2. Custom Input and Output

2.1 Input Pipes

There are two different command line formats used by the IRIS input process to launch an input pipe. In **setup**, these are called "Pathnames" and "Pipe". Sigmet recommends that you code all new pipes using the more flexible Pathnames scheme. In this scheme, the pipe program is invoked with the following command line:

```
$ <pipe-path> -if:<in-file> -ip:<in-path> -op:<out-path> -device:<number>
```

In this case, your pipe must explicitly open the input and output files. The purpose of the "–if" argument (which is somewhat redundant with "–ip") is to allow the pipe program to make nice diagnostic printouts, or to parse the filename for information, without having to search the pathname for the final "/".

The only information passed back to the calling program is the 8-bit exit status. In normal cases is should return EXIT_SUCCESS (0). If it returns EXIT_FAILURE (1), then IRIS will signal a messages saying "Error running pipe, check log". It is important for the pipe program to write diagnostic information to a log file. Sigmet suggests writing to the ${IRIS_LOG} directory.

There are 2 other possible exit returns: EXIT_RERUN (2), and EXIT_NOTHING (3). EXIT_RERUN indicates that the pipe has both produced a normal output, and wants to product a second output file. In this case, the pipe needs to store information about the second output on disk. Sigmet suggests that you write this in the ${IRIS_TEMP} directory. IRIS will process the first output, the invoke the pipe again with the following command line:

```
$ <pipe-path> -rerun -op:<out-path> -device:<number>
```

Note that the input path is gone, and the option "–rerun" is supplied. When the pipe is thus called, it can retrieve the stored information and write the second output to the specified pathname.

EXIT_NOTHING indicates that the pipe ran without error, but there was no output produced. IRIS will ignore and delete the output file. This is useful for a pipe which needs to read several input files before producing one output file. An example of such a pipe is the Rainbow input pipe RainbowToIris, found in the directory ${IRIS_ROOT}/utils/rainbow.

2.2 User Product Insert

Customers are generate their own IRIS products, and to process IRIS products within their own programs. The file formats are fully documented in this manual. The difficulty is often in passing the product between the user program and IRIS.

One way to do this is by using IRIS's input and output mechanisms with no handshaking. This scheme uses polling. For example the user places a product file in a know directory, IRIS checks every few seconds. When it finds a file it reads it in. Similarly, IRIS's output can place a file in a know directory, and the user's program can poll for it. When using a polling scheme, it is always desirable to first copy the file there with a "." prefix. Once the file is fully there, rename it to a name without the dot. This prevents the recipient from seeing a partial file.

User Product Insert (UPI) is an alternative mechanism which sends a message to notify the recipient of a product. This removes all time lag and cpu usage due to polling. The user routine communicates with IRIS in exactly the same manner in which different IRIS host systems communicate with each other. The user routine can execute on the same computer as the IRIS host, or it can be on a different CPU because the communication is via network socket routines.

SIGMET supplies a sample routine to get you started. The routine is written in C++, and uses sockets to communicate with a host IRIS system.

2.3 Building the Example Programs

There are three example programs shipped with the iris release media to demonstrate the UPI process. These are all in the utils/examples directory, and are called upi_rcv, upi_xmt, and change_product. The upi_rcv program simply receives a file from IRIS, prints out the file name, then deletes it. It can easily be customized to your requirements. The change_product program will modify an existing product file to add in a circular target. The upi_xmt program sends files back to IRIS.

To compile these programs, first configure your IRIS compiling environment as discussed in section 1.2. Then go to the directory containing the source code: ${IRIS_ROOT}/utils/examples. Compile it using the resident Makefile, as follows:

```
$ make upi_rcv
```

You may want to rename it, copy it elsewhere, and make a script file to compile it. You can take the compile and link commands from the makefile.

