ISDAT Programmers Guide
4. Filters

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

May 4, 1994

Document Status Sheet			
1. Document Title: **ISDAT Programmers Guide Filters**			
2. Document Reference Number: **CWD-IPG-004**			
3. Issue	4. Revision	5. Date	6. Reason for Change
Draft	0	94 March 3	New document.

Contents

1 Introduction

1.1 Purpose of the document

The purpose of this document is to provide the information needed to write filters for the scientific analysis to be implemented in the ISDAT system. The document is intended for the programmer. A general introduction to the ISDAT is given in [Ref. 1]. An overview of documents related to coding and general guidelines are given in [Ref. 3].

1.2 Scope of the software

The scope of the ISDAT scientific filters are to provide run-time modifiable tools for the scientific analysis of the ISDAT clients.

1.3 Definitions and acronyms

Acronym	Meaning
DBH	Data Base Handler
ISDAT	Interactive Science Data Analysis Tool
N/A	Not Applicable
TBD	To Be defined
TBW	To Be Written

2 Filter functions

Filters are intended to provide the user with an uniform set of tools to facilitate run-time modifiable data handling. The filters are functioning as "pipes" in the process. The implementation of pipes in the clients is described in [Ref. 2]. Examples of filter functions are:

- One way pipes to external files.
- LP or HP digital filtering.
- spline functions.
- Arithmetic operations like add or multiply.
- Subtraction of average level or trend.
- Export to external software packages.
- Import of external data.

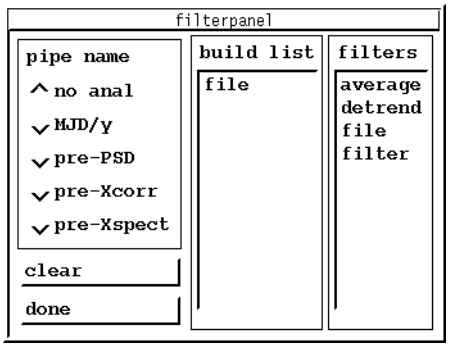

Figure 1: Filter window

3 Filter coding

3.1 Activating a filter

If properly implemented in the client, a filter is activated by doing a $< ctrl >$ left button click in the graphic panel. You will then get a menu containing the *filter* option. Click on the filter and you will get the window shown in Figure 1. You then construct the desired pipe by selecting the appropriate filters in the menu to the right. By *double clicking* on the *file* in the pipe field, you will get the window of Figure 2 where you can specify the file parameters.

3.2 Design

The filter design will be explained by commenting on a filter that reads from or writes to a file. The filter is called *file* and the full source code is listed in Appendix A. How to include piping in the client is described in [Ref. 2]. The basic design of a filter is very similar to the design of a client. Here we will only discuss features that are specific for filters. For a detailed discussion of the general design, the reader is referred to [Ref. 2].

First we include the standard header files:

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

Here the Is.h contains an important structure:

```
typedef struct _IsPipeDesc {
    int  dimension;           /* Max 5, in                          */
    int  typeOfData[5];       /* one for each dim, in               */
    int  samples[5];          /*number samples for each dim, in/out*/
    int  filterId[8];         /* Max 8 filter in a sequence, out    */
    char filterDesc[8][128]   /* Text descr of the filter, out      */
    char textToFilter[128];   /* string to the filter, in           */
    int  filterWarning[8];    /* Warning, out                       */
    char warningText[8][128]; /* Warning text, out                  */
}IsPipeDesc;
```

The FilterPopupCB is the "double click" filter specification window callback mentioned above.

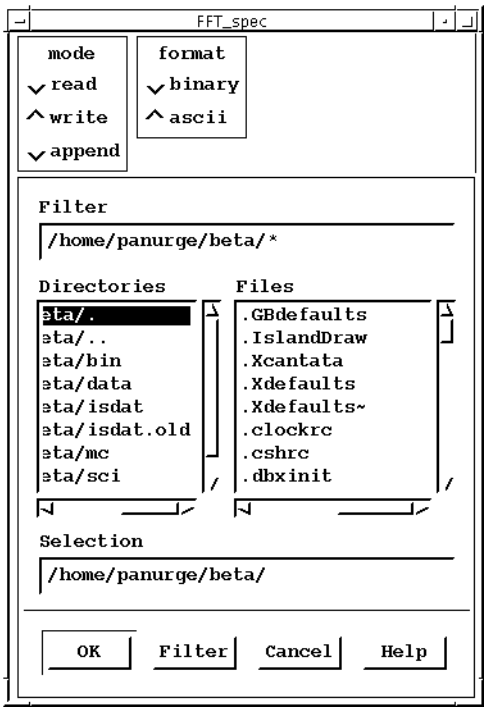

Figure 2: File specification window

The *main* process begins with a call:

```
IsFilter()
```

to identify this program as a filter.

