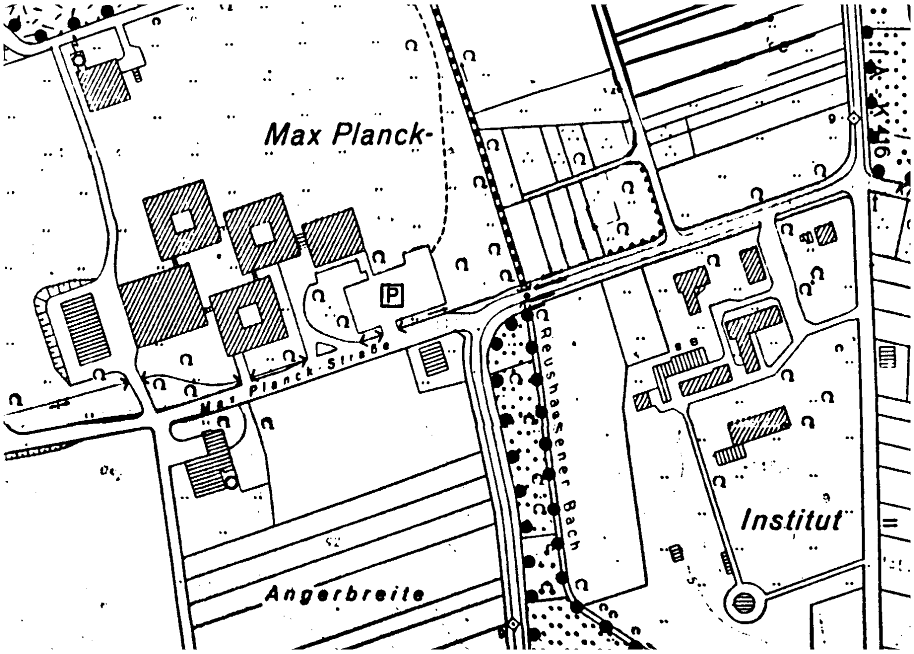
Los Alamos
National Laboratory

PAPCO USER'S GUIDE

Version 9.0



Reiner Friedel
LANL, NIS-2, MS 426
Los Alamos, NM 87544
United States of America
Tel.: +1 505 665 1936
Fax.: +1 505 665 4414
email: friedel@lanl.gov

Los Alamos, February 3, 1999

Contents

List of Figures

List of Tables

Chapter 1

Overview

PaPCo (PAnel Plot COmposer) is a widget-driven plotting package based on IDL, allowing the user to interactively custom-make a plotting product of several panels on the same time base from a variety of data sources.

1.1 Rationale

PaPCo grew out of the CRRES data analysis performed at MPAe, which requires the use of several data sets for common analysis. Indeed, one of the scientific aims of CRRES was to investigate the particle budget in the inner magnetosphere with an extremely wide energy range; from plasma eV right up to highly relativistic particles in the GeV range, using a suite of particle instruments.

However, PaPCo was designed with the needs of a common tool for the computer layman in mind, who wants to access large and varied data sets. PaPCo is thus suited for use in the ISTP community, where the prime objective is the simultaneous analysis of multi-spacecraft / ground-based multi-instrument data.

PaPCo intends to provide some basic graphics tools for a wide variety of data. It is NOT intended to be a substitute for a detailed analysis system for an individual data product from a specific instrument, such as may be used for extremely high time resolution detector by detector analysis on a particle spectrometer. Such tools are also used for instrument debugging. However, if such a tool is an independent stand-alone IDL widget application, it can be easily interfaced into PaPCo.

The most important design feature of PaPCo is expandability. PaPCo supplies a basic framework in such a way that existing data extraction and plotting routines can be used with minimal modification. A great deal of effort was invested to make extension of PaPCo to new data sets as easy and simple as possible. It is, however, the responsibility of the user to write his own modules for PaPCo. In other words: We will maintain the car. You decide who to put in it and where to go.

1.2 Can one get help?

The PaPCo package provides ample examples of a variety of data products which have been added as modules, and provides templates for all the tasks required to add a new module. In contrast to most scientific software, care was taken to extensively comment all the code, and this manual is not intended as a substitute for reading those comments and the documentation in each procedure header. PaPCo itself provides an on-line help button, which is also standard for all the panel editor windows of the modules. However, it is the responsibility of the module author to provide the help-files needed for his/her module.

LANL's responsibility in the maintenance of PaPCo is linked to whoever is currently maintaining the code and is not guaranteed. Currently, that person is Reiner Friedel. In particular, this responsibility is limited to the main core of PaPCo (excluding modules). The modules are maintained by their respective authors (refer to the List of Current Public PAPCO modules[1]), and you need to contact them for help if needed.

1.3 Distribution

1.3.1 PAPCO Web page

The PAPCO package can be obtained through the PAPCO Software Home Page which is maintained on the CCR (CAMMICE/CEPPAD/Rapid) team home page. This includes an on-line Web manual equivalent to this printed copy. You can reach these sites as follows:

- PAPCO Software Home Page LANL[2]
- PAPCO Software Home Page RAL[3]

This site does not contain any of the raw data files needed for the various PAPCO modules. It's up to the user to obtain the data he/she needs. The latest modules use the "PAPCO get remote data" facility, which automatically gets the files needed over the net.

The main PAPCO functionality can be used with the standard `plot_type` module which works with a sample data file for Jan 11, 1997. This is a fully functional module and can be used for testing.

1.3.2 PAPCO servers

Currently under construction are PAPCO servers at LANL and UIOWA. Initially these will allow you to select from several pre-defined products and to submit data requests for user-defined time ranges via the Web. The GIF or Postscript file of the plot can then be viewed or down-loaded.

As a second step a Web Panel Composer that mimics some of the PAPCO features will become available, so that the user can "Compose" his/her own plot on the Web.

1.3.3 PAPCO mailing list

A mailing list for PAPCO users is maintained at LANL. On this list are the currently known users of the system, which includes most people on the POLAR and CLUSTER instrument teams.

```
-------------------------------------------------------------------------------
WHO SHOULD BE ON THIS LIST:     PAPCO Module developers
                                Anyone who needs to install & maintain PAPCO
                                PAPCO Users

WHAT THE LIST WILL BE USED FOR: Updates on new PAPCO versions
                                Updates on new PAPCO modules and
                                  where to get them
                                Report on bugs and fixes

THE LIST IS PUBLIC:             Anyone can sign on or sign off. Anyone on the
                                list can post to the list.

LIST ADDRESS:                   papco@lanl.gov
-------------------------------------------------------------------------------
```

[1] http://leadbelly.lanl.gov/ccr/software/papco/papco_modules.html
[2] http://leadbelly.lanl.gov/ccr/software/papco/papco.html
[3] http://shaper.bnsc.rl.ac.uk:8080/ccr/software/papco/papco.html

Feel free to subscribe yourself to this list:

PaPCo mailing list[4]

1.4 Portability

The PaPCo core package has been written with portability in mind. It was developed and extensively tested under UNIX. **All** PaPCo functionality is only guaranteed to work fully under UNIX at this stage.

To date PaPCo core has been further run under VMS and Windows 95. Experience on these platforms is still limited and unforeseen problems might still occur. Expect all PaPCo core functions **except** the get remote data facility to work on these platforms.

Portability of PaPCo modules is up to the module authors. Older modules are only supported under UNIX, modules written from this version onwards can make use of the new portability features.

1.5 Configurability

There is only one compulsory edit of a file needed to configure PaPCo for a site: You need to set the directory of the main PaPCo package and of your own private site from which you run PaPCo. All other configuration can be done via user-friendly widgets.

PaPCo automatically configures itself to use whichever modules the user supplies and chooses. Adding and removing modules is simply done through a startup widget. Each such configuration can be saved, and old configurations can be loaded through the startup widget.

At start-up PaPCo sets environmental variables pointing at the location of PaPCo, the local `PAPCO_USERS` directory and the modules. The location of the data directories can be configured in each module's panel editor by setting the correct environmental variables.

Printers can also be set-up through a printer config file which tells PaPCo the name of the printer and the associated call needed to access it. Only PostScript (normal and encapsulated) and GIF output is supported. If no printer config is found then only output to file is possible.

Several PaPCo variables are set via a defaults file which "remember" the last state PaPCo was in when exited. Some of these defaults are user-configurable, and aim to make the widget exportable to any kind of screen size.

1.5.1 Data products

We use the term "product" to denote a user-defined combination of panels of various data types, such as the one shown in Figure 4.1. The product is the description of the panels, i.e. how many, what they contain, in which order they appear, what sizes they should be. Such a product could be an overview plot, a summary plot or anything else that the user wants to put together.

PaPCo allows the saving and loading of such product files. The built-in batch processing capability then uses a product for a range of time intervals. The intention is that for collaboration with co-workers, the user can use the vehicle of these product files and associated batch files to exchange data relevant for a given scientific project. Since a lot of the data nowadays is made available in the form of plots on the Web, PaPCo can be used as a front end to automatically make any plot desired.

1.6 Interfaceability restrictions

This section applies to the user-supplied routines as part of a PaPCo module.

[4]"mailto:papco@lanl.gov"

1.6.1 Data extraction routines

The only restriction here is that data must eventually end up in a form that can be accessed by an IDL plotting routine for that data, either in an IDL COMMON block (preferred) or passed by argument. IDL offers several options for this:

1. Write the data extraction routine in IDL. IDL offers a wide variety of supported formats including fortran binaries. Fast for files written as IDL binaries only, slow for ASCII or other formats.
2. If there are existing and running programs in C or Fortran which extract the data then one can write an interface program in IDL which performs a `call_external`; this is the IDL method of calling subroutines from external relocatable object files. Note: the `call_external` procedure will be dependent on the platform specific installation of Fortran and/or C, and in general is not easily portable even between identical workstations.
3. The preferred method here is to access the data in whichever format it is written using IDL even though it's slow, but then to make a carbon-copy IDL binary for subsequent use. This binary file should be an XDR file, which is platform independent and can be read by IDL running on any platform under any operating system. Saving of the file can be done with the convenient IDL save/restore feature, or as IDL binaries with the /XDR keyword. In this way data reading is only slow once: the IDL binaries restore quickly.

The user can decide for himself whichever method involves the least amount of work. To make the read part of the module more portable it is advised to have the location (directory) of the data set via an environmental variable.

Current read-routines use an error-checking method which allows PAPCO to "crash softly" if it can't find data or there was a problem in reading it. This is done via an error-common block. It is up to the user to decide the extent of error checking he wants to put into his read-routine. At least the common block `get_err_no` variable has to be set to zero if there is no error checking done.

From PAPCO 09 onwards, PAPCO attempts to trap any errors that occur in any of the user-supplied routines. Instead of a plot an error message is then displayed. As this does not allow tracing of errors outside thePAPCO core this feature is one of the configurable defaults - turning error trapping off will then allow your routine to crash and you can perform error tracing.

Since PAPCO works by defining a time range to plot, it makes sense to write a wrapper routine for the data read which will utilize this time information to automatically read in data for that time and concatenate data together over periods longer than one file. It is up to the user to write a "clever" reading routine for his specific data. An example of such a reading routine can be found in the Los Alamos module and is described in Section 5.6.

NOTE:

PAPCO prescribes nothing about data format, variables, timing resolution or data product that you choose: Since it's your plotting routine that will plot the data, PAPCO doesn't care!

1.6.2 Plotting routines

The only restrictions for the plotting routines is that they are in IDL and have access to the data from the read-routine. Current modules work via COMMON blocks, but this is not a requirement. The idea was to be able to "remember" which data had already been read, which allows relative quick zooming across a number of large data sets once they are in memory. Further, the routine must be able to plot a given time range which is supplied by PAPCO. PAPCO originally worked internally in seconds since 1990 (a relict of CRRES processing), but now uses an MJDT format (modified Julian Date plus seconds since the start of that day), so that there are no time restrictions anymore. PAPCO provides many conversion utilities to change both MJDT and the old T90 to more common formats.

There are very few essential lines that have to be included into your plotting routine to make it work with PAPCO. Any plotting routine that produces a plot in a coordinate system can be used and will be scaled by PAPCO to fit into the panel chosen by the user. PAPCO prescribes nothing about the appearance of your plot, number of tick marks and labeling, etc.