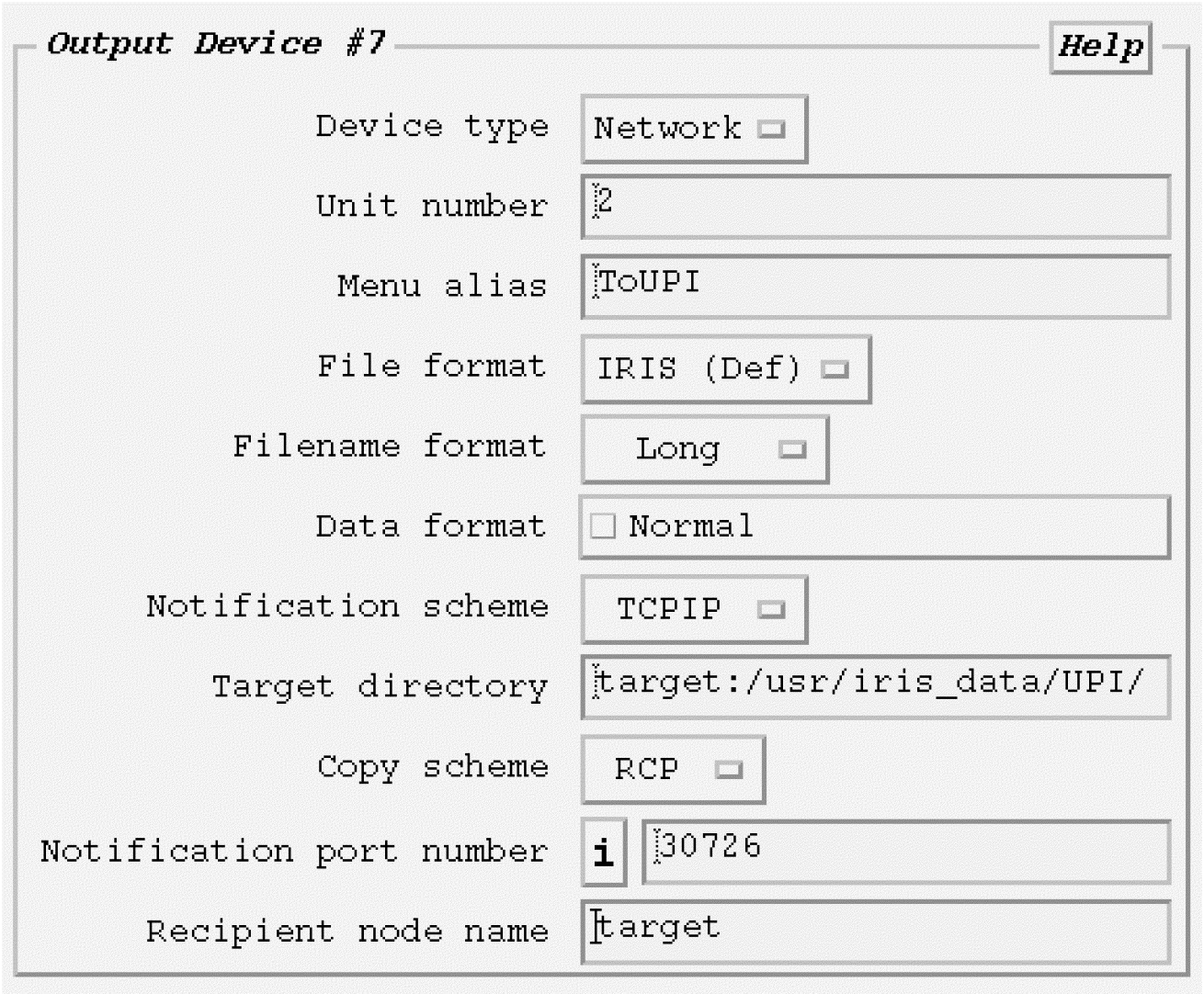
2.4 Running the upi_rcv Program

These instructions for running the sample program assume that you are running upi_rcv on a computer node called "target", and that IRIS is running on a computer called "host". Running on a single computer is somewhat simpler.

First create the directory on target into which IRIS can copy products. A typical example might be /usr/iris_data/UPI. The actual directory name is not important.

Copy the upi_rcv to the target computer (if it is different from where you compiled it) so it can run there. You will need to set the mode again after copying.

On the IRIS host system, create a new network output device called "ToUPI" for sending products to the upi_rcv program. This is all done from the **setup** utility – an example is shown below:



Next, on the target computer, run the upi program:

```
$ upi_rcv -d /usr/iris_data/UPI
```

It should print something like the following:

```
Initializing UPI receiver
Connection established
Received file: '/usr/iris_data/UPI/host950710171113.PPI'
Connection closed
```

Finally, using the Product Output menu, send a horizontal reflectivity product to "ToUPI".

2.5 IRIS Product and Data Types

User generated products must fit into one of the IRIS product and data types. A good way to help understand these is to look at the display of a typical square product (such as a PPI) on a medium sized window. The product type is used by IRIS to decide in general the interpretation of the image, and the data type is used to figure out how to convert between data number and colors.

The data type in a product file controls the color legend (the box containing an array of colored rectangles with text next to them) The data type is converted to text to label the legend, and is used internally to help control how data values in Cartesian products are converted to colors. Some data types do not display a color legend. Examples of these are products like WARN, SLINE, TRACK, and VVP displayed as graphs.

