From: mcalabre@macabre.CV.NRAO.EDU (Mark Calabretta)
To: aips2-cv@macabre.CV.NRAO.EDU
Subject: Directories & RCS
Date: Tue, 25 Feb 92 12:16:51 EST

The aips++ directories have been setup according to part 1 of the system management discussion document. I've created the whole directory tree, including directories that we won't be using for a lonnggg time as an indication of what the complete structure will look like (I placed the aips++ root directory under /aips2 to avoid confusion with existing subdirectories). The directories of immediate interest are

```
/aips2/aips++/code/kernel/include      ...class header files
/aips2/aips++/code/kernel/implement    ...class implementation
/aips2/aips++/code/kernel              ...applications
```

We will be using RCS in the short term for code management. It is very easy to use. In the first instance create your class header file, say Foo.h, and move it to the include directory

```
cp Foo.h /aips2/aips++/code/kernel/include
cd /aips2/aips++/code/kernel/include
ci Foo.h
```

That's all that's required to create the RCS file, Foo.h will be deleted and Foo.h,v will appear in its place. To checkout the file for further revision do the following

```
cd /aips2/aips++/code/kernel/include
co -l Foo.h
```

don't forget the "-l" option which locks the file for your exclusive use. You should then move the file to your home directory to work on it, although it will probably be ok to work directly in the aips++ directories for now. You can get RCS to add an identification line to your source code if you include a line of the form

```
// $Id: 108.text,v 1.1 1992/04/09 19:53:14 bglenden Exp $
```

usually somewhere near the top. When it comes time to begin compiling we'll throw together a few makefiles. If you have any problems with RCS let me know.

Mark

Subject: RCS
Date: Wed, 26 Feb 92 12:31:17 EST

ChrisG has pointed out a nice little scheme for making it easier to use RCS from your work area. Here are the basics

```
ipx% cd ~/aips++
ipx% ln -s /aips2/aips++/code/kernel/include RCS
```

which is required once only. Then you can work directly from your own area via

```
ipx% cd ~/aips++
ipx% co -l header.h
ipx% ci header.h
  :
  :
```

This particular example only works with one aips++ directory. To allow for more you can duplicate the aips++ directory tree under your ~/aips++ in the obvious way.

Mark

Subject: RCS symbolic links
Date: Fri, 6 Mar 92 18:29:28 EST

In attempting to construct a general purpose makefile for the kernel classes

it has become obvious that we do need to create proper RCS subdirectories in the ~aips++ tree. This is also apparent from the present undesirable situation where everyone has to have separate plain-text copies of all include files and implementation files that they are interested in.

You will need to change your existing symbolic links from

```
~bloggs/aips++/code/kernel/include/RCS ->
                    /aips2/aips++/code/kernel/include
```

to

```
~bloggs/aips++/code/kernel/include/RCS ->
                    /aips2/aips++/code/kernel/include/RCS
```

etc. For the time being I've created a bit of legerdemain which will allow both forms to work simultaneously. However, this is only an interim measure which will last only till everyone changes over (i.e. not very long I hope).

Sorry about the inconvenience, but having a generic makefile should make life easier in the long run. Please mail me when you've made the change.

Mark

Subject: Personal aips++ workspace
Date: Mon, 9 Mar 92 09:28:59 EST

I've noticed that a number of people are not using RCS symbolic links as described in previous mail. There are increasingly good reasons for doing so, not the least that it encourages people to check their code back in, and that a generic makefile is being written which will assume this structure.

To make it really easy we have a script called "mktree" (courtesy of Peter) - here's what you do

```
mkdir ~/aips++/code
cd ~/aips++/code
mktree
```

mktree will ask a couple of questions which have sensible defaults. It will construct a directory tree which shadows the /aips2/aips++/code directory tree. It will not overwrite any existing directories.

You should then move your prototype header files to

```
~/aips++/code/kernel/include
```

and your prototype implementation files to

```
~/aips++/code/kernel/implement
```

In each of these directories you will find a symbolic link pointing to the ~aips++/code/kernel/*/RCS repositories.

For your information, the following ~aips++ directories are currently in use

```
~aips++/code/kernel/include
~aips++/code/kernel/include/RCS
~aips++/code/kernel/implement
~aips++/code/kernel/implement/RCS
~aips++/code/install
~aips++/code/install/RCS
~aips++/code/install/lwfa
~aips++/code/install/lwfa/RCS
~aips++/sun4/apex/bin
~aips++/sun4/apex
~aips++/sun4/apex/cville
~aips++/sun4/apex/doc/man1
~aips++/sun4/apex/doc/cat1
~aips++
```

Mark

Subject: aipsinit, aipsrc, etc.
Date: Fri, 6 Mar 92 11:01:22 EST

I have implemented the rudiments of a system for defining and using an aips++ user and programmer "environment". You can get this by putting the following lines in your .login file:

```
set a_vers = apex
source /aips2/aips++/aipsinit.csh
```

and/or in your .profile file

```
a_vers=apex
. /aips2/aips++/aipsinit.sh
```

(do not export a_vers). aipsinit currently does three things:

1) Defines an environment variable called AIPSPATH which is composed of a sequence of five blank-separated strings which define

   a) the root of the aips++ directory tree
   b) the machine architecture
   c) the aips++ version (base or apex)
   d) the local site name
   e) the local machine name

   Host information comes from the /aips2/aips++/aipshosts activation database (q.v.).