However, to be compatible with all other modules and to produce plots with some uniformity, all existing plotting routines follow a common plotting philosophy (see Section 5.3), and for which a template has been written. In practice, these conventions apply only to the "dressing" around a plot and are implemented easily and quickly. This is described in more detail in Section 5.3.

1.6.3 Using Color

Since IDL provides one color table at a time, some effort has been undertaken to provide the user with more than one color table for SIMULTANEOUS use. PAPCO subdivides the normal color table into 4 sub-tables plus some fixed primary colors for use in color line plots. User routines are provided to manage these color tables.

This means that any color plot routine that the user wants to use should work with this color table scheme - otherwise the colors for all the other modules might be messed up. Again, this is not a fixed requirement - however, to work well with the other modules adherence to the PAPCO color scheme is strongly advised. More details are in Section 5.4.

PAPCO supplies a color plot routine which is compliant with the PAPCO color scheme. This is a fairly advanced and sophisticated routine which allows the user to to plot xyz plots where z is represented by a color scale, and both the x and y bin sizes can be variable. This routine is described in more detail in Section C.1.

1.6.4 Using the mouse functions

PAPCO provides some built-in functionality via the mouse, called "slices". Here again PAPCO provides the functionality: it is up to the user to make use of it; the user "tells" PAPCO in his user-written module which of the slice functions he wants to support.

The idea here is to be able to launch some function or application by using the mouse - by moving the mouse pointer over a drawn panel and clicking in it. The are several PAPCO slice functions built into PAPCO - some of which should always work once enabled, such as the zoom and hair-cursor function, and others which require a user-written application, for which there are templates provided.

A typical slice might be a separate spectrum line plot for a single time when the user points his mouse into a spectrum color plot; another one calls up the level zero analysis tool for that product at that time. Adding slice-support to your module is considered "advanced PAPCO " and should only be done once the user has a good feel for the package.

There is also a mechanism for adding a completely new slice type to PAPCO, so called "user slices". The only limitation here is that PAPCO can currently support a maximum of 16 slices (PAPCO plus user slices). You can configure PAPCO with any combination of PAPCO plus user slices. This is definitely advanced PAPCO, but offers great power to the user to add in any functionality that he/she wants. Here you can not only define the mouse action you want, but also what action is taken in the draw window. For that you can choose from a range of PAPCO supplied draw window actions, such as "draw vertical line" or "draw haircursor" or "drag box", or you can write your own draw window action! Adding your own user-written draw window action is considered "*definitely* advanced PAPCO".

1.7 Caveats

PAPCO is not a professional package designed to be fool-proof and un-crashable. It is designed for the intelligent user who has some idea of what it can do and what not. It is very likely that the user can come up with many things that PAPCO cannot do, and since we haven't thought of them, there is probably no check against you trying, and PAPCO will crash. Error trapping in PAPCO is only performed at the points where user-written functions are called, if PAPCO crashes elsewhere then there is a problem with PAPCO core.

The PAPCO main core is very stable by now with most of the bugs out. But as new modules are written they are also a source of new errors: they, however, are the problem of the module author. Most new problems crop up in the modules and the individual plotting routines, and most of those are due to a violation of the PAPCO naming convention.

If you make changes and add to PAPCO, PLEASE do this in your modules. DON'T touch the main core of PAPCO: this would make any maintenance by us at LANL very difficult. If you absolutely need a change that affects the core of PAPCO, contact us here and we will try to add the change and include it a the next version of PAPCO. In fact, this is how the package has progressed, thanks to the many useful suggestions by users.

PAPCO has been in use and running since May 1995 and as problems are encountered they are fixed. So as we go along it will get better and better, and more functionality will probably be added in subsequent releases.

1.8 PAPCO sites

PAPCO has been installed at various sites and there has been quite some commitment by the ISTP POLAR community to use it as it's main data display and analysis package. Discussions are underway for its use in the EQUATOR-S and CLUSTER mission. PAPCO should grow as more people program in it - and as more modules become available, the return on investment on PAPCO by far outweighs the investment in time in it!

This list of institutions is constantly growing and is maintained on the PAPCO home page[5] which is part of the CCR Web site[6]. The CCR Web page is also mirrored at RAL, United Kingdom[7].

[5] http://leadbelly.lanl.gov/ccr/software/papco/papco.html
[6] http://leadbelly.lanl.gov/ccr/
[7] http://shaper.bnsc.rl.ac.uk:8080/ccr/CCR.html

Chapter 2

Version Control

As PaPCo is not a "professional" piece of software and has been pieced together from the contributions of many people, version control is somewhat problematic. For a time there was no attempt at all to keep track of versions and changes, but as the package is growing this is becoming more important.

Proper version control started with PaPCo 05 and will be documented in this chapter. It is not intended to be an exhaustive list of edits, changes etc, but will document major changes to the PaPCo core only. All routines in the PaPCo core have some history in their headers, but here also no attempt is made to keep a record of all bug fixes or modifications.

Keeping this User's Manual up to date is a major effort. We will not attempt to update this manual whenever any changes is made but will adopt the following strategy:

1. The Manual will only be updated for major version changes (i.e. version 5.0 to 6.0).
2. Incremental updates (ie. V6.0 to V6.1) will only be reported via the PaPCo mailing list and only if they effect the user.
3. Small bug fixes will not be reported.
4. Important bugs or fixes requested by users will be acted upon and reported on the PaPCo Home Page under the PaPCo Patch Page[1] There will be a downloadable "patch" for the user with instructions of how to install it. In this way we avoid having to release a new version of PaPCo whenever there is a fix.
5. All the patches will be implemented in each new release of PaPCo, and the PaPCo Patch Page will be cleared.

2.1 Old versions

No attempt is made to keep older versions of PaPCo supported. The user is expected to obtain the latest version: reports of bugs and problems will only be attended to in the context of the latest PaPCo version.

The CCR home page from which PaPCo can be obtained (see Chapter 3) will only keep the latest version. There is no archive of older versions of PaPCo core. There is a limited archive of older modules only.

If you have PaPCo products written with Versions older than 5.0 you might not be able to read them anymore in anything up to PaPCo 08. You will need to redefine the product under the latest version. From PaPCo 09 onwards, this restriction has been removed, and ALL PaPCo products should be readable under PaPCo 09.

[1] http://leadbelly.lanl.gov/ccr/software/papco/papco_patch/

2.2 PaPCo core v. PaPCo modules

Any version control will only be implemented for PaPCo core - the modules remain the responsibility of the module author. As PaPCo has grown, it has provided more and more functionality, which not all the existing modules support, as they were written earlier. That does not mean that older modules do not work at all, only that they do not make full use of the functionality that PaPCo can support.

Initially all modules had been written by the author of this document, but as PaPCo gets used more widely more and more modules are being written by other authors. It is up to the provider of a module to support it. We will attempt to include as many modules as possible in the distribution, and to make sure that they are functional. But it is up to the module provider[2] to keep us up to date on his/her module development.

We will endeavor to make any changes in the PaPCo core transparent to older modules: they should still work, albeit without some of the newer functionality. However, at times this may not be possible, and small edits may be needed to make your existing module work with the new version of PaPCo. We will try to keep the user informed of such occurrences - ons such case for PaPCo 09 is documented here (2.7.3).

2.3 Keeping things up to date

If you have older modules which you might want to update to the latest PaPCo standards, make use of the standard module `plot_type`. This is a sample working module which is always kept updated (see Section 7.3) and which always implements the latest PaPCo functionality.

2.4 Changes to Version 6.0

1. A completely newly structured startup procedure which is aimed making installation and customizing PaPCo simple and easy, with a minimum user-editing of files:
   (a) A new startup widget allowing easy selection of standard or user-defined modules.
   (b) Choice between new compilation and restore of compiled procedures and functions, increasing startup-speed.
   (c) New panel-editor functionality "configure", allowing easy user-configuration of module data paths and environmental variables.
   (d) The PaPCo distribution has been extended by a further tar file, `PAPCO_USER_XX`. This file is meant to be unpacked in the user's home directory and PaPCo should be run from the directory `PAPCO_USER_XX` that is created. All the directory structure expected by PaPCo is contained here to provide a standard sample setup.
2. Several changes to the PaPCo main widget:
   (a) "Previous" and "Next" buttons: Allowing the user to quickly step forward and backwards in time by the difference between the current start and end times.
   (b) New mouse action "Mouse Writes Paneldata". Allows the user to write default (time and y-coordinate) or user-defined information about the cursor position in a panel to a file (see Section 6.5.5). The times written to these files are then compatible to the Event times module (Section 7.1).
   (c) "Save&Done" button. Exits PaPCo and also saves all currently compiled procedures and functions for a quick startup.
3. Making it easier to create own modules:

[2] http://leadbelly.lanl.gov/ccr/software/papco/papco_modules.html

(a) Overhauled standard module `plot_type`. This module is now a completely working example module using a dummy data file for 11. January 1997. It implements all the latest PAPCO standards and supports all PAPCO features. This should now always be used as the model for designing own modules.

(b) A new procedure, `make_new_module.pro`, is provided. This routine creates a new module directory structure will all the files needed, using the `plot_type` module template. All occurrences of "`plot_type`" in the directory names, file names and inside all files are replaced by the user defined new module name. In this way all the major editing tasks are performed automatically, while making sure that the PAPCO naming convention is observed.

4. A new procedure `papco_get_remote_data.pro`. Makes uses of the UNIX GNU free-ware "wget" to automatically fetch data from a remote site given the ftp address, directory and filename. The user can use this in the user-written read routine to automatically obtain data from a remote site if it is not available locally. It is up to the user to implement.

5. No sample data. With older distributions, some sample CRRES data was supplied for testing purposes. Now PAPCO is distributed with a working sample module (`plot_type`) and a sample data file for this module for Jan 11, 1997. This file and module can now be used for testing. Some sample products are included in the distribution.

6. A PAPCO mailing list is now being maintained for updates and news. The list is public, see Section 1.3.3

7. The "Choose time by orbit" feature has been generalized for easy inclusion of new missions (currently supported for CRRES and POLAR).

2.5 Changes to Version 7.0

1. PAPCO core has been made portable to VMS and Windows 95. This included:

   (a) A total change to lower case filenames only, as this minimizes problems when going to VMS.

   (b) All routines that deal with file I/O, filenames, environmental variables and such like have been replaced by "papco_" routines, making the operating system dependence transparent to the user. All these routines are collected in a new file `papco_opsys.pro` under `papco_XX\papco`. These routines are documented there and the user should make use of them in making his/her own module portable.

2. A new papco slice function has been added: Hair cursor. This plots a large haircursor in the draw window and outputs x, y and z coordinate values in the draw window.

3. A new routine has been added to papco.color.pro in PAPCO core under `papco_XX\papco`. This routine allows access to the color table index used to represent a data value in a given PAPCO color subtable:

   `function PAPCO_Get_subtable_index`

   This is used by the new Hair cursor slice function.

4. The whole slice functionality has been overhauled to follow the PAPCO modular philosophy. There are now five PAPCO slices plus the easy facility for the user to add his/her own slice type. See Section 6.5 for more details.

5. A template now exists for the "Choose time by orbit" feature.

6. A new routine has been added under `papco_XX\papco_lib` to deallocate logical units from `get_lun` which are closed but still allocated.

   `procedure dealloc_lun`

   This is needed to deallocate logical units which have been allocated by the restore function, since this is not done by IDL internally.

7. `papco_get_remote_data.pro` under under `papco_XX\papco` has been debugged and streamlined. The standard PAPCO error checking has been added, to conform to other read routines.

8. The `plot_type` sample module read function has been rewritten to be conform to the latest PAPCO philosophy and standard. It now used the routines in `papco_get_remote_data` to obtain the sample data file from anonymous ftp on

   `hydra.mpae.gwdg.de/pub/friedel/plot_type_data`.

   The sample data file is no longer part of the distribution.

9. `right_side_label.pro` under `papco_XX\papco_lib` has been rewritten to achieve nicer line breaks in the labels using the "!C" format string. The change is transparent to the user.

10. A new sub-widget has been added to th PAPCO supplied widgets for Panel Editors: An Auxiliary file group. This is a standard sub-widget now used widely in PAPCO.