The product type in a product file controls some of the text legend as well as the geometry. Based on the product type, IRIS divides products into several geometrical categories which determine what kind of overlays can be drawn on the image:

- **Horizontal:** These products (such as CAPPI) can have political overlays, product overlays, range rings, and latitude/longitude grids drawn on them. The user cursor works ok. There are minor variations within this category, such as: constant elevation (e.g. PPI), constant height (e.g. CAPPI), and no height implied (e.g. Echo tops).
- **Vertical:** These products (such as Cross Section) can have a range/height overlay grid. The user cursor works ok.
- **None:** These products (such as VVP) have not geometrical interpretation. There are no overlays drawn, and the user cursor does not return information.
- **No Display:** These products (such as RAW) cannot be displayed.

The definitions for the data type parameters are found in the include/dsp_lib.h file. The are also discussed in more detail in section 3.3.

The definitions for the product types are found in the include/product.h file.

2.6 Customizing the Legend

The legend on user defined products can be customized to some extent. The text displayed in the 5-line legend box on the window legend can be overridden as well at the 6 lines above the time on the old-medium sized legend, and the 4 lines above the time on the old-small sized legend. This is controlled by the OUTPUT_LEGEND.DAT file. An example of this file is below. As you can see, the format is more complicated. The column headed by "R" stands for resolution, the number of characters on the line. Low resolution windows can fit only 13 characters, while medium and high resolution can fit 17 characters. Therefore, in general you

must add two definitions for each string you want to control. However, IRIS defaults to the 11-character format for bigger displays if no 17-character entry is defined. The column headed by "L" stands for line: "1" means line 7 on the text legend, and "2" means line 8 on the text legend. The first entry in the table is the product type, which must be USER, VUSER, or OTHER.

The general pattern of operation is as follows: When IRIS draws the legend and the product type is either PROD_USER, PROD_USERV, or PROD_OTHER, it searches this table. If it finds a match of the product type, product name, resolutions, and line number, it is considered a match. If it matches everything except the product name, then the first such match in the table is taken as a default for all product names, and is considered a match. If it would otherwise match, except that the desired resolution is larger than the table, it is considered a match. If no match is found, it displays blank.

If the flag is set to "SCALE", a 32-bit word is grabbed from the first byte of the product header "product_specific_info" field offset by the offset in the file. This number is converted to floating point, and multiplied by the scale factor in the file. The C library routine **`sprintf`** is called with the format statement and the resulting number as arguments. The resulting string is displayed on the legend.

If the flag is set to "REF", a pointer to the product header "product_specific_info" field offset by the offset in the file is passed to **`sprintf`**. This is useful if you want to display a character string from the header. The resulting string is displayed on the legend.

If the flag is neither of the above, a 32-bit word is grabbed from the first byte of the product header "product_specific_info" field offset by the offset in the file without any conversion. The C library routine **`sprintf`** is called with the format statement and the 32-bit number as arguments. The resulting string is displayed on the legend.

```
# File: output_legend.dat
# Format:
# Valid flags are: SCALE, REF, <anything else>
# R=resolution, L=line, O=offset
#ptype  productname  R  L O  scale   flag  format
USER    DEFAULT     11 1 4  0.00001 SCALE "Hght:%3.1f"
USER    Z_010_120   11 1 4  0.00001 SCALE "Hght:%3.1f"
USER    DEFAULT     11 2 8  1       -     "Flag:%d"
USER    Z_010_120   11 2 8  1       -     "      "
VUSER   DEFAULT     11 1 4  1       -     "      "
USER    DEFAULT     17 1 4  0.00001 SCALE "Height:%3.1f"
USER    Z_010_120   17 1 4  0.00001 SCALE "Height:%3.1f"
USER    DEFAULT     17 2 8  1       -     "Flag:%d"
USER    Z_010_120   17 2 8  1       -     "      "
VUSER   DEFAULT     17 1 4  1       -     "      "
```

2.7 The NORDRAD Area Definition File

You want to make a configuration file to control converting between NORDRAD areas and IRIS sites. This file is called NORDRAD_AREAS.DAT, and it is in the IRIS_CONFIG directory. An example of this file is shown in Figure 2–1.

