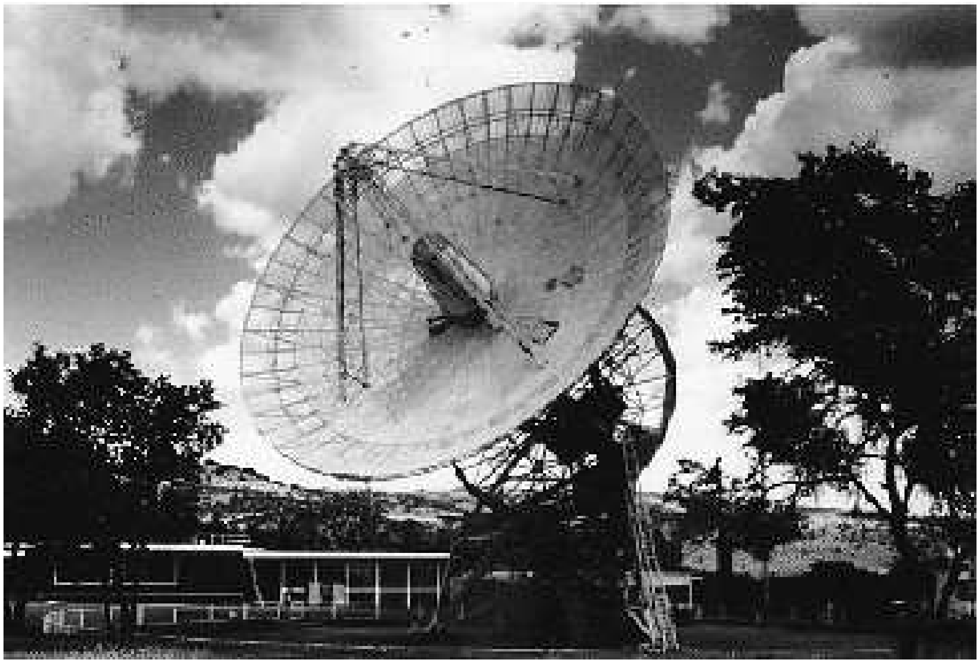
Hartebeesthoek Local Guide
for
Device Server Programmer's (DSP)
version 1.0

Andy Götz

May 24, 1996



LOCAL GUIDE for DSP
LOCAL GUIDE for DSP
LOCAL GUIDE for DSP
LOCAL GUIDE for DSP
LOCAL GUIDE for DSP
LOCAL GUIDE for DSP
LOCAL GUIDE for DSP

Contents

1 Introduction

This document gives practical details on how to write a device server at Hartbeesthoek.

2 Writing Device Servers

Anyone planning to write a device server should be in possession of or have access to two basic documents - *Device Server Programmer's Manual*, the *Automatic Class Generator* and the *Local Guide* (this document). Together these documents describe everything about writing device servers without going into site specific details. Although these documents are fairly uptodate they were not written at the same time and therefore the first one (the programmer's manual) does not refer to the second one (automatic class generator). Do not be put off by this, simply read them in their chronological order.

Whatever you do **READ BOTH THESE DOCUMENTS** before starting to write your first device server !

Once you have read these two documents return to this document and read on. The seven stages in writing a device server are explained in chapter 5 of the DSPM (cf. figure 4). Follow these stages and you should arrive at a device server.

Code is best produced by using the automatic class generator. This means that once you have decided on what will constitute a device and defined the device commands (action, input and output parameters) the next step is to prepare an input file for the class generator e.g. *class.dat*. The format of this file is described in the class generator document. Although the class generator is a very useful (I would almost say indispensable) tool neither it nor the document are perfect. Some things to be aware of are :

- the class name as it appears after the *classname* keyword is used as is in the code and is therefore case sensitive i.e. start all new words and/or abbreviations with a capital letter.
- all object fields entries need all keywords to be specified for correct code to be generated (e.g. even if the field in question does not require a resource it must be specified).
- some types are not correctly supported e.g. the generator does not know how to generate code for a variable length array of doubles (D_VAR_DOUBLEARR) neither for an unsigned long type (D_ULONG_TYPE), and the code has to be modified manually or specify the real type (DevVarDoubleArray and DevULong).

