
#include <math.h>
#include <Is.h> /* Isdat declarations */
#include <Ui.h> /* User interface declarations */
#include <Db.h> /* Data base declarations */
#include <instrument/Viking2.h> /* To get Viking V2 declarations */
#include <X11/xgks.h>
#include "bfield.h"

/* define main layout: a menu bar and three drawing areas */
static char *layoutString = " \
    layout 'bfield' layoutPart=. { \n \
menubar 'Menu' menubarPart=.; \n \
drawing 'drawing0' drawing0=.; \n \
drawing 'drawing1' drawing1=.; \n \
drawing 'drawing2' drawing2=.; \n \
    } \
";

/* define select component menu: three toggle buttons */
static char *selectMenuString = " \
    menu 'select' { \n \
      'bx' -t -s SelectMenuCB(0); \n \
      'by' -t -s SelectMenuCB(1); \n \
      'bz' -t -s SelectMenuCB(2); \n \
    } \
";

/* forward declaration of callback functions */
static void SelectMenuCB();

/* fill in callback mapping table */
UiCallbackMap cbMap[] = {
    UiCB_MAP_ENTRY(SelectMenuCB)
    UiCB_MAP_END
};

/* define the part identifiers we need */
static UiPart layoutPart; /* top level part */
```

```
static UiPart menubarPart; /* menubar */
/* define an array of drawing parameters to keep track of bx, by and bz
   bx = 0, by = 1, bz = 2 */
static BfDrawing drawings[DRAWING_COUNT] = {
    {1, NULL, DbVIK2_BX, "BX"},
    {1, NULL, DbVIK2_BY, "BY"},
    {1, NULL, DbVIK2_BZ, "BZ"}
};

/* fill in identifier mapping table */
UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(layoutPart)
    UiID_MAP_ENTRY(menubarPart)
    "drawing0", &drawings[0].part,
    "drawing1", &drawings[1].part,
    "drawing2", &drawings[2].part,
    UiID_MAP_END
};

Database *db; /* opened database */

/*
 * SelectMenuCB()
 *
 * Callback that is called when the users wants to
 * disable/enable a drawing.
 */
static void SelectMenuCB(part, reason, state, drawing)
UiPart part; /* drawing part */
int reason; /* callback reason */
int *state; /* set to one if enable */
int drawing; /* which drawing */
{
    IsTmInfo *info;

    if (*state) { /* turn on drawing */
info = IsGetTmInfo();
drawings[drawing].on = 1;
UiMapPart(drawings[drawing].part);
UpdateBfield(&drawings[drawing], info);
    } else { /* turn off drawing */
drawings[drawing].on = 0;
UiUnmapPart(drawings[drawing].part);
    }
}

/*
 * UpdateTimeCB()
 *
 * Callback that is called when the orbit manager
 * has sent us new time information.
 */
static void UpdateTimeCB(reason, data, closure)
int reason; /* callback reason */
IsPointer data; /* orbit manager information */
IsPointer closure;
```

```
{
    int i;

    for (i = 0; i < DRAWING_COUNT; i++)
UpdateBfield(&drawings[i], (IsTmInfo *)data);
}

/*
 * main()
 *
 * The program starts here.
 */
main(argc, argv)
int argc;
char **argv;
{
    int i; /* loop counter */
    Display *dpy; /* opened X display */

    dpy = UiInitialize(argc, argv); /* initialize user interface library */
    IsInitialize(argc, argv, dpy); /* initialize isdat library */
    GkInitialize(argc, argv); /* initialize Gks library */
    GiInitialize(argc, argv); /* initialize Graphical interface library */

    /* create the basic layout */
    if (!UiCreateLayout(NULL, layoutString, idMap, cbMap)) exit(1);

    /* attach select menu to the menubar */
    if (!UiCreateMenu(menubarPart, selectMenuString, idMap, cbMap)) exit(1);