2.6 Changes to Version 8.0

Release date: 05 Feb 1998

Version 7.0 was an intermediate version to be used at the PAPCO workshop held at Los Alamos Jan 13–14 1998, and never officially released.

Version 8.0 takes account of a lot of the suggestions made at the workshop and further fixes many smaller problems. A lot of headaches were caused by trying to make PAPCO run under IDL 5.x, as some of the widget management had changed.

1. Establishment of PAPCO Rules of the road, guidelines for PAPCO participation, and guidelines for PAPCO module Authors. The PAPCO home page[3] has been extensively updated, please make sure you visit it.

2. PAPCO core has been made to run under **both** IDL Versions 4.x and 5.x. This included:

   (a) Inclusion of a new set of routines in PAPCO core under `papco_XX\papco`: `papco_idlver.pro`. Analogous to the way system dependencies were handled in `papco_opsys pro`, here calls to IDL widget routines are replaced by PAPCO calls.

   (b) Parts of the PAPCO main widget have been redesigned to make sure they behave the same under both IDL versions (see below).

3. The automated PAPCO startup procedure has been further refined and made portable to VMS and WINDOWS95. Some limited testing has now been done to check the basic operation of PAPCO core and PAPCO utility modules (plot_type, text and event) under VMS and WINDOWS95.

4. Redesign of PAPCO main widget:

   (a) A "top" part for module buttons has been added.

   (b) A "bottom" part for action buttons has been added with extra buttons for future development.

   (c) The "middle" part has become smaller, but the panel editor list is now scrollable.

   (d) the "Print" button functionality has been split up into a "Print Action" button that sets the behavior of the "Print" action button. Output labels have been added that report the currently set printers and printer file names.

   (e) The zoom time windows are now editable and allow the user to set exact zoom times. Checking has been added to always assure that zoom start time is less than zoom end time.

   (f) Mouse action information widget has been changed from text to label as it is not editable.

[3] http://leadbelly.lanl.gov/ccr/software/papco/papco.html

5. Distribution changes:

   (a) Apart from providing the distribution as variously compressed tar files, a straight zipped version is also supplied. Using the unzip utility both on VMS and WINDOWS95 allows unpacking of the distribution into the correct directory structure on those platforms.

   (b) The html version of this manual is also available from the PAPCO download site in various zipped and compressed tar formats.

6. Change to Plot Style Group for panel editors. The default color chosen here used to be black - which causes problems if you've chosen your background color as black. There is now one more color in addition to the 16 predefined colors available, which is the new default: The foreground color. This foreground color is set in the papco defaults widget.

2.7 Changes to Version 9.0

Release date: 25 January 1999

Version 8.0 was probably the first version that saw a more widespread use - and thus a lot of reports came back on bugs and problems - all of them happily minor. A lot was learned by PAPCO installations under diverse systems, such as SGI/IRIX, OPEN/VMS v7.1 and v6.2.

Incorporating all these changes warrants this release of the next Version of PAPCO. Thanks for all the reports on bugs and help in getting PAPCO to be truly operating system transparent.

A lot of new functionality has also been added to make life easier. For instance, on re-drawing a product only those panels get redrawn that have been edited. This really speeds things up when you are designing a product! Many of the user suggestions have also been incorporated.

N.B.: PAPCO 09 is the first version fully developed under IDL 5.x. While everything should still work under IDL 4.x, backward compatibility is not guaranteed!

2.7.1 New features on PAPCO Home Page

The PAPCO home page[4] has been substantially updated. Apart from distributing the latest version of PAPCO, it now also has a "patch" page for fixes and reports of bugs, it maintains a list of where to get modules, you can access the PAPCO mailing list from there, and more. It pays to check out the site periodically as it is expanding, and esp. the "patch" page is constantly updated.

Check out the "Rules of the Road" if you're a new PAPCO user. A full list of available modules, their maintainers and sites running PAPCO is also available.

2.7.2 PAPCO core changes & new features & bug fixes

All these changes are documented in more detail where needed elsewhere in this manual.

1. Implemented bug fixes from the PAPCO patch page:

   (a) Renaming of papco_default and papco_printer_config to have extension ".dat". This is necessary for portability to VMS as each VMS system treats files without extension differently. If you're updating an older version of PAPCO and don't want to lose your configuration, just rename your old papco_default and papco_printer_config files (they could have been hidden files under UNIX).

   (b) Naming Conflict: with the POLAR HYDRA module's color plot routines has been fixed. The PAPCO supplied color plot routine is now called `papco_plot_colorspec.pro`.

[4] http://leadbelly.lanl.gov/ccr/software/papco/papco.html

(c) PAPCO Configure change: It was noticed that on startup PAPCO reported module directories under UNIX ending with // instead of /. This did not create problems but is now fixed. By request (Jeremy Faden) the order of the module buttons has been changed - your user defined modules now come up first, followed by the "standard" modules. This give the user some control over the order of the buttons in PAPCO.

(d) Misspelling in built-in panel editor panel: The y-scale and z-scale widgets had "logarithmic" misspelled. This is now fixed.

(e) Problem using file dialog when printing to .gif: This has been fixed.

(f) Major `papco_cursor` improvement: For advanced PAPCO programmers only! Due to several suggestions by Doug Rowland and Jermey Faden the PAPCO core functionality of the cursor-interaction with PAPCO has been improved and further modularized. You can now add in USER_SHOW_ACTIONS - define your own action that happens in the draw window for your user slice. You can choose from PAPCO supplied options (like draw a vertical line, or a hair cursor) or write your own show action.

(g) Color fix for 24bit workstations: IDL internally runs on 256 colors, which is what the PAPCO color scheme is based on. Problems arose on installations where IDL reported more than 255 colors available. This has been fixed. This problem so far has occurred on SGI/IRIX and OPEN/VMS systems.

(h) Time hand-over fix to PAPCO batch processing: Module designers using the old PAPCO T90 standard encountered difficulties using the PAPCO batch facility as this was written to work only with the new PAPCO mjdt time standard. This has been fixed.

2. Expanded PAPCO configurability: The PAPCO startup widget now allows you to save and load the current configuration of PAPCO . There is also a new option to edit the `papco_include_addins.pro` file, and enable or disable the use of the addins.

3. PAPCO error trapping: In the past, an "unhandled" error in one of the user written interface or plot/read routines would crash PAPCO. These errors are now trapped by PAPCO at the point the user written routines get called, and an error message is printed without PAPCO crashing. Using the defaults widget this feature can be turned off if you want to trace your error.

4. Smart re-draw: new logic has been added to the draw loop to only redraw the panels of a product when needed, i.e. if they have been edited, the panel sizing/position/composition has been changed, and when a new time-range is chosen. This makes for faster redraw when designing a product.

5. Updated PRINT ACTION button behavior: The choice "Select default output" calls up the print widget directly: you no longer have to press "PRINT" afterwards.

6. PAPCO output option: As part of being funded as an ISTP utility PAPCO needs to conform to some ISTP rules - such as providing output. This feature has now been implemented in a simplified form. A new action button has been added - "Output". Once a given product has been drawn, pressing output sets a keyword which "tells" the modules to output data to file. The modules themselves need to support this option and take action. A new routine has been supplied for this purpose - `papco_output.pro` which takes a standard set of inputs and keyword to produce output files in either text, IDL saveset or CDF (still under development) format. It is, however, up to the module developer and maintainer to implement this feature in his/her module!

7. Edit batch option: A new action button, "Edit Batch" has been added. It allows the user to interactively edit and create batch files for PAPCO batch processing from within PAPCO. Files are automatically searched for in `papco_user_XX/papco_products/batch/`.

8. A new action button, "Annotate": This calls up the IDL supplied annotate utility, which allows the user to overlay text, lines and simple graphics on their plots. Very useful for making conference or publication plots (highlight some area of your plot, add comments, draw arrows etc.). The annotate overlay can be saved in a file and becomes part of your PAPCO product. A second new action button, "Rm. Annot." is supplied to delete any annotation.

9. A new action button, "Back". As you work with PaPCo , you will go through many versions of a plot. Each time you draw a new product, a copy of the previous product is now automatically saved. Hitting the back button loads the previous product, all the way back to the first one drawn. Once you exit PaPCo, all these history product files are deleted.

10. Smart crash recovery: If you for any reason need to exit PaPCo ungraciously, PaPCo will start up the next time you run it and automatically load the last saved product file from your crashed session.

11. New option in printer defaults widget: When selecting the default options for your print button, one of the options is to write the plot to file with an automatically generated file name. This file name uses a template to substitute the time range and orbit. A new symbol, "$", can now be used, which will be substituted with the current product name if there is one.

12. The PaPCo defaults widget has been redesigned and expanded to include several new options:
    (a) Choose PaPCo data output format
    (b) Enable/disable PaPCo trapping of errors
    (c) Enable/disable remote data get with wGeT

13. Changed behaviour of draw widgets "Del all" button: Hitting this not only closes all draw windows, but also re=activates the main PaPCo widget. This is useful when you are debugging a new module: You can recover from a crash in your plot routine by hitting "Del all", then the "Stop" button in the PaPCo main widget, edit your routine, recompile it, and carry on with ".con". You never have to restart PaPCo itself.

14. Updated `papco_panel_AddFile_Widgets`: This is the built-in widget you can use to select files and edit them. This has been re-written to seperate out directory path and filename, this is more robust.

15. Extended user variable space for module structure: Each module's panel editor uses a structure called `plotinfo` to keep information about it's state. We have extended the variables a user can use to the following:

```
;        --------------------------------------
         typeVector:intarr(4),  ; user tag (old, for compatibility)
         switch:lonarr(3),      ; user tag (old, for compatibility)
         options:fltarr(20),    ; user tag (float options)
         ioptions:lonarr(16),   ; user tag (long options)
         soptions:strarr(16)    ; user tag (string options)
;        --------------------------------------
```

    Also, PaPCo products are now compatible across PaPCo versions.

16. Extending PaPCo read by orbit functionality: This has been rewritten and modularized, which makes it relatively easy for the user to add in another satellite for this feature.

17. More buttons on PaPCo draw widget: The "Print", "Output" and "Back" buttons from the main widgets have been replicated here. This avoids having to toggle between the draw and main widget too often.

18. Quick access to a panel list entry's panel editor: Instead of having to double click on the panel entry in the plot composer list in the main widget, clicking inside the draw window NEXT to (left or right) the panel also calls up that panel's panel editor.

19. Updated color logic in the panel editor schematic draw widget. More colors are used to warn the user of overlaps and overplot conflicts:

    solid: panel not an overplot

outline: panel is an overplot

blue: Single panel with no overlapping or overplotting panels

cyan: Two or more panels OF THE SAME SIZE at this location (used for lining up overplots)

dark green: O.K. overplot arrangement at this location. All overlapping panels have the same size, and all but one are overplots!

red: Conflict! Not all panels are overplots, not all overlaps here are of the same size - this will possibly not create the panel plot the user intended!

20. Updated panel editor Reform and positioning actions: These actions have been redesigned with new logic to make them more intuitive, and to also handle overplots. Any panels at the same position are handled together, ie resized together, etc.

2.7.3 PaPCo core changes affecting modules

In PaPCo 09 we have fixed an insiduous problem with the way the PaPCo-supplied widgets in `papco_panel.pro` behaved. There was some "crosstalk" between these PaPCo internal widgets and user written ones, which was puzzling and which took a while to trace down.

Some elements of the user tag "switch" had also been used in the PaPCo widgets. As a result, whenver the user actually used that tag, the behaviour was unpredictable!

This has been fixed. As a result, IF A USER PANEL EDITOR DOES NOT USE THE TAG "SWITCH", THEN IT DOES NO LONGER NEED TO BE RETURNED. If you still specify "switch" as one of your return tags, your panel editor will crash on exit.

To fix this simply delete the tag from you return data structure. In many older panel editors, the return structure looks something like this:

```
returnData={typevector:widgetData.typevector, $
            typevector:widgetData.switch, $
           }
additionalRData=papco_panel_ReturnData(widgetData)

return, create_struct(ReturnData, additionalRData)
```

If you're not using the "switch" tag yourself, simply edit your panel editor file and remove the tag from `returnData`:

```
returnData={typevector:widgetData.typevector}

additionalRData=papco_panel_ReturnData(widgetData)

return, create_struct(ReturnData, additionalRData)
```

This is the ONLY edit you will need to make to your module to get it to run under PaPCo 09!