Figure 2–1: Example of NORDRAD_AREAS.DAT

```
# Example file for NORDRAD_AREAS.DAT file
# This controls the scale used when converting to the RRSRT_Z product:
rainfall_accumulation_span 0.01 100.0
# This controls the height (in km) used when converting a NORDRAD
# CAPPI to IRIS if the Height attribute is missing from the header.
default_cappi_height 1.0
# Network access information used for the NORDRAD API.
# Either 1 or two devices can be defined.  The output device
# number is used to select which to use.  For input, the first
# device is used.
api_device_number   1 2
api_access_control ”operator xxxxxx” ”operator xxxxxx”
api_host_name      ”haze”          ”haze”
api_receiver_id    ”iris_trans”    ”iris_trans”
# Below there are two mapping tables used to convert
# between IRIS sites, and NORDRAD areas.  One map for each direction
# Mapping of NORDRAD areas to IRIS sitenames
#       areaname sitename       wavelength in 1/100 of cm
ntoi_map FIABO1   ”MASKU”       500
ntoi_map FIABO2   ”MASKU with s” 500
ntoi_map FIABO3   MASKU         500
ntoi_map FIABO4   MASKU         500
ntoi_map FIABO6   MASKU         500
ntoi_map FIHEL    Helsinki      500
ntoi_map FIROVMTA COMPOSITE     500
ntoi_map FIROV1   ROVANIEMI     500
ntoi_map FIROV2   ROVANIEMI     500
ntoi_map FIAB01   ”UND Huntsville” 500
ntoi_map FIAB03   ”SIGMET, haze” 500
ntoi_map FIHOT1   ”SIGMET, HOT”  500
ntoi_map FIDRY    ”SIGMET, dry”  500
ntoi_map FIHOT2   ”SIGMET, HOT”  500
ntoi_map FISTO3   STOCKHOLM     500
ntoi_map SVNKP2   NORRKÖPING    500
ntoi_map SVOVIK1  OVIK          500
ntoi_map SVOVIK2  OVIK          500
# Mapping of IRIS sitenames to NORDRAD areas.
# There are two kinds of entries, iton_prod and iton_map
# Map based on sitename, product type, and product name.
# If there is an exact match here, take it.  This overrides
```

```
# the iton_map lines.  The first match will control.  Size
# and resolution are not used for matching.
#        areaname sitename ptype pname   h–size v–size resolution in meters
iton_prod FIABO1   ”MASKU”  CAPPI DEFAULT 150000 150000 1000
# Map based on sitename and resolution.  For multiple areas from
# the same sitename,  look for the closest in resolution.  If
# there is a resolution tie, then look for the closest in size.
#       areaname sitename      h–size   v–size  resolution in meters
iton_map FIABO1   ”MASKU”       150000   150000   1000
iton_map FIABO2   ”MASKU with s” 300000  300000   1000
iton_map FIROV1   ROVANIEMI     300000   300000   1000
iton_map FIROV2   ROVANIEMI     600000   600000   1000
iton_map FIAB01   ”UND Huntsville” 100000 100000   1000
iton_map FIAB03   ”SIGMET, haze” 200000  200000   1000
iton_map FIHOT1   ”SIGMET, HOT” 200000   200000   1000
iton_map FIDRY    ”SIGMET, dry” 200000   200000   2000
iton_map FIHOT2   ”SIGMET, HOT” 200000   200000   2000
iton_map FIVAN1   VANTA         480000   480000   1000
iton_map FIVAN10  VANTA         480000   480000   1000
iton_map SVKKR1   KARLSKRONA    480000   480000   1000
iton_map SVKKR2   KARLSKRONA    240000   240000   1000
iton_map SVOVIK1  OVIK          480000   480000   1000
iton_map SVOVIK2  OVIK          240000   240000   1000
```

The NORDRAD_AREAS.DAT file consists mainly of a list of all the NORDRAD areas with an associated IRIS site name. The file is used for conversions both ways between IRIS and NORDRAD products. When a NORDRAD product is converted to an IRIS product, the NORDRAD area name is looked up in the ntoI_map list, and the associated IRIS site name is used. There is only one entry for each area in the table. The radar wavelength in the file is put into the IRIS header. The wavelength and PRF are needed to display velocities and widths in IRIS.

When an IRIS product is converted to a NORDRAD product, the iton_map list is searched for site names that match. IRIS looks for the closest match between the product size and the size in the file. IRIS converts the product to the exact size (in meters) of the NORDRAD area by either clipping the edges or padding with blank. If you want the product to be properly handled by NORDRAD, you must make the area size match. It is slightly less important to make the resolution in pixels match. Formally, the NORDRAD area defines only a rectangle on the earth, not the resolution of the data. However to do compositing, the resolutions have to be converted, and it may be easier to start at the expected resolution. If there is an exact match to a iton_prod map, then that overrides the iton_map table.

The file can optionally include a line starting with “rainfall_accumulation_span”. This is used to control the scaling of rainfall accumulation data when it is converted to NORDRAD format. The two numbers are the minimum and maximum rainfall in mm. Rainfall is converted to NORDRAD using the formula:

FILE = (log10(Physical) – logMIN) / ((logMAX–logMIN)/254)

The minimum must be between 0.001 and 1000.0, the maximum between 0.01 and 10000.0, and the span must be at least a factor of 10. If there is an error, IRIS will signal when it reads the file. If there is no such line, IRIS defaults to a span of 0.1 to 1000.