    /* create system menus on each drawing */
    for (i = 0; i < DRAWING_COUNT; i++) {
IsCreateSystemMenu(UiDrawingWidget(drawings[i].part));
    }

    /* make layout visible */
    UiMapLayouts();

    /* add callback for orbit manager information */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    /* open a Gks workstation on each drawing */
    for (i = 0; i < DRAWING_COUNT; i++) {
GkOpenWs(GkWsId(drawings[i].part),
 UiDrawingWidget(drawings[i].part), NULL);
    }

    /* register a pipe for each component */
    for (i = 0; i < DRAWING_COUNT; i++) {
IsRegisterPipe(UiDrawingWidget(drawings[i].part), "data");
IsRegisterPipe(UiDrawingWidget(drawings[i].part), "plot");
    }

    /* Get in touch with data base handler */
    if ((db = DbOpen(NULL, argc, argv)) == NULL) {
IsError("cannot connect to database %s", DbName(NULL));
```

```
    }

    /* enter main loop and never return */
    IsMainLoop();
}
```

E.2 File update.c

```
#include <stdio.h>
#include <math.h>
#include <Is.h> /* Isdat declarations */
#include <Ui.h> /* User interface declarations */
#include <Db.h> /* Data base  declarations */
#include <instrument/Viking2.h> /* To get Viking V2  declarations */
#include <Gi.h> /* Graphics interface  declarations */
#include <X11/xgks.h>
#include "bfield.h"

extern Database *db; /* opened database */

/*
 * UpdateBfield()
 *
 * Gets the data from the specified time and interval,
 * generates a plot using the autoscale function.
 */
void UpdateBfield(pDrawing, tmInfo)
BfDrawing *pDrawing; /* current drawing */
IsTmInfo *tmInfo; /* orbit manager information */
{
    int result;
    DbDataDesc desc; /* data description */
    float *data;
    IsPipeDesc piped;
    GiGraphDesc gd;

    /* describe the data we want */
    desc.start = tmInfo->start;
    desc.interval = tmInfo->interval;
    desc.spec.project = tmInfo->project;
    desc.spec.member = tmInfo->member;
    desc.spec.instrument = DbVIK_V2; /* V2 instrument */
    desc.spec.sensor = pDrawing->sensor; /* DbBX, DbBY or DbBZ */
    desc.units[0] = DbUN_PHYS; /* in physical units */
    desc.samples[0] = 1000; /* get at most this number of samples */
    desc.reduction[0] = DbRED_AVERAGE; /* reduce by averaging */
    desc.gapFill[0] = DbGAP_NAN; /* fill gaps with IEEE NaN */
    result = DbGetData(db, &desc, (void **)&data);

    piped.samples[0] = desc.samples[0];
    piped.dimension = desc.dimension;
    piped.type = IsPIPE_FLOAT;
    IsCallPipe(UiDrawingWidget(pDrawing->part), "data", &piped, &data);
    desc.samples[0] = piped.samples[0];
    desc.dimension = piped.dimension;
```

```
    if (piped.type != IsPIPE_FLOAT)
IsError("Can't handle data type %d", piped.type);

    /* any analysis will be inserted here */

    piped.samples[0] = desc.samples[0];
    piped.dimension = desc.dimension;
    piped.type = IsPIPE_FLOAT;
    IsCallPipe(UiDrawingWidget(pDrawing->part), "plot", &piped, &data);
    desc.samples[0] = piped.samples[0];
    desc.dimension = piped.dimension;
    if (piped.type != IsPIPE_FLOAT)
IsError("Can't handle data type %d", piped.type);