Chapter 3

Installing PaPCo

PaPCo is obtained in the form of tar files from the PAPCO home page[1] which is part of the CCR Web site[2]. The CCR Web page has a mirror at RAL, United Kingdom[3]

From the PaPCo Software Home Page click on `Download the latest PAPCO version` to access the download site. The quick installation guide is contained in the readme file on the download site:

```
The following files are on this site:

papco.XX.tar.       - tar: PAPCO core package
papco_user.XX.tar.  - tar: PAPCO user package
papco_US_XX.ps.     - Postscript: PAPCO user's manual, US paper format,
                      150 pages, double sided
papco_A4_XX.ps.     - Postscript: PAPCO user's manual, A4 paper format,
                      146 pages, double sided
readme              - ASCII text: this file, a short installation guide

XX stands for the PAPCO release number. All these files are supplied as is and
in various compressed formats (gzip, zip, compress).

The installation procedure is simple. PAPCO is designed to have a central core
(papco.XX.tar), which only needs to be installed once centrally for all users
at a site, and a papco_users directory, which is unique to each user for
his/her own configuration of PAPCO.

The least amount of editing and configuring is required if you install (untar)
papco_user.XX.tar under your home directory. Run PAPCO from your papco_user_XX
directory.

However, you can choose to have the PAPCO core code reside anywhere. PAPCO
comes with all modules supplied with the distribution residing under
papco_XX/papco_modules. Your own modules can reside anywhere, as you specify
their location on startup.

  1. Copy papco.XX.tar its destination directory.
  2. Untar it (tar -xvf papco.XX.tar). This creates a directory \papco_XX -
     the PAPCO package with all supplied modules. This directory has four
     subdirectories:
       A. \papco - PAPCO core files
       B. \papco_lib - Utility routines also useful outside of PAPCO.
       C. \papco_doc - PAPCO documentation including this
                       document as a set of LaTeX files.
       D. \papco_modules - all existing user and Standard Modules
  3. Copy papco_user.XX.tar to your home directory.
  4. Untar it (tar -xvf papco_user.XX.tar). This creates a directory
     papco_user_XX which contains the following files and directories:
       A. papco_default.dat - Sample PAPCO defaults file.
       B. papco_printer_config.dates - Sample printer configuration file.
       C. A directory \papco_products - the place for all the panel products
          that can be loaded into PAPCO or saved from PAPCO. Includes some
          pre-defined products to work with the the Standard Modules
          (papco_event, papco_text and plot_type) This directory has several
```

[1] http://leadbelly.lanl.gov/ccr/software/papco/papco.html
[2] http://leadbelly.lanl.gov/ccr/
[3] http://shaper.bnsc.rl.ac.uk:8080/ccr/CCR.html

```
         subdirectories:
           I. batch/          - for PAPCO batch files.
          II. event_times/    - for the event time files used by the
                                papco_event Standard Module.
         III. panel_text/     - for the text files used by the papco_text
                                Standard Module.
          IV. batch_products/ - for the product files that work with the
                                batch files.
           V. annotate/       - for the annotation overlay files
          VI. test/           - a scratch directory for testing products
         VII. project_1/      - for your own project product files.
         It is advised that you create your own further subdirectories here
         for the product files of whatever projects you are working with.
      D. A directory papco_modules_defaults - will contain the local
         configuration files for the modules you use.
      E. A directory papco_download_temp - if you use the automatic get
         data feature, this is the temporary place files get downloaded to
         and wGet logfiles are written to.
      F. papco_startup.pro - A sample IDL startup file for PAPCO. This is the
         master configuration batch file, which itself calls several others:
           I. papco_set_variables.pro - sets paths and environmental
                variables for the PAPCO main package.
          II. papco_include_modules.pro - sets paths and environmental
                variables for the modules that you want to include.
         III. papco_include_addins.pro - sets paths and environmental
                variables for any add-in package you might want.
          IV. papco_compile_new.pro - compiles all procedures/functions
                needed for the PAPCO system (slow) OR
           V. papco_restore.pro - restores a previously saved set of all
                procedures/functions needed for the PAPCO system (fast).
      G. papco_setup.pro - equivalent to papco_startup.pro but calls
         a PAPCO configuration widget first which allows the user to select
         modules and startup mode (compile/restore). This  program re-writes
         the papco_startup.pro and papco_include_modules.pro files before
         calling them.
      H. papco_autostartup.pro - A simple startup file that automatically
         initializes PAPCO, runs the current configuration and then exits IDL.
  5. Edit the start-up files to ``localize'' PAPCO. You will need to:
      A. edit papco_set_variables.pro to set the paths to your PAPCO
         installation directory and to your papco_user directory. Use full
         path names only here!
      B. edit papco_printer_config.dat for your site.
      C. edit papco_default.dat (not essential. It's easiest to simply start
         PAPCO without the defaults file (delete it, PAPCO will create a
         defaults file the first time you exit).
         NB.NB.NB: delete any OLD papco_default you might have from previous
         installations, as they may not be compatible with new versions!!
  6. Start PAPCO by changing to you papco_user_XX directory and type
     idl papco_startup (see Manual Section 3.6).
  7. Select the modules you want to use from the Configure Widget.
  8. Once PAPCO has compiled, startup from the IDL prompt by typing papco.
  9. Configure the main PAPCO widget using the Defaults button. You might need
     to choose a font and draw window size that makes the widgets fit on your
     screen.
 10. Each time a module is included for the first time, you need to configure
     it using the configure button in the module's panel editor.

NOTE: papco_XX and papco_user_XX will at times have different version
numbers. New PAPCO core versions do not always require the installation of a
new papco_user directory, to update to a new version of PAPCO reinstalling
papco_XX is often sufficient.

This description is from Chapter 3 of the PAPCO User's Guide "Installing
PAPCO".
```

The next sections explain the files installed under `papco_user` in more detail.

3.1 papco_default.dat

This is a very simple file containing some default settings for PAPCO widgets and some default paths for files. A sample defaults file is included in the distribution under `papco/papco_doc/papco_default.dat`.

```
; file papco_default.dat: Default setting for papco
; lines starting with ; are comments and are ignored
```

```
;
; widgetData.saveStatePath, default/last path :
/nh/toaster/u/friedel/mypapco/papco_products/test/
; widgetData.batchFilePath default/last path :
/nh/toaster/u/friedel/mypapco/papco_products/batch/
; widgetData.outputFilePath, default/last path:
/n/toaster/u/friedel/
; printerInfo.fileName_tmpl, default/last template:
$_<->.gif
; Printer menu defaults
       1       2       0       1       3
; widgetData.default_output_format, data output format
ASCII
; widgetData.land_port: default/last landscape or portrait
       0
; widgetData.mouse_doesSlice, default/last slice option
       4
; widgetData.satellite, default/last satellite :
       1
; widgetData.default_charsize, annotaion char size :
     0.850000
; widgetData.default_charthick : annotation char thickness
      1.00000
; widgetData.default_font, default X-font :

; widgetData.minTime : plot start time
{       50323      0.00000   210038400}
; widgetData.maxTime : plot end time
{       50323      86400.0   210124800}
; widgetData.us_europe : time format
       1
; widgetData.default_gif_size : file gif pixel size
     570     760
; widgetData.default_draw_size : screen draw widget size
     570     760
; widgetData.default_ps_mult : charsize factor for PS
      1.50000
; widgetData.default_draw_ident :
       1
; widgetData.default_ps_eps : encapsulated postscript = 1
       0
; widgetData.default_bckg_color : background plot color
     200
; widgetData.default_forg_color : foreground plot color
       0
; widgetData.default_TICKLEN : normal coords ticklength
   -0.0150000
; widgetData.default_XTICKS : no of x-ticks
       6
; widgetData.default_XSTYLE : x style
       1
; widgetData.default_XMINOR : minor x ticks no
      10
; widgetData.trap: trap user errors (1) or not (0)
       0
; widgetData.wget: enable remote get (1) or not (0)
       0
;
; end of defaults
```

This file should resides in the user's `papco_user` directory. You may change the defaults either by editing this file or changing the corresponding widget entries in the PAPCO defaults widget (See Figure 4.4). The current state when you exit PAPCO is always written to this file, and is loaded again the next time you start.

NB. You do not require this file to be present the first time you start PAPCO. If PAPCO cannot find the file, it uses built in defaults, and the first time you exit, it writes this defaults file. *To ensure a smooth startup of* PAPCO *the first time you start* PAPCO*, it is strongly advised to start off without a* `papco_default.dat` *file and to attempt editing it only after* PAPCO *has constructed the file on exiting. Preferably, use the defaults widget.*

Upgrading from previous Versions of PAPCO

In general, it is best to also install a new `papco_user_XX` directory for your new version. You can then copy over your product files from the old `papco_user/papco_products` subdirectory to the new one: from PAPCO 09 onwards all products are upwards compatible!

You might also want to copy over your old `papco_printer.dat`, these files always have been compatible across PAPCO versions.

DO NOT use your old `papco_default.dat` file, as the default setting capability may have been extended. It's best to start up the new version of PAPCO without this file, it will be "re-made" the first time you exit, and you can then edit the new file to reflect your old defaults.

3.2 papco_printer_config.dat

This file is used by PAPCO to know which printers are available and how to access them. The sample file included in the distribution under `papco_user_XX` contains printer calls used under a SunOS UNIX PAPCO installation at LANL.

```
;*************************************************************************
;* Printer Configuration File for PAPCO system
;*
;* Only ONE printer name and print command per line separated by a comma.
;* Lines starting with a semi-colon will be ignored.
;*
;* Format:  printername, pirnt command
;*
;* e.g. for UNIX: UNKNOWN, lp -d unknown
;*      for VMS : UNKNOWN, print/que=unknown
;*************************************************************************
Phaser 550 w108, lpr -P w108rgb
Phaser 550 w108 (transparency), lpr -P w108rgb_t
Phaser 550 n114, lpr -P n114rgb
Phaser 550 n114 (transparency), lpr -P n114rgb_t
Phaser 450 4-color (CMYK) dye-sublimation s110, lpr -P s110dye
Phaser 450 4-color (CMYK) dye-sublimation s110 (transparency), lpr -P s110dye_t
Dye Sublimation CR, lpr -P n114dye
QMS 410 (laser b&w) w108a, lpr -Pw108abw
```

On each line you need to enter a printer name, a comma, and your system call to access this printer. Use only PostScript printers.

This file needs to be in your `papco_user` directory. If PAPCO cannot find this file, it is configured to run without access to a printer, and you can only print to file.