2) Appends the aips++ bin directories to $PATH. If a_vers is set to "apex" .../apex/bin and .../base/bin are appended, otherwise just .../base/bin.

3) For SUN4 and SUN3 architectures only, appends the aips++ man directories to $MANPATH. If a_vers is set to "apex" .../apex/doc and .../base/doc are appended, otherwise just .../base/doc.

In addition, I have implemented an "aipsrc" resource database mechanism. Check the file /aips2/aips++/sun4/apex/aipsrc to get an idea of how it works. A sequence of aipsrc files are scanned in the following order:

```
~/.aipsrc
$root/$arch/$vers/$site/$host/aipsrc
$root/$arch/$vers/$site/aipsrc
$root/$arch/$vers/aipsrc
```

The resource database is implemented via the "getrc" utility written in C (no man page yet).

The first use to which the resource database mechanism has been put is the implementation of a printing system. The "pra" utility may be found useful for printing aips++ class implementations in a compact form. The "pri" utility prints ASCII or PostScript files and may also be useful. Related utilities are "prq" (list print queue), "prm" (remove entry from print queue), and "prd" (list default aipsrc print queue). Each of these has a man page.

The code for all of this stuff is in an RCS repository, /aips2/aips++/code/install.

Please report any problems, preferably via email.

Mark

Subject: Include file mechanism
Date: Wed, 11 Mar 92 15:32:21 EST

I've noticed that lot of class implementation files have several #include statements at the top.

I propose that the implementation files include ONLY THEIR OWN CLASS HEADER FILE, all other required includes being moved into the class header itself. That way, including a class header file in an application guarantees that all required header files will be included.

This will become important as soon as we start writing serious applications.

The TESTBED program appended to a class implementation file may need to include extra header files, but certainly should not duplicate those in the class header file.

Mark

Subject: include file logic
Date: Wed, 11 Mar 92 17:07:09 EST

Well, my previous proposal for this quickly fell flat on its face! There are subtle logical problems with including all header files required by a class in the class header file. It may also be undesirable from the information-hiding point-of-view. Therefore, the revised proposal is -

*) Class header files should include only header files that they actually need.

*) Class implementation files should not duplicate header files included in the class header file.

*) The TESTBED program should include any additional header files it needs.

Mark

Subject: kernel makefile
Date: Thu, 12 Mar 92 15:23:55 EST

A turbo-charged version of the kernel makefile is now available. As well as being much faster to startup it has the following new features:

*) Update the kernel library for a specific module -

```
gmake 'lib(PixelCoord.o)'
```

The quotes are necessary to stop the shell from interpreting the (). Only checked-in header and implementation files will be used.

After updating a single module it is necessary to ranlib the library explicitly via

```
gmake ranlib
```

Old features which some may not have heard of

*) Build a TESTBED executable and .i and .o files -

```
gmake PixelCoordTEST
```

*) Build a .i file (e.g. to load into ObjectCenter) -

```
gmake PixelCoord.i
```

*) Build a .o file -

```
gmake PixelCoord.o
```

(also makes a .i file as an intermediary).

*) Make the kernel library from checked-in header and implementation files -

```
gmake lib
```

(Does not yet protect against clashes.)

Except when updating the library, the makefile will look first for .C files in the current directory, then in the standard implement directory. Likewise, it looks for include files in ../include, then the standard include area. It is not necessary to checkout a copy of the class files you want to make.

Mark

Subject: Plain-text copies of the RCS files
Date: Mon, 16 Mar 92 10:11:07 EST

Checking a file back into RCS is not enough to update the plain-text copy of the file in the corresponding ~aips++ area. This will be done automatically whenever you preprocess or compile the file, but can also be done explicitly by using

```
gmake chkout
```

Mark

Subject: aips++ makefiles
Date: Tue, 31 Mar 92 20:04:51 EST

All aips++ makefiles now have a "makefile" target. This will ensure that out-of-date makefiles are automatically updated when you run gmake, thereby fixing the problem of stale makefiles which plagued us.

In order to get the new mind-reading makefiles you will need to delete any copy of an old makefile from your work area (or checkout the newone on top of it). After that it's automatic!

Mark

Subject: RCS directories
Date: Tue, 7 Apr 92 18:02:34 EDT

The RCS source repositories have been reorganized somewhat, basically to support future implementation of CVS, but also with useful repercussions for the remote source code distribution. The change is to have plaintext source code under ~aips++/code, with the RCS repositories under ~aips++/master.

For the time being we will continue to use RCS as before, but the RCS symbolic links will point to a different place.

To implement the new scheme do the following

```
find ~/aips++/code -name RCS -print -exec rm {} \;
cd ~/aips++
mktree
```

or similar, depending on how you set up your private work area.

Mark