    /* use an auto scaled plot */
    gd.label = pDrawing->label;
    gd.points = desc.samples[0];
    gd.x.label = NULL;
    gd.x.scale = GiTIME_SCALE; /* use time along x axis */
    gd.x.start = desc.start;
    gd.x.interval = desc.interval;
    gd.y.label = "nT"; /* y axis label */
    gd.y.scale = GiAUTO_SCALE;
    gd.y.align = 2.5; /* use multiples of 2.5 in auto scale
   (eg. 250, 500, 750, 1000, 2500 ...) */
    gd.y.data = data;
    gd.y.drop = desc.warning; /* pass data drop flag */
    if (result == DbSUCCESS) {
GiGraph2D(GkWsId(pDrawing->part), &gd);
DbFree(data); /* free data base allocated storage */
    } else {
gd.x.label = DbErrorString(result);
GiMessage(GkWsId(pDrawing->part), &gd);
    }
}
```

E.3 File bf.h

```
#define DRAWING_COUNT 3 /* bx, by & bz */

typedef struct _BfDrawing {
    int on; /* set if drawing is active */
    UiPart part; /* drawing part */
    int sensor; /* sensor used */
    char *label; /* plot label */
} BfDrawing;
```

E.4 File Imakefile

```
DEPLIBS = $(DEPUILIB) $(DEPGILIB) $(DEPGKLIB) $(DEPDBLIB) $(DEPISLIB)
LOCAL_LIBRARIES = $(UILIB) $(GILIB) $(GKLIB) $(DBLIB) $(ISLIB) \
  $(MOTIFLIB) $(XTLIB) $(XGKSLIB) $(X11LIB) \
  $(STDLIB)
```

```
INCLUDES = -I. -I$(INCLUDESRC) $(MOTIFINC) $(XTINC) $(X11INC) $(XGKSINC)

            SRCS = main.c update.c
            OBJS = main.o update.o

ComplexProgramTarget(bf)
RegisterClient(bf,$(TECHBINDIR),viking)
```

F bf (PHIGS) source code

F.1 File main.c

```
/************************************************************
 *
 *    main.c
 *
 *    Shows the three magnetic field components bx, by, bz.
 *    Each component can be selected individually by a menu.
 *    Each drawing has a system menu attached that is popuped
 *    by the <Ctrl>right mouse button.
 *
 *    Written Aug 3, 1990 by Anders Lundgren, IRFU
 *
 ************************************************************/
#include <stdio.h>
#include <math.h>
#include <Is.h> /* Isdat declarations */
#include <Ui.h> /* User interface declarations */
#include <Db.h> /* Data base declarations */
#include <instrument/Viking2.h>
#include <phigs.h>
#include "bfield.h"

/* define main layout: a menu bar and three drawing areas */
static char *layoutString = " \
    layout 'bfield' layoutPart=. { \n \
menubar 'Menu' menubarPart=.; \n \
drawing 'drawing0' drawing0=.; \n \
drawing 'drawing1' drawing1=.; \n \
drawing 'drawing2' drawing2=.; \n \
    } \
";

/* define select component menu: three toggle buttons */
static char *selectMenuString = " \
    menu 'select' { \n \
      'bx' -t -s SelectMenuCB(0); \n \
      'by' -t -s SelectMenuCB(1); \n \
      'bz' -t -s SelectMenuCB(2); \n \
    } \
";

/* forward declaration of callback functions */
static void SelectMenuCB();

/* fill in callback mapping table */
UiCallbackMap cbMap[] = {
    UiCB_MAP_ENTRY(SelectMenuCB)
    UiCB_MAP_END
};

/* define the part identifiers we need */
static UiPart layoutPart; /* top level part */
```

```
static UiPart menubarPart; /* menubar */
/* define an array of drawing parameters to keep track of bx, by and bz
   bx = 0, by = 1, bz = 2 */
static BfDrawing drawings[DRAWING_COUNT] = {
    {1, NULL, DbVIK2_BX, "BX"},
    {1, NULL, DbVIK2_BY, "BY"},
    {1, NULL, DbVIK2_BZ, "BZ"}
};

/* fill in identifier mapping table */
UiIdentifierMap idMap[] = {
    UiID_MAP_ENTRY(layoutPart)
    UiID_MAP_ENTRY(menubarPart)
    "drawing0", &drawings[0].part,
    "drawing1", &drawings[1].part,
    "drawing2", &drawings[2].part,
    UiID_MAP_END
};