3.3 papco_startup.pro

This is the main PAPCO startup file. *You should not need to edit this file.*

```
;******************************************************************************
;* FILE:        $PAPCO_USER/papco_startup.pro
;*
;* DESCRIPTION: initializes PAPCO
;*
;* CALLED BY:   papco_setup
;*
;* HOW TO START:
;*    from command line: type idl papco_startup   (must be in current dir)
;*        or, in your shell set up an environmental variable IDL_STARTUP
;*        which points to the location of papco_startup
;*        eg for ksh:  IDL_STARTUP=~/papco_startup
;*                     export IDL_STARTUP
;*        then simply type idl
;*    from IDL:      type @papco_startup (if papco_startup.pro in local dir)
;*                or type @PATH/papco_startup (if papco_startup.pro in PATH)
```

```
;*    then you can start PAPCO by executing the command "papco".
;*
;* MODIFICATION HISTORY:  written July 1997, R.Friedel
;******************************************************************************

; Set up some variables used in this startup file
@papco_set_variables

print, ''
print, '--- Start papco_startup ---------------------------------------------'
print, ''
print, '--- Module Paths and module data path env. var. settings -------------'
print, ''

@papco_include_modules
if !error eq -170 then $
 print,'% papco_startup: Use @papco_setup to first configure modules for PAPCO'

; compiling/Restoring PAPCO
@papco_compile

; setting up other add-inn IDL applications
;no addin batch file chosen

; force quiet mode off to see idl messages
!quiet=0

print, ''
print, '--- Environmental variables now set: ---------------------------------'
print, ''
print, 'PAPCO_QUIET:        ',getenv('PAPCO_QUIET')
print, 'PAPCO_STARTUP_MODE: ',getenv('PAPCO_STARTUP_MODE')
print, 'PAPCO_ADDINS:       ',getenv('PAPCO_ADDINS')
print, 'PAPCO_HOME:         ',getenv('PAPCO_HOME')
print, 'PAPCO_PATH:         ',getenv('PAPCO_PATH')
print, 'PAPCO_MODULES:      ',getenv('PAPCO_MODULES')
print, 'PAPCO_DOWNLOAD:     ',getenv('PAPCO_DOWNLOAD')
print, 'TIME_CONV:          ',getenv('TIME_CONV')
print, 'POLAR_ORBITS:       ',getenv('POLAR_ORBITS')
print, 'CRRES_ORBITS:       ',getenv('CRRES_ORBITS')
print, 'USER_MOUSE_ACTIONS: '
uma=getenv('USER_MOUSE_ACTIONS')
uma=str_sep(uma,'/')
for i=0,n_elements(uma)-1 do print,'                    ',uma(i)
print, 'USER_SHOW_ACTIONS:  '
usa=getenv('USER_SHOW_ACTIONS')
usa=str_sep(usa,'/')
for i=0,n_elements(usa)-1 do print,'                    ',usa(i)
print, ''
print, '--- Done papco_set_variables -----------------------------------------'
```

This file is written as an IDL batch file, which itself calls a series of batch files described below.

3.3.1 papco_set_variables.pro

This batch file is called from `papco_startup.pro` and sets up all the IDL search paths and environmental variables needed by the PAPCO core (excluding modules).

NB!! YOU WILL NEED TO EDIT THIS FILE to set the correct paths to your `papco_XX` and `papco_user_XX` directories.

```
;******************************************************************************
;* FILE:        $PAPCO_USER/papco_set_variables.pro
;*
;* DESCRIPTION: Sets env. variables and expands IDL path as needed by PAPCO
;*
;* CALLED BY:   papco_startup, papco_setup
;*
;* MODIFICATION HISTORY:  written July 1997, R.Friedel
;*                        updated for papco_09 August 1998
;******************************************************************************
print, ''
print, '====================================================================='
print, '--- W E L C O M E   T O   P A P C O ! --------------------------------'
```

```
print, '=================================================================='
print, ''
print, '--- Start papco_set_variables -------------------------------------'
print, ''

; local variables used here:
; NET            is the base network "address" . This is central the location
;                PAPCO should be installed, once, accessible by all users.
; PAPCO_BASE     is the directory where the PAPCO core code resides
; PAPCO_MODULES  is the directory in which the distribution modules reside
; PAPCO_HOME     is the user's directory from which PAPCO is run
; PAPCO_DOWNLOAD is the temporary download directory
; HOME           is the user's home directory

device,pseudo_color=8    ;necessary for Sparc ULTRA 2 or any machine with
                         ;24-bit color card

NET=            '/n/leadbelly'
PAPCO_BASE=     NET + '/belly1/home/papco/papco'
HOME=           '/n/toaster/u/friedel'
PAPCO_HOME=     HOME+'/papco_user'
PAPCO_DOWNLOAD= HOME+'/papco_user/papco_download_temp'

print, '--- Expanding IDL search path for PAPCO core, Lib, Home --------------'
print, ''

!path = !path +':'+ PAPCO_BASE + '/papco'
!path = !path +':'+ expand_path('+'+PAPCO_BASE+'/papco_lib')
!path = !path +':'+ PAPCO_HOME

; compile module that contains all routines for platform
; independance. This needs to be done right at the beginng, here!

print, '--- Compile PAPCO routines needed for startup ------------------------'
!quiet=1
@papco_startup_init
; print out the current PAPCO version.
print, ''
print, '=================================================================='
print, '--- ',papco_version(),' -----------'
print, '=================================================================='

; setting all environmental variables need by PAPCO.

; PAPCO_QUIET and PAPCO_STARTUP_MODE are used by the startup widget
papco_setenv,      'PAPCO_QUIET=1'
papco_setenv,      'PAPCO_STARTUP_MODE=COMPILE'
papco_setenv,      'PAPCO_ADDINS=NO'
; PAPCO_HOME, PAPCO_PATH, PAPCO_MODULES, PAPCO_DOWNLOAD, TIME_CONV  are used by
; the papco core programs
papco_setenv, /FS, 'PAPCO_HOME='+PAPCO_HOME
papco_setenv, /FS, 'PAPCO_PATH='+PAPCO_BASE+'/papco'
papco_setenv, /FS, 'PAPCO_MODULES='+PAPCO_BASE+'/papco_modules'
papco_setenv, /FS, 'PAPCO_DOWNLOAD='+PAPCO_DOWNLOAD
papco_setenv, /FS, 'TIME_CONV='+PAPCO_BASE+'/papco_lib/cdssw/time'
; the XXXXX_ORBITS variables are used for the plot by orbit utility
papco_setenv, /FS, 'POLAR_ORBITS=/n/papco/data/polar/eph'
papco_setenv, /FS, 'CRRES_ORBITS=/n/papco/data/crres/eph'
; USER_MOUSE_ACTIONS can be used to configure new mouse actions by the user
papco_setenv,      'USER_MOUSE_ACTIONS=View_Polar_Slice/View_Ephemeris/Distribution'
; USER_SHOW_ACTIONS selects the action in the drZw window associated
; with the above USER_MOUSE_ACTIONS
papco_setenv,      'USER_SHOW_ACTIONS=hair_cursor/hair_cursor/vert_line'

!quiet=long(getenv('PAPCO_QUIET'))

print, ''
print, '--- Environmental variables now set: ---------------------------------'
print, ''
print, 'PAPCO_QUIET:        ',getenv('PAPCO_QUIET')
print, 'PAPCO_STARTUP_MODE: ',getenv('PAPCO_STARTUP_MODE')
print, 'PAPCO_ADDINS:       ',getenv('PAPCO_ADDINS')
print, 'PAPCO_HOME:         ',getenv('PAPCO_HOME')
print, 'PAPCO_PATH:         ',getenv('PAPCO_PATH')
print, 'PAPCO_MODULES:      ',getenv('PAPCO_MODULES')
print, 'PAPCO_DOWNLOAD:     ',getenv('PAPCO_DOWNLOAD')
print, 'TIME_CONV:          ',getenv('TIME_CONV')
print, 'POLAR_ORBITS:       ',getenv('POLAR_ORBITS')
print, 'CRRES_ORBITS:       ',getenv('CRRES_ORBITS')
```

```
print, 'USER_MOUSE_ACTIONS: '
uma=getenv('USER_MOUSE_ACTIONS')
uma=str_sep(uma,'/')
for i=0,n_elements(uma)-1 do print,'                    ',uma(i)
print, 'USER_SHOW_ACTIONS:  '
usa=getenv('USER_SHOW_ACTIONS')
usa=str_sep(usa,'/')
for i=0,n_elements(usa)-1 do print,'                    ',usa(i)
print, ''
print, '--- Done papco_set_variables -----------------------------------------'
```

3.3.2 papco_include_modules.pro

This batch file is called from `papco_startup.pro` and expands the IDL search path to include the modules selected. *You should never need to edit this file*, as it is automatically written by the `papco_configure` widget (see Section 3.4).

This file will always reflect the last state of PAPCO, containing all the paths to modules last chosen.

The distribution comes *without* this file. It will be created the first time you configure PAPCO when you exit the `papco_configure` widget.

3.3.3 papco_include_addins.pro

This batch file is called from `papco_startup.pro` and is an empty batch file included to enable the user to add in any initialization needed for further add-in IDL applications called from PAPCO.

You should never need to edit this file, as it can be edited through the `papco_configure` widget (see Section 3.4).

Typically you would include one line each for the batch file which "adds in" your application, eg:

`@my_IDL_application.pro`

The batch file `my_IDL_application.pro` typically expands the IDL search path to point to where your applications `.pro` files reside, and then compiles all routines needed.

Such add-ins are considered advanced PAPCO and should not be attempted by the casual user.

Often this add-in facility can be used to compile other code in the PAPCO environment, even if one does not intend to use PAPCO itself. This is useful if you want to make use of PAPCO and PAPCO modules routines for other applications!

3.3.4 papco_restore.pro and papco_compile.pro

These batch files are called from `papco_startup.pro`. Which one is called is determined by the `papco_setup` batch file (see Section 3.4).

papco_compile.pro

This batch file compiles PAPCO core and module procedures and functions, which can take some time. It has to be called each time you modify any procedure in PAPCO core (which you shouldn't) or any of the modules (which you may), or if you setup PAPCO with a new set of modules.

```
;******************************************************************************
;* FILE: $PAPTO_PATH/papco_compile.pro
;*
;* DESCRIPTION: Run the initialisation files to freshly compile (.run)
;*              PAPCO core and PAPCO modules.
;*
;* CALLED BY:   idl_papco_startup
;*
;* MODIFICATION HISTORY:  written July 1997, R.Friedel
```

```
;****************************************************************************
print, ''
print, '--- Start papco_compile --------------------------------------------'

; calling papco_init: compile PAPCO core
@papco_init

print, ''
print, '--- Compile add-in modules: only those found in module paths: --------'
print, ''

papco_find_modules
COMMON MODULE_PATHS, paths

if paths(0) eq '' then print,'% papco_compile: No modules configured for PAPCO'

; calling papco_add_init: compile the modules found
@papco_add_init

print, ''
print, '--- End papco_compile ----------------------------------------------'
```

Here the batch file `papco_init` is called, which compiles (.run) all the procedures and functions needed by PAPCO core. It resides in the PAPCO core directory `papco_XX\papco`.

Before compiling the procedures needed for all chosen modules, the routine `find_modules` is called. This routine searches the IDL search path for all the initialization files contained in each module directory. All these init file paths are written to the batch file `papco_add_init.pro` in the users `papco_user_XX` directory, which is then executed.

papco_restore.pro

This batch file restores a previously saved set of procedures and functions from an IDL saveset file. This file of course has to exist for this: It is made when you exit PAPCO using the "Save&Done" button, which writes a file `papco_routines.save` to your `papco_user_XX` directory.

This startup mode is exceedingly quick, and should be used whenever you work with a static PAPCO setup. Using this saveset file also allows you to run a given PAPCO setup from an IDL "runtime" license (which is a lot cheaper than the full IDL license).

```
;****************************************************************************
;* FILE:        $PAPTO_USER/papco_restore.pro
;*
;* DESCRIPTION: Restore PAPCO core and PAPCO modules compiled procedures
;*
;* CALLED BY:   idl_papco_startup
;*
;* MODIFICATION HISTORY: written July 1997, R.Friedel
;****************************************************************************
print, ''
print, '--- Start papco_restore --------------------------------------------'
print, ''

print, '--- Restoring procedures from ---------------------------------------'
print, '    ',papco_getenv('PAPCO_HOME')+'papco_routines.save'
restore,papco_getenv('PAPCO_HOME')+'papco_routines.save'
print, ''

; finding Current modules
papco_find_modules

print, '--- End papco_restore ----------------------------------------------'
```

Here previously saved procedures and functions are restored utilizing the IDL save/restore procedures. The routine `find_modules` needs to be called since it also generates a common block of module names needed by the PAPCO main routine.