Before entering the main loop we have to make the call:

```
sprintf(string,"file %s", filename};
IsUpdatePipeStruct(IsFilter,string);
```

that fills in the filterId and filterDesc members of the pipeDesc structure. It is assumed that the client has reset the pipeDesc structure before entering the pipe by the call IsResetPipeStruct().

3.3 Documentation

[to be written]

3.3.1 Filter installation

[to be written]

References

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

[2] G. Holmgren and A. Lundgren. ISDAT programmers guide. 3. clients. Technical report, IRF-U, February 1994.

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

A Source code list

A.1 File main.c

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

static char *controlString = {"                                          \
    layout 'FFT_spec' -f controlPart=. {                                 \n \
        frame 'mode' -l -r {                                             \n \
            button 'read'  ModeCB(0);                                   \n \
            button 'write' -s ModeCB(1);                                     \n \
            button 'append' ModeCB(2);                                    \n \
        }                                                                \n \
        frame 'format' -l -r {                                           \n \
            button 'binary'  FormatCB(1);                               \n \
            button 'ascii' -s FormatCB(0);                                   \n \
        };                                                               \n \
        frame 'Data spec' {                                              \n \
            file '' FileSelectionCB()                                    \n \
        };                                                               \n \
    }                                                                    \
"};

/* forward declaration of callback functions */
static void ModeCB();
static void FormatCB();
static void FileSelectionCB();

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

/* define the part identifiers we need */
static UiPart controlPart; /* control part */

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

static FILE *filep = NULL;
static int writeFlag = 1;
static int binaryFlag = 0;
static char filename[256];

void FilterPopupCB(reason, call, closure)
int reason;                     /* callback reason */
```

```
IsPointer call;
IsPointer closure;
{
    UiMapPart(controlPart);
}

void PendingDataCB(reason, call, closure)
int reason;                     /* callback reason */
IsPointer call;
IsPointer closure;
{
    int i;
    int items;
    IsPipeDesc piped;
    float *buf;

    IsPipeRead(&piped, &buf);
    filep = (FILE *)0;
    if (filename[0]) {
if (writeFlag==1) filep = fopen(filename, "w");
else if(writeFlag==2)
filep = fopen(filename, "a");
else
filep = fopen(filename, "r");
    }
    if (filep) {
if (writeFlag>0) {
    WriteToFile(filep, buf, piped.samples[0]);
} else {
    items = ReadFromFile(filep, buf, piped.samples[0]);
    if (items) piped.samples[0] = items;
}
fclose(filep);
    }
    IsPipeWrite(&piped, buf);
}

static void ModeCB(part, reason, state, item)
UiPart part;                    /* drawing part */
int reason;                     /* callback reason */
int *state;
int item;                       /* which item */
{
    writeFlag = item;
}

static void FormatCB(part, reason, state, item)
UiPart part;                    /* drawing part */
int reason;                     /* callback reason */
int *state;
int item;                       /* which item */
{
    binaryFlag = item;
}

static void FileSelectionCB(part, reason, name, closure)
```

```
UiPart part;
int reason;
char *name;
int closure;
{
    if (name) {
strcpy(filename, name);
    } else {
filename[0] = '\0';
    }
    UiUnmapPart(controlPart);
}

int WriteToFile(fp, buf, n)
FILE *fp;
float *buf;
int n;
{
    int i;

    if (binaryFlag) {
fwrite(buf, sizeof(float), n, fp);
    } else {
for (i = 0; i < n; i++) fprintf(fp, "%e\n", buf[i]);
    }
}

int ReadFromFile(fp, buf, n)
FILE *fp;
float *buf;
int n;
{
    int i;
    char str[256];

    if (binaryFlag) {
return fread(buf, sizeof(float), n, fp);
    } else {
for (i = 0; i < n; i++) {
    if (!fgets(str, 256, fp)) break;
    sscanf(str, "%e", buf + i);
}
return i;
    }
}

main(argc, argv)
int argc;
char **argv;
{
    Display *dpy; /* opened X display */

    IsFilter();
    dpy = UiInitialize(argc, argv); /* initialize user interface library */
    IsInitialize(argc, argv, dpy);
    IsAddCallback(IsCR_FILTER_POPUP, FilterPopupCB, NULL);
```

```
    IsAddCallback(IsCR_PENDING_DATA, PendingDataCB, NULL);

    if (!UiCreateLayout(NULL, controlString, idMap, cbMap)) exit(1);
    UiUnmapPart(controlPart);
    /* make layout visible */
    UiMapLayouts();

    IsMainLoop();
}
```

A.2 File Imakefile

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

           SRCS = main.c
           OBJS = main.o

ComplexProgramTarget(file)
LinkFiles(all,file,$(FILTERDIR))
```