Database *db; /* opened database */

/*
 * SelectMenuCB()
 *
 * Callback that is called when the users wants to
 * disable/enable a drawing.
 */
static void SelectMenuCB(part, reason, state, drawing)
UiPart part; /* drawing part */
int reason; /* callback reason */
int *state; /* set to one if enable */
int drawing; /* which drawing */
{
    IsTmInfo *info;

    if (*state) { /* turn on drawing */
info = IsGetTmInfo();
drawings[drawing].on = 1;
UiMapPart(drawings[drawing].part);
UpdateBfield(&drawings[drawing], info);
    } else { /* turn off drawing */
drawings[drawing].on = 0;
UiUnmapPart(drawings[drawing].part);
    }
}

/*
 * UpdateTimeCB()
 *
 * Callback that is called when the orbit manager
 * has sent us new time information.
 */
static void UpdateTimeCB(reason, tmInfo, closure)
int reason; /* callback reason */
IsTmInfo *tmInfo; /* orbit manager information */
IsPointer closure;
```

```
{
    int i;

    for (i = 0; i < DRAWING_COUNT; i++) UpdateBfield(&drawings[i], tmInfo);
}

/*
 * main()
 *
 * The program starts here.
 */
main(argc, argv)
int argc;
char **argv;
{
    int i; /* loop counter */
    Display *dpy; /* opened X display */

    dpy = UiInitialize(argc, argv); /* initialize user interface library */
    IsInitialize(argc, argv, dpy); /* initialize isdat library */

    /* create the basic layout */
    if (!UiCreateLayout(NULL, layoutString, idMap, cbMap)) exit(1);

    /* create system menus on each drawing */
    for (i = 0; i < DRAWING_COUNT; i++) {
IsCreateSystemMenu(UiDrawingWidget(drawings[i].part));
    }

    /* attach select menu to the menubar */
    if (!UiCreateMenu(menubarPart, selectMenuString, idMap, cbMap)) exit(1);

    /* make layout visible */
    UiMapLayouts();

    /* add callback for orbit manager information */
    IsAddCallback(IsCR_TM_INFO, UpdateTimeCB, NULL);

    if (PhOpenPhigs(dpy)) {
fprintf(stderr, "No PHIGS support on this system\n");
exit(1);
    }
    /* open a Phigs workstation on each drawing */
    for (i = 0; i < DRAWING_COUNT; i++) {
drawings[i].ws_id = PhWsId(drawings[i].part);
/* only one structure is used for each workstation */
drawings[i].struct_id = drawings[i].ws_id;
PhOpenWs(drawings[i].ws_id, UiDrawingWidget(drawings[i].part));
InitPlot(&drawings[i]);
    }

    /* Get in touch with data base handler */
    if ((db = DbOpen(NULL, argc, argv)) == NULL) {
IsError("cannot connect to database %s", DbName(NULL));
    }
```

```
    /* enter main loop and never return */
    IsMainLoop();
}
```

F.2 File update.c

```
#include <stdio.h>
#include <math.h>
#include <Is.h> /* Isdat declarations */
#include <Ui.h> /* User interface declarations */
#include <Db.h> /* Data base  declarations */
#include <instrument/Viking2.h>
#include <phigs.h>
#include "bfield.h"

extern Database *db; /* opened database */

/*
 * UpdateBfield()
 *
 * Gets the data from the specified time and interval,
 * generates a plot using the autoscale function.
 */
void UpdateBfield(pDrawing, tmInfo)
BfDrawing *pDrawing; /* current drawing */
IsTmInfo *tmInfo; /* orbit manager information */
{
    int result;
    DbDataDesc desc; /* data description */
    float *data;