3.4 papco_setup.pro

This batch file calls `papco_startup.pro`, but it calls the `papco_configure` widget (see Section 3.5) first and then calls a program which rewrites `papco_startup.pro` and `papco_include_modules.pro` accordingly. Use this file to "configure" PAPCO or to load up previous PAPCO configurations, use `papco_startup.pro` to run the latest previous configuration of PAPCO.

```
;***************************************************************************
;* FILE:        $PAPTO_USER/papco_setup.pro
;*
;* DESCRIPTION: first calls papco_configure and then papco_startup used if the
;*              way in which PAPCO starts up needs to be changed,i.e.:
;*              a different set of modules is to be loaded
;*              the startup mode is to be changed (compile or restore)
;*
;* CALLED BY:   none
;*
;* HOW TO START:
;*    from command line: type idl papco_setup   (must be in current dir)
;*        or, in your shell set up an environmental variable IDL_INITIALIZE
;*        which points to the location of papco_setup
;*        eg for ksh:  IDL_STARTUP=~/papco_setup
;*                     export IDL_STARTUP
;*        then simply type idl
;*    from IDL:   type @papco_setup (if papco_setup.pro in local dir)
;*             or type @PATH/papco_setup (if papco_setup.pro in PATH)
;*    afterwards, you can start PAPCO by executing the command "papco".
;*
;* MODIFICATION HISTORY:  written July 1997, R.Friedel
;***************************************************************************

; set up some variables used in this startup file
@papco_set_variables

; call configure widget and change papco_startup file

print, ''
print, '--- Running papco Configure widget -----------------------------------'
print, ''

papco_configure
papco_change_startup

print, ''
print, '--- Done papco Configure --------------------------------------------'

; Run normal papco_startup
@papco_startup
```

3.5 papco_configure.pro

This program is a startup widget which is part of PAPCO core. As PAPCO is a configurable package, the user can choose which modules he wants to start PAPCO with. In addition, such a configuration of modules can be saved and previous configuration can be loaded. You can also edit the `papco_include_addins.pro` file from this widget (see 3.3.3).

This widget is shown in Figure 3.1 and is called when PAPCO is initialized with the `papco_setup.pro` batch file. The current (or last) installation of PAPCO is always reported by this widget and can then be modified. The modified configuration then becomes the current configuration.

This widget always reports all the modules that are currently included in the PAPCO distribution. The user has the option of adding further modules from any other location. By default, additional modules are assumed to be in `papco_user_XX\papco_user_modules`.

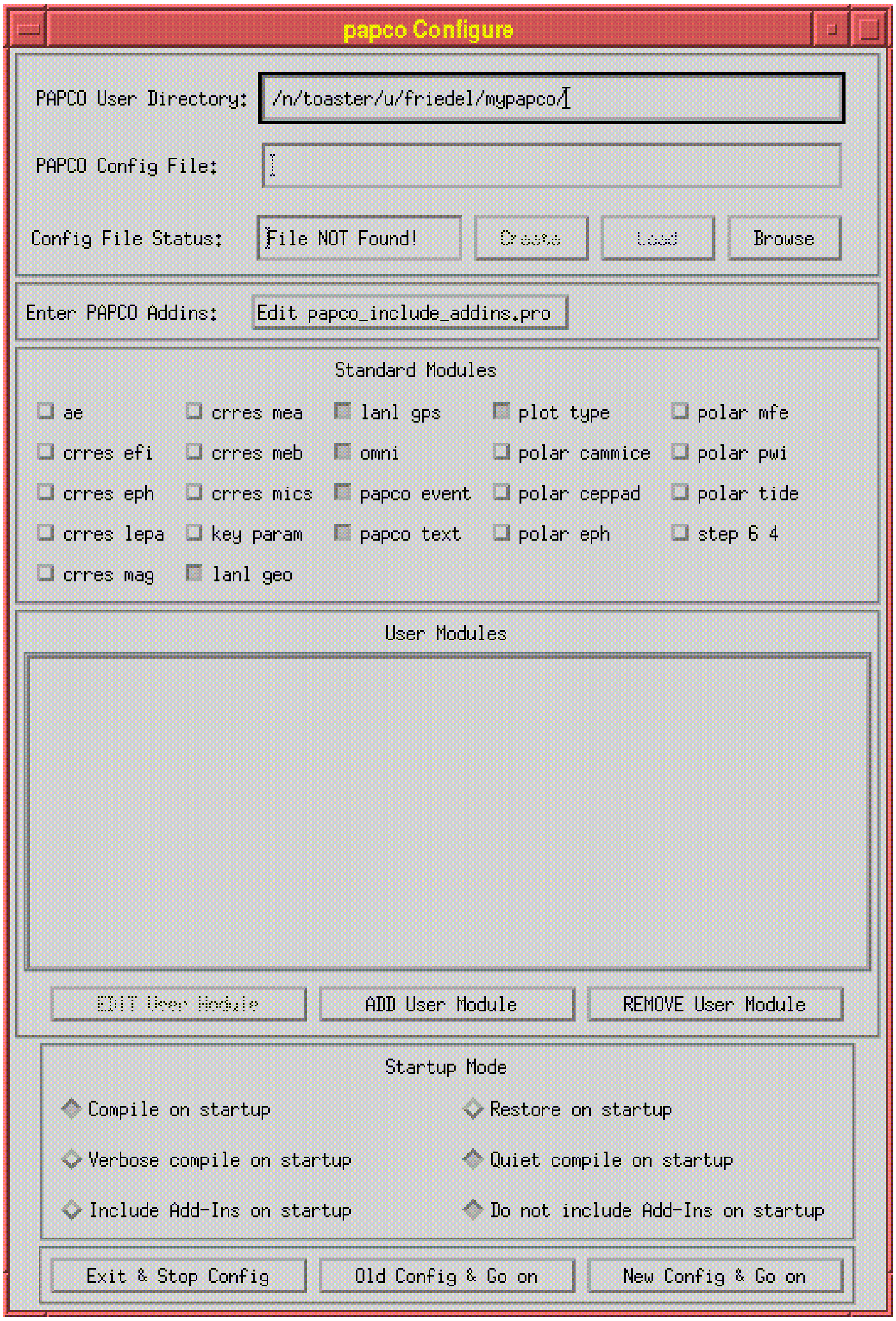


Figure 3.1: PaPCo Configuration Widget

Load/Save PaPCo configuration

The first part of the configuration widget uses the PaPCo file widget to load or save PaPCo configurations. By default the configuration files are stored in your `papco_user_XX` directory. You can type in a configuration file name or use the Browse button to choose one. Configuration files have the default extension `.papco_config`. Once a file has been specified you can either save the current configuration to it or load the stored configuration from it. The `.papco_config` have exactly the same structure as the `papco_include_modules.pro` file (see 3.3.2).

Enter PaPCo addins

This button calls up a simple editor to edit the `papco_include_addins.pro`file (see 3.3.3).

Standard Modules

The first set of buttons show all the modules that were part of the PaPCo distribution and that reside under `papco_XX\papco_modules`. Click any combination of them as needed.

User Modules

Here the user can "Add", "Remove" or "Edit" further modules which reside elsewhere.

To add a new module use the "Add" button which brings up the widget shown in Figure 3.2.

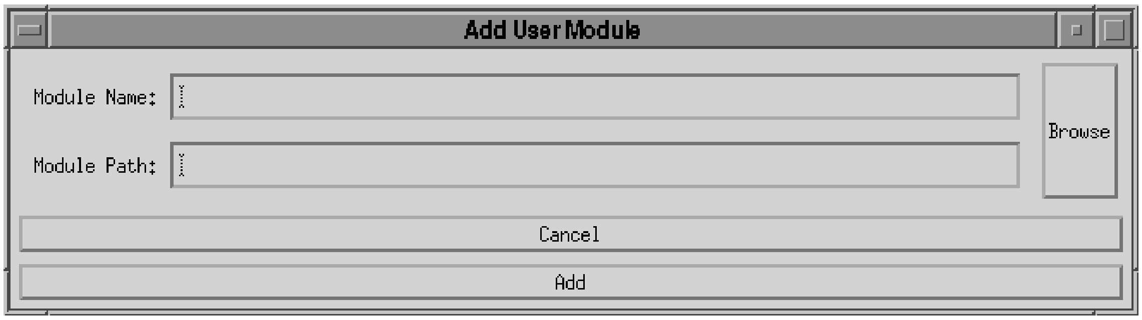


Figure 3.2: PaPCo Add User Module Widget

The module paths can either be typed in explicitly or browsed for with the standard IDL file choose widget, accessed through the "Browse" button. The last part of the module path chosen is automatically used as the module name for the list (you can override this). Once a path has been typed or chosen it is added to the list using the "Set" button.

N.B. The "Browse" function does not operate properly under Windows95 as IDL for Windows95 uses the standard Windows95 file choose widget, which does not return paths only. You need to type in the path manually.

By default, additional modules are assumed to be in `papco_user_XX\papco_user_modules`. However, you can define any path you want.

Startup Mode

The startup mode selects the "compile" or "restore" mode as explained in Section 3.3.4. The "restore" option becomes unavailable whenever the module configuration has been altered, or if no `papco_routines.save` file exists under the `papco_user_XX` directory.

Further, you can set the compilation of PaPCo to proceed "quietly' or 'verbosely". For quiet, no compile message are output. Use this only once you know that your installation of PaPCo compiles without problems, as you will not be warned about any compilation errors!

The include Add-Ins choice allows you to either include or skip the Add-In batch file on startup.

Action Buttons

"Exit & Stop Config" exits the configure widget and stops the compilation process.

"Old Config & Go on" ignores any changes you might have made and proceeds with compilation with the previous configuration.

" New Config & Go on" compiles with your new configuration settings and makes all changes required so that the new configuration is "remembered" for the next startup.

3.6 Start-up Summary

A sample output of what you should see in you IDL command window when you start PAPCO is given in Appendix A.1. Starting from the operating system prompt, from the directory `papco_user_XX`, PAPCO is normally started by

`idl papco_setup`

This calls up the configure widget before proceeding with the PAPCO initialization. Use this if you want to setup which modules and which startup mode to use.

Once your setup is static you can use

`idl papco_startup`

to initialize the last setup of PAPCO. This is a bit quicker, since it skips the configuration widget.

The start up then proceeds in the following order:

1. IDL is started.
2. The batch file `papco_setup.pro` or `papco_startup` is processed:
   (a) The IDL search patch is extended.
   (b) PAPCO environmental variables are set.
   (c) Module paths are added to IDL search patch
   (d) Core PAPCO programs are compiled.
   (e) User Modules are compiled.
3. The PAPCO main widget is then started from the IDL prompt by typing
   `papco`
4. All module directories found in the search path are parsed for all files starting with `papco_interface/papcoadd_` and the button widgets for those modules are constructed.
5. Each module is checked for plot sub-types and they are added to a list of valid plot_types.
6. The configuration settings for the modules are read. This is done by first checking the `papco_user_XX\papco_modules_defaults` directory. If no configuration file is found there, the default configuration file `defaults.config` from the modules `plot_read` subdirectory is read, and a copy is written to the `papco_user_XX\papco_modules_defaults` directory.
7. The printer configuration file is read and a list of available printers is reported.
8. The defaults file is read.
9. PAPCO is ready for use.

The first time you start up PAPCO you also need to do the following:

1. Should the main PAPCO widget be too large for your screen, use the "Font" button to choose a smaller Font. PAPCO resizes with Font size (See Section 4.5).

2. If the draw window is too large/small, or the character size for annotation is too large/small etc use the "Defaults" button (See Section 4.5).

3. Each module, when it is included for the first time, needs to be configured for your site. Each module *should* include a "Config" button as part of its panel editor. You need to call up the module configure widget to set data paths and such like for your module (see Section 6.4.1).

3.7 Installing new PAPCO modules

Before installing any new module make sure that you inform the module author - you will probably need his help to obtain some data anyway! Make sure that the module is compatible with

- Your platform. Unfortunately most module authors don't pay much heed to cross-platform compatibility. Most modules should work under UNIX though.

- Your version of PAPCO.

3.7.1 Adding a new PAPCO module

You will probably obtain the module off the Web as a zipped tar file. The best place to unzip & untar it into would be your `papco_user_XX\papco_user_modules` directory.

You will then need to exit IDL and reconfigure PAPCO to include that module. In the PAPCO configuration widget, add the new module to the User Modules section (see 3.5). Once you start PAPCO , a button for the new module should appear in the top part of the main PAPCO widget.

BEFORE USING the new module you will need to configure it, by using the configure option of the new module's panel editor of the configure sub-menu choice. Newer modules will not allow you to use them before they have been configured! Edit the the configuration for your site and then don't forget to save it. See section 6.4.1 for details on the panel editor config widget.

Review the information in the module's panel editor help file. This is the place where all the information about the module *should* be. This file is under the module `papco_interface` directory. You can also access it with the help button of the module's panel editor.