At HartRAO we only have the *ascii* version of the code generator for SunOS. It can found in the directory `/usr/local/bin` under the name `classgen`. Before running it make sure that the environment variable `TEMPLATEPATH` is pointing to the directory where the template files are stored -

`/usr/local/src/dserver/classgen/classes/template`

To run classgen type `classgen file-name` on `apus` or `bootes`.

The root directory of the output file to be generated should be

`/usr/local/src/nccs/dservers/classes/myclass` (where `myclass` is the name of the new class).

The class generator will generate the following files :

1. `Makefile` - the multi-platform *GNU* make compatible makefile for making the server and client.
2. `Class.c` - the source file for implementing the class, this file needs to be edited to add the source code for the individual commands.
3. `ClassP.h` - the private include file.
4. `Class.h` - the public include file, this file will require editing to add any public symbols or constants which are required by client and server alike.
5. `Class_menu.c` - a simple test menu program to test all the commands, this file will require editing if the server uses any "exotic" types and to pretty format the menu.
6. `startup.c` - a standard startup routine (which will be called by `main()` for creating, initialising and exporting a list of devices of this class, this file will need editing if any non-standard features are required by the server e.g. creating devices of different classes, special hardware support.
7. `Class.res` - an example resource file.
8. `Class.tex` - a user's guide in Latex format which can be Latex'ed and distributed to application programmer's, this file will require editing if the input file descriptive text was kept terse.

All these files are generated from a set of template files which can be found in `$TEMPLATEPATH/simple` in the different subdirectories (`src`, `include`, `ug` and `res`). These files can be modified (by a device server guru) if necessary to generate different code if a bug is detected or HartRAO wants to simply customize the code to their liking.

3 Make

Once the code has been generated *make* should be used to compile and link it. The makefile has been generated for the *GNU make* program and makes use of extensions specific to it. It is a multi-platform makefile which means that there is only one makefile for compiling on different platforms. Two platforms are supported so far - Linux and SunOS. All tests were done on Linux. Compiling under SunOS has not been tested and has to be correctly setup before being used e.g. the dserver libraries have only been compiled for Linux-386 so far. Differences between platforms e.g. different compiler options, are supported by enclosing them in conditional make statements of the form :

`ifdef linux`
linux specific make rule
`endif`

To use the make file you must call it with the script `linuxmake` (installed in `/sr/local/bin`). This script does the following :

```
make unix=1 linux=1 $@
```

Which calls make with the two variables unix and linux set.

To compile the newly generated class type `linuxmake Class.o` in the `~class/src` directory.

To compile and link the device server type `linuxmake Class` (assuming your device server is called Class) in the `~class/src` directory.

To make everything type `linuxmake all` in the `~class/src` directory.

To generate the documentation (the user guide) go to the `~class/ug` directory and type `latex Class` then `dvips Class` and print the resulting postscript file `Class.ps`.

4 The Resource Database

At this point you should have a device server ready to run. The next step is then to declare the device server in the resource database and which devices it will serve and configure any resources required by the devices.

Fundamental to the device server concept is the notion of a device. Every device server which wants to run has to be declared in the resource database so that it has an entry in the `NAMES` table. If the device server is not going to export any devices then a dummy device name has to be given.

The resource database is served by a database server (of course) called `dbm_server`. The database is based on the *GNU dbm* routines. Each database table is stored in a separate dbm "database" and consists of two files `TABLE.pag` and `TABLE.dir` where TABLE is the name of the table. Tables are specified in an environment variable called `$DBTABLES`.

4.1 Where is all the database stuff ?

The database utilities are stored in `nccs:~dserver/system/bin/linux` Add this directory to your `PATH`.

In this directory you will find a script called `dbm_env` which sets up all the necessary environment variables. This script should be called at login time using the command `source` if you are running `csh` or `.` (dot) if you are using `bash`.

The database tables are stored in `$DBM_DIR` which is set to `~dserver/dbase` on `nccs`.

A special table called `HARTRAO` has been created for Hartebeesthoek[1]. All NCCS devices should be created in this domain. If necessary other domains can be added in the future.

The database resources are stored in `$DBM_RESDIR ~dserver/res` on `nccs`. A subdirectory called `hartrao` has been created for HartRAO specific resources and most new files will be created here.