    /* describe the data we want */
    desc.start = tmInfo->start;
    desc.interval = tmInfo->interval;
    desc.spec.project = tmInfo->project;
    desc.spec.member = tmInfo->member;
    desc.spec.instrument = DbVIK_V2; /* viking V2 */
    desc.spec.sensor = pDrawing->sensor; /* DbVIK2_BX, DbVIK2_BY or DVIK2_bBZ */
    desc.units[0] = DbUN_PHYS; /* in physical units */
    desc.samples[0] = 1000; /* get at most this number of samples */
    desc.reduction[0] = DbRED_AVERAGE; /* reduce by averaging */
    desc.gapFill[0] = DbGAP_NAN; /* fill gaps with IEEE NaN */
    result = DbGetData(db, &desc, &data);
    if (desc.warning) return;

    DoPlot(pDrawing, desc.samples[0], data);
    DbFree(data);
}
```

F.3 File doplot.c

```
#include <stdio.h>
#include <math.h>
#include <Is.h>                 /* Isdat declarations */
```

```
#include <Ui.h>                 /* User interface declarations */
#include <phigs.h>
#include "bfield.h"

void DoPlot(pDrawing, n, data)
BfDrawing *pDrawing;
int n;
float *data;
{
    int i;
    int j;
    Pint err;
    Ppoint3 fixed;
    Pvec3 shift;
    Pvec3 scale;
    Pmatrix3 mat;
    Ppoint_list3 plist;
    Ppoint3 polyData[5000];

    for (i = 0; i < n; i++) {
polyData[i].x = i;
polyData[i].y = data[i];
polyData[i].z = 0.0;
    }
    popen_struct(pDrawing->struct_id);
    pempty_struct(pDrawing->struct_id);
    fixed.x = 0.0; fixed.y = 0.0; fixed.z = 0.0;
    shift.delta_x = 0.0; shift.delta_y = 0.5; shift.delta_z = 0.0;
    scale.delta_x = 1.0 / n; scale.delta_y = 1.0 / 8000.0; scale.delta_z = 1.0;
    pbuild_tran_matrix3(&fixed, &shift, 0.0, 0.0, 0.0, &scale, &err, mat);
    pset_local_tran3(mat, PTYPE_REPLACE);
    plist.num_points = n;
    plist.points = polyData;
    ppolyline3(&plist);
    pclose_struct();
    predraw_all_structs(pDrawing->ws_id, PFLAG_ALWAYS);
}

void InitPlot(pDrawing)
BfDrawing *pDrawing;
{
    Pcolr_rep colr; /* colr rep. */

    colr.rgb.red = 1.0; colr.rgb.green = 1.0; colr.rgb.blue = 1.0;
    pset_colr_rep(pDrawing->ws_id, 0, &colr);
    colr.rgb.red = 0.0; colr.rgb.green = 0.0; colr.rgb.blue = 0.0;
    pset_colr_rep(pDrawing->ws_id, 1, &colr);
    ppost_struct(pDrawing->ws_id, pDrawing->struct_id, 1.0);
}
```

F.4 File bf.h

```
#define DRAWING_COUNT 3 /* bx, by & bz */
```

```
typedef struct _BfDrawing {
    int on; /* set if drawing is active */
    UiPart part; /* drawing part */
    int sensor; /* sensor used */
    char *label; /* plot label */
    Pint ws_id;
    Pint struct_id;
} BfDrawing;
```

F.5 File Imakefile

```
        DEPLIBS = $(DEPUILIB) $(DEPPHLIB) $(DEPDBLIB) $(DEPISLIB)
LOCAL_LIBRARIES = $(UILIB) $(PHLIB) $(DBLIB) $(ISLIB) \
  $(MOTIFLIB) $(XTLIB) $(PHIGSLIB) $(X11LIB) \
  $(STDLIB)
       INCLUDES = -I. -I$(INCLUDESRC) $(MOTIFINC) $(XTINC) $(X11INC) $(PHIGSINC)

           SRCS = main.c update.c doplot.c
           OBJS = main.o update.o doplot.o

ComplexProgramTarget(bf.phigs)
RegisterClient(bf.phigs,$(TECHBINDIR),viking)
```