(If you added the module into you main PAPCO core `papco_modules` directory, the module would appear as one of the Standard Modules in the configuration widget. This is not encouraged, as you would "loose" the module when you upgrade the PAPCO core to a new version)

3.7.2 Updating to a new PAPCO module version

This could be either a module which is part of the distribution or one of your own modules in `papco_user_XX\papco_user_modules`. Simply copy the new module to the same location - it would be prudent to keep a copy of the old module though (simply rename the module directory name).

Once you have the module loaded in PAPCO you will again need to re-configure it - the modules config options might have changed. To do this use the module configure option and re-load the module defaults by hitting the defaults button. Edit the the configuration for your site and then don't forget to save it. See section 6.4.1 for details on the panel editor config widget.

3.8 Troubleshooting

There are several reasons for PAPCO not initializing properly, or crashing because some or other routine couldn't be found:

1. The paths to the PAPCO package and/or modules are not set properly.
2. There is a conflict between the way PAPCO sets up IDL and the way IDL is set up by the system or current user.
3. There is a naming conflict between PAPCO routines and other IDL routines on the system (and in the IDL search path).

3.8.1 Are you using an own idl_startup.pro?

On starting IDL, one can give the name of a batch file on the command line that is executed on start-up. This is the method employed by PAPCO. However, if the environmental variable `IDL_STARTUP` is set to the name of a different start-up file, then IDL will execute this batch file, on start-up, BEFORE the file on the command line is executed. If this different start-up file also contains a command expanding the IDL search path, then the PAPCO paths will be appended to that path. Now, if IDL searches throughout this path to find a .pro it needs to compile, then it will compile the one that is found first; and if there are duplicate file names present, then the one it compiles might not be the one PAPCO needs!

The way to get around this, and to make sure that IDL ONLY searches the path set up by PAPCO, you need to unset the environmental variable `IDL_STARTUP`. Under UNIX the command for this is:

```
unset IDL_STARTUP
```

3.8.2 Are my own routines found?

The only way your routines can be found is when the path to their location is included in the IDL search path `!path`. For the PAPCO core routines this is done in This is done in the file `papco_set_variables.pro`, and for the modules in the file `papco_include_modules.pro` which is written by the configuration widget program. Check this file to make sure all paths are correctly set.

Traditionally all the module directories are subdirectories under `papco_XX/papco_modules`, but this is not prescribed. Putting them there just simply makes it easier to distribute them with the whole PAPCO package using a tar file. But since you can include the path to your modules in the PAPCO configuration widget, they can be located anywhere, even on other workstations! The idea here is that you can use whatever is already existing on your workstation and interface those modules to PAPCO by simply expanding the search path accordingly.

Just beware that none of your routine names (or procedure/function names) are in conflict with PAPCO routines!

The additional files and directories needed in your module directory are described in Chapter 6.

3.8.3 Avoiding naming conflicts

In any large package such as PAPCO the chances of naming conflicts arising is large. When adding your own modules, or interfacing your own widget application, you might end up having procedure or function names that have already been used by PAPCO.

We have tried to give unique names to all the routines in core PAPCO (most of them starting with `papco_`), but of course have no control over what anyone else does. Instead of prescribing some rigid naming convention for everything (although there IS a naming convention for the interface files for your modules for PAPCO),

PaPCo provides a utility for searching and documenting an IDL tree structure of programs, which provides warnings on duplicate file, procedure, functions, and structure names.

You may use the program 'comments.pro' in `papco_XX\papco_lib` to extract this information from IDL-files. Issue those commands from IDL:

```
.run string_functions
.run comments
```

then, use the routine `commentsFor` by specifying the path you want information for - all files in that directory and in its subdirectories are analyzed and information is written to some files:

`commentsfor, 'papco_XX/papco', 'papco'`

The first parameter specifies the directory start point to analyze (automatically goes through subdirectories), and the second the file name root for the output files (can include a path).

The second parameter is optional - if you omit it, the information is written to `~/protocol.xxxx`.

This generates the following files:

papco.def_warnings Warns you of multiple definitions in different files.

papco.definitions List definitions of functions, procedures and structures found in any of the files (sorted by definition name).

papco.proc_ref This is almost the same as papco.definitions, but the definitions are not complete - they are abbreviated, so that you get a quick overview (sorted by definition name).

papco.file_warnings Warns you of file names that appear in more than one directory.

papco.files List all .pro-files and the definitions made in them (sorted by file name)

This is extremely useful for finding out what routine is where and how it is defined, and serves as a mini-reference guide to the whole program structure. It also warns you of duplicate definitions in the two _warnings files.

This has been done for the files in:

```
papco_XX
papco_XX\papco
papco_XX\papco_lib
papco_XX\papco_modules
```

The output files are in `papco_XX\papco_doc\commentsfor`. These files are very useful for the developer to trace down faults, but are also useful as reference material on what files and procedures are defined where.

It is strongly recommended that you use this procedure to check your own module for naming conflicts, and then include your module under the `papco_XX` tree and check again for any naming conflicts with existing PaPCo procedures or modules.

If all your modules are placed under `papco_modules` then running

`commentsFor, 'papco_XX', 'papco'`

goes through the whole of the PaPCo structure INCLUDING your module. Any conflicts are then listed in papco.def_warnings and papco.file_warnings.

3.9 Current PAPCO modules

A list of all current modules and their authors and Web-sites is maintained under the PAPCO home page[4] which is part of the CCR Web site[5]. The CCR Web page has a mirror at RAL, United Kingdom[6].

Click on "List of current PAPCO modules" from the PAPCO home page.

Listing of modules in this manual will be discontinued as module availability changes rapidly.

A subset of the above modules is included in the standard distribution of PAPCO. The decision to include modules with the distribution lies with the PAPCO module home institutes, some may prefer to have more control over who gets their module and restrict distribution to occur through their own site.

As PAPCO modules mature and become public inclusion with the PAPCO distribution is encouraged. For module updates between PAPCO releases please refer to the module sites as listed on the PAPCO home page.

[4] http://leadbelly.lanl.gov/ccr/software/papco/papco.html
[5] http://leadbelly.lanl.gov/ccr/
[6] http://shaper.bnsc.rl.ac.uk:8080/ccr/CCR.html

Chapter 4

Running PaPCo

In this chapter the functionality of PaPCo is explained by giving examples of the widgets you will encounter, and describing the action of each button or slider.

The top level widget is the widget that appears after you start PaPCo. All PaPCo functions are controlled via this window, it is here that your plot is "composed" and manipulated before it is actually drawn or plotted. For the following description of the various buttons refer to Figure 4.1.

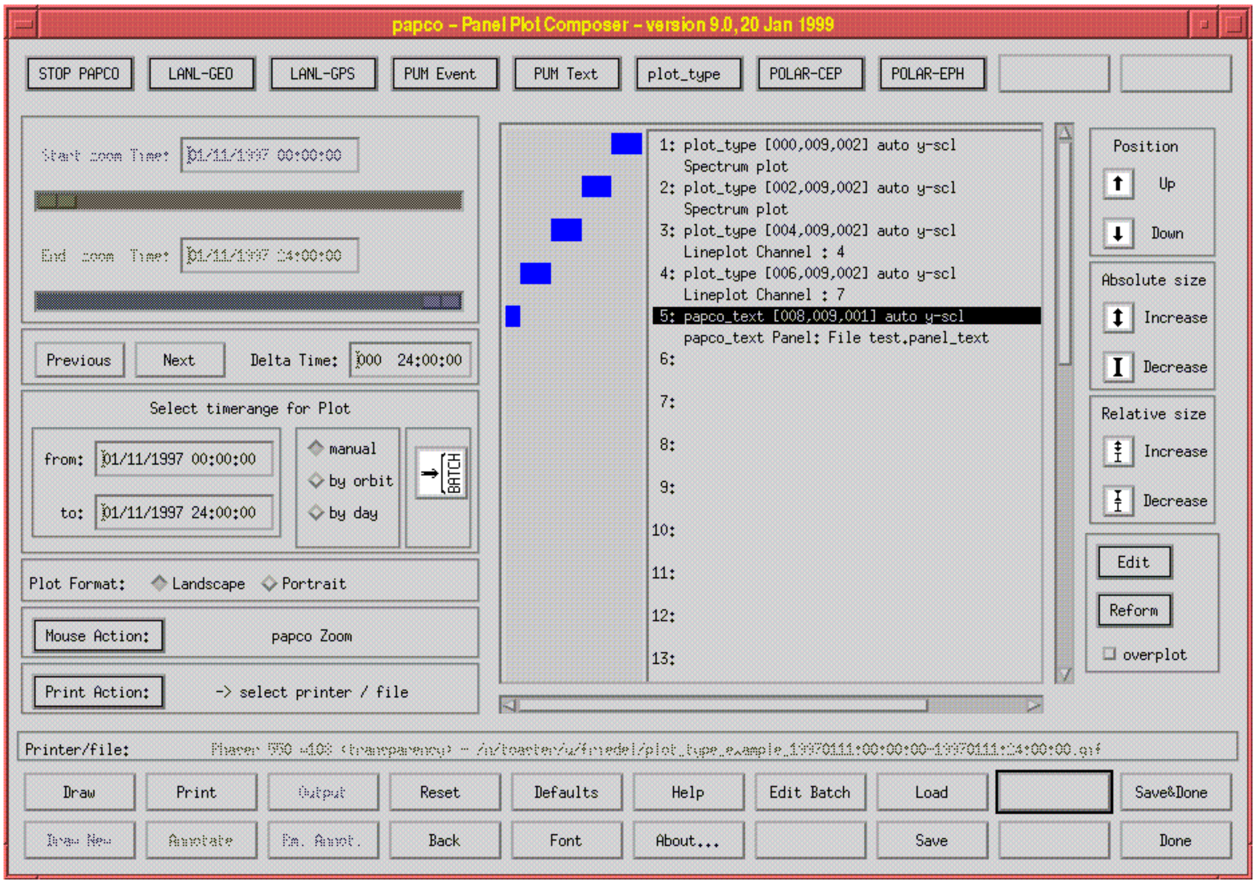


Figure 4.1: Top-level PaPCo widget

The PaPCo main widget is divided into four parts. The top part contains all the Module Buttons PaPCo was configured with. The middle part contains a left hand side, which is the "Control Panel" and a right hand side, which is the "Panel Composer". The bottom part contains all the Action Buttons.

4.1 The top widget part: The Module Panel

4.1.1 The STOP PAPCO button

This is provided mainly for debugging purposes for the module developer. PAPCO execution is suspended and the IDL prompt returned. At this point all IDL routines of the PAPCO package apart from the top level routine papco.pro may be recompiled. For example, if you wants to change the appearance of your plot you can edit your plot routine and recompile it. By typing

```
.con
```

you can then continue in PAPCO from where you left off. If your edit produces a crash, you will get an error message. See section 4.9 on how to proceed.

NOTE: While PAPCO is stopped, all variables of the PAPCO main routine are accessible and can be investigated using the IDL `print` or `help` commands.

4.1.2 The module buttons

These appear next to the STOP PAPCO button. ALL modules found are added here with their own user-defined button. Each button accesses a user-written module panel editor widget which is used to set the parameters for the required plot, or a further sub-menu of several plot types for the data, e.g. color spectrograms or line plots. The contents of these panel editors and /or structure of the sub-menus are user-defined. An example of such an editor is the `plot_type` panel editor template, described in more detail in Section 6.4.1.

PAPCO provides for a basically unlimited number of possible product buttons. A new row of buttons is added whenever needed: this will make the widget grow. So the size of your screen becomes limiting to the number of modules you can have added any given time!

Note that PAPCO configures itself to use only those modules it finds in it's search path, which is user-configurable using the PAPCO configure widget (see Section 3.5).

4.2 The middle left widget part: The Control Panel

4.2.1 Time slider and label

The top and bottom time slider are set to the selected time range for the plot. The associated text windows show the selected times and are also editable, for exact zoom selection. The user may select a sub-time range by either adjusting the sliders with the mouse, or by entering a start and stop zoom time manually. Checking is provided to make sure that the zoom start time is always less than the zoom stop time.

The new times will be shown in the associated text windows.