4.2 Editing the resource database

To edit/create a resource file take you favourite editor and create a file in `$DBM_RESDIR/hartrao`. Use the template file which the class generator has generated for you in `~class/res`

[1]an alternative would be *KGAMMA*, the Setswana word for Hartebeest, once HartRAO becomes more African

as an example. When choosing device names a good practice is to define some local rules. At the ESRF we often use the convention of DOMAIN/FAMILY/nn where DOMAIN refers to the domain or control system, family refers to the type of device and nn is an integer from 1 to the number of devices in the control system. An example for an HP3330 frequency synthesiser would be HARTRAO/HP3330/1 and the resource file might look like this :

```
#
# resource file for HP3330 frequency synthesiser
#
# andy - 9/5/96
#
hp3330/mike/device: hartrao/hp3330/1
#
hartrao/hp3330/1/gpibdev: dev04
hartrao/hp3330/1/frequency_set: 1.0
```

Of course other variations abound depending on the number and types of devices and what extra information one would like to encode in the device name.

4.3 Updating the database

To update the database use the utility `dbm_update file` where `file` is the name of your resource file w.r.t. to the `$DBM_RESDIR`. For example `dbm_update hartrao/hp3330.res`

4.4 Examining the database

To look at the contents of the database use `dbm_read domain` where domain is HARTRAO for example or `dbm_read all` to read the entire contents of the database.

4.5 Miscellaneous utilities

The following utilities exist which could be of use to the device server programmer :

1. `dbm_devdel device` - delete a device and its resources from the database,
2. `dbm_devinfo device` - check and report if a device is exported and print the name of the server if it is,
3. `dbm_fillup 0|1` - create the database and fill it from the contents of the resource files or a backup copy of the database (not supported at HartRAO yet) make sure you delete the database files in $(DBM_DIR)/* before running this command,
4. `dbm_servinfo server name` - give information on all devices served by this server, the name is specified by the "executable name"/"personal name".

5 Starting and Stopping the NCCS

The control system can be started and stopped with the help of scripts. The scripts are organised hierarchically. The idea being to have a single starting or stopping point for the control sytem. These points then call other points and so on until in the end the job is done.

To start the control system use `nccs.startup`. You will be prompted whether you want to start level 1 , 2 or 3. At HartRAO only level 1 (Manager + database + message server) and level 3 (device servers) exist. For each level `nccs.startup` will then call an appropriate script to start that level i.e. `manager.startup` and `dserver.startup`. The control system must be started in ascending level order viz. 1 then 3.

Stopping the control system should be done in the reverse order viz. 3 then 1. The shutdown scripts make use of an awk script `off` for killing a process by name as opposed to by pid.

All the scripts for starting and stopping the control system can be found in `nccs: dserver/bin`.

6 Where to find it ?

The following directories contain important files :

- `/usr/local/src/dserver` - top level directory of all device server and database and related source code,
- `/usr/local/src/dserver/dserver` - the pure device server code viz. root class, api and system libraries,
- `/usr/local/src/dserver/dserver/doc` - top level directory for device server documentation (contains subdirectories for notes, manual pages and manuals),
- `/usr/local/src/dserver/dbase` - top level directory for database source code,
- `/usr/local/src/dserver/dbase/rtdb/set_up/clnt` - source code for database client api,
- `/usr/local/src/dserver/dbase/ndbm/svc` - source code for ndbm database server (`dbm_server`),
- `/usr/local/src/dserver/dbase/ndbm/tools` - source code for ndbm database utilities,
- `/usr/local/src/dserver/classgen` - source code and template files for class generator,
- `/usr/local/src/dserver/dataport` - source code for dataport,
- `nccs:/home/dserver/include` - released include files,
- `nccs:/home/dserver/lib` - released libraries,
- `/usr/local/src/nccs` - top level directory for all course code specifically developed for and around the nccs,
- `/usr/local/src/nccs/bench` - source code and results for nccs benchmarks,
- `/usr/local/src/nccs/doc` - nccs documentation,
- `/usr/local/src/nccs/drivers` - nccs device drivers,

- `/usr/local/src/nccs/dservers` - nccs device server classes,
- `/usr/local/src/nccs/tcl/obsh` - simple beginnings of an observer's shell based on TCL, `/usr/local/src/nccs/steer` - source code for DOS implementation of PC-STEER
- `/usr/local/src/nccs/userif` - example of a user interface using Tk/TCL to build a prototype interface for DSCAN.