Note that this group of widgets is inactive so long as you have not drawn a plot. You need to define an overall time range first with the next group of widgets, and draw the plot before you can zoom into it.

4.2.2 Time skip

The "Previous" and "Next" buttons allow the user to step through the same time interval easily. Delta time is always set to the difference between the current start and end times. This is not an entry field.

4.2.3 Manual time entry

There are three different ways of selecting a time range: By start and end time, by orbit and by date. The first entry type is the most general, and the last two reflect the way data is normally stored in files (one file per day or per orbit). The three entry methods are toggled by push buttons, and each entry method has it's own widget which appears to the left of the buttons.

Entry by time

When entry by time is chosen the left side of this widget group changes to two text entry fields for start and end time. This can then be entered in a variety of format combinations of day, year, month, day of year and time. The supported formats are accessible via the help button; a wrong entry will bring up a message window with the permitted formats.

Entry by orbit

When entry by orbit is chosen the left side of this widget group changes to an text entry field, a satellite button and a List Orbit button.

The user can enter an orbit number in the show orbit text window, press return and the start and end times for that orbit are automatically read and shown in the above slider and text output widgets. The List Orbit button calls up a separate window which reads the file containing all the orbit information, which shows all orbits and their corresponding times and allows the user to click on a desired orbit. One of the options in this widget is to re-read the orbits info file. This causes all the files containing the ephemeris information for that satellite to be reread, and a new orbit info file is written. In this way the system can expand if more orbits become available.

This functionality is only supplied for CRRES and POLAR at the present time, but will be extended to other satellites. PAPCO is NOT distributed with the orbitinfo files needed for this functionality, as they are considered to be part of the CRRES and POLAR modules. So this functionality will not be available if those modules are not loaded.

Note that here we use the orbit only to set a start and end time: you can then use these times with ANY modules that you've added. Obviously this makes most sense with the corresponding modules only! PAPCO does no checking here. For CRRES the orbit times given are perigee to perigee, for POLAR apogee to apogee.

The way this functionality has been added to PAPCO has been generalized such that it is relatively easy to include "new" missions. This still requires changes to PAPCO core, and is thus considered advanced PAPCO. A guide to doing this is given in Section 6.8.

Entry by day

When entry by day is chosen the left side of this widget group changes to a text entry field for the day. The user enters only the date information and the full time range for that day gets selected.

The batch button

PAPCO includes the capability of processing plots by batches: This means that for the currently defined plot product (user defined combination of panels) a series of output plots at different times can be produced.

The batch output option is accessible via the PRINT button (see below) and will normally read a file with a list of times for which plots are to be produced.

Using the batch button here allows the user to automatically make such a file. Pressing this button adds the current time range to a default file called `interactive.papco_batch` in the `papco_users_XX\papco_products\batch` directory. If the file does not exist, it is created; if it does, it is appended.

The idea here is that the user can put together a set of plots of a given product which all belong to a certain group, e.g. all those data times that have boundary crossings, or substorms, etc., interactively. This group of plots can then be processed as a batch to produce hardcopy. For co-workers in the same field that both have PAPCO and want to work on the same topic all that needs to be exchanged is the data product and the associated batch file.

4.2.4 Graphics output format choice

The next group of buttons selects the output format between landscape or portrait. This affects the printed output for the next print operation and the appearance of the PAPCO Draw Window for the next Draw Window created.

Currently PAPCO produces printed output in A4 and letter format only. Choosing US or Europe in the defaults widget (3.1) sets the output format.

4.2.5 Mouse Action

The Mouse Action button accesses a sub-menu of all current mouse types built into PAPCO. The information label next to it shows the current mode.

Added to this list are any User Slices that you have configured PAPCO with. User Slices are described in more detail in Section 6.9.

If a module supports the selected mouse action clicking in the corresponding plot panel will perform that action, if not, nothing will happen. Check the module's panel editor help, it *should* tell you which slices it supports.

4.2.6 Print Action

The Print Action button accesses a sub-menu of the various printer actions that will then be performed when the "Print" button in bottom widget part is pressed. The submenu has several choices:

→ **standard printer** Sends output to the standard default printer

→ **standard file** Sends output to the standard default file

→ **select printer / file** Brings up the printer widget (see Figure 4.2) to select a specific printer / filename / output format for the next print job.

select standard output Brings up the printer widget (see Figure 4.2) in a different mode which allows you to select the default printer, default output format, default file name template. These defaults are "remembered" and stored when you exit PAPCO.] This widget comes up immediately upon selection - you don't need to press "PRINT" first.

batch → screen preview Calls up the batch file select widget and then processes through the batch file with output going to the draw window.

batch → data output Calls up the batch file select widget and then processes through the batch file with output going to screen but also to file for those panels that support the output functionality.

batch → default file Calls up the batch file select widget and then processes through the batch file with output going to the currently selected default output file template.

The printer widget

The printer widget (see Figure 4.2) is a multi-purpose widget which allows you to either select a print output destination or to set print/output defaults.

When you choose the option → **select printer / file** this widget is in the "direct graphics output mode". If you have configured printers using the `papco_printer_config.dat` file (see Section 3.2) then a list of your printers should appear. This list becomes active when you choose "Output to printer".

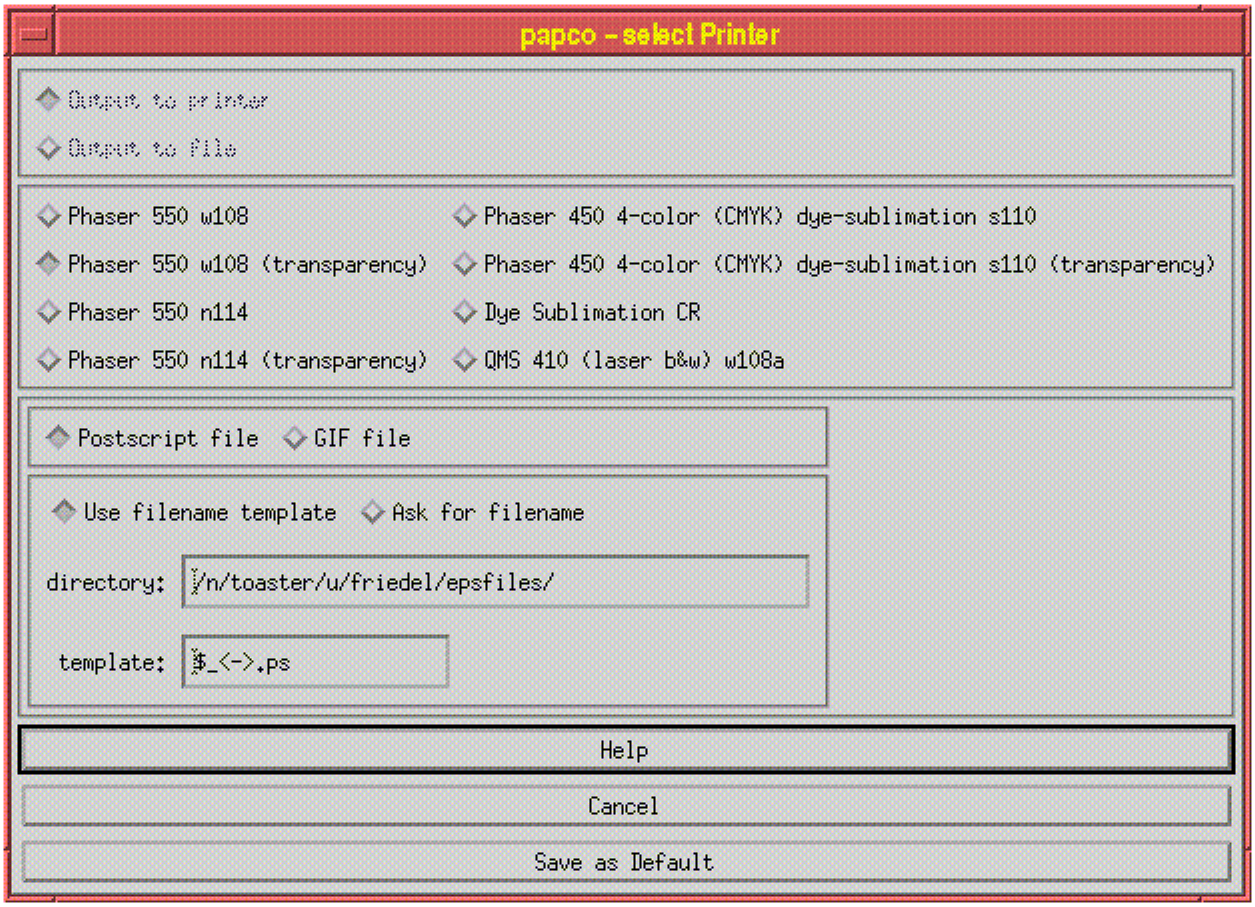


Figure 4.2: PaPCo Printer Widget

When you choose "Output to File" the remained of the widget becomes active. here you can choose between postscript or gif output (the files will have extension .ps or .gif), whether to use the file template or be asked for a filename. You can set the output directory for the graphics output files and choose a filename template

The file names are configurable and include "place holders" which are substituted with the current time range or orbit number, whichever way the user selects. In this way one can guarantee unique output file names:

"$" stands for the product name (the name of the product save file)

"%" stands for orbit number

"<" stands for the start time

">" stands for the start time

Any other alphanumeric characters can be used in combination. VMS users: Avoid using "." in filenames!

When this widget is called with the option **select standard output** it start up in the "set defaults mode". All the options described above apply, but whatever you choose becomes the default for printing to file/printer. It is these defaults that will get used when you use the → **standard file** and → **standard printer** options.

The currently set default printer and default filenames are displayed in the label widgets just above the Action Buttons.

4.3 The right widget side: The Panel Composer

The right side of the PAPCO main window is the actual panel composer where you can custom make your plot product. Each time a plot is chosen with its corresponding panel editor a panel entry is made into the compose area: An entry is made into the panel list, and a schematic representing the position of the panerl is drawn into the graphics area to the left of the list (think of it as the graphics output window in small, rotated by 90 degrees anti-clockwise).

Up to thirty panels can be entered, the whole list is scrollable. Initial positioning of the panels is from the bottom up, all panels equal size. The edit buttons provided here allow the user to edit existing entries, delete them, change the positioning and or size, put plots over each other and selecting one of the panels to be an "overplot" (plots into same panel as previous plot adopting the previous axis scaling).

The next section 4.3.1 explains the functions of the composer in more detail. However, the best way to learn how to use it is to add a few panels and then just to play around with the white positioning and sizing buttons.

Once you have "composed" a plot, this can be draw by clicking on the "Draw" action button, this opens up a draw window and the panels are processed and drawn by PAPCO in the order from top to bottom in the panel list, overplots last.

The entries in the panel composer corresponding to a plot are called a "product" in PAPCO terminology. These products can be saved and restored for later use using the Action Buttons "Load" and "Save" (see Section 4.5 for an explanation of the action buttons). These product files then also serve as input to batch processing, and the product name can be used as part of the printfile template (see Section 4.2.6).

Once you have drawn a product, you might want to change the appearance of some of the panels, the size and positioning, or overall plot appearance. To change a given panel you can edit that panel's settings using the panel editor for that panel. To call up the panel editor,you can

- double click on the panel's entry in the panel composer, or
- highlight the panel's entry in the panel composer and then use the "Edit" button (this gives you the choice of editing the panel or replacing it with a different one alltogether).
- click in the active draw window to the left or right of the panel (not inside the panel, that activates the current slice function!)

We have made an attempt to do "clever re-draws". Once a product has been drawn and you press the action button "Draw" again, only those panels that have been edited will be redrawn. This speeds up the design of a product considerably! (NB. To force a complete redraw, simply re-enter the current time settings for the plot).

4.3.1 The compose area functionality

The panel compose area of the main widget is where the main "work" gets done, this is what it's all about. As you select plots from the various modules an entry is made in the compose window. Entry in the compose window is from the top down in the order that you add panels, and this order is adhered to when plotting, regardless of which actual position that panel eventually has.

In addition to the text entry, a schematic black bar entry is made to the small graphics (grey) area at the left side of the compose area. This shows you the relative position of your panels as they will be plotted. Reading this might take a bit of practice: the position of the bar relates to the actual position of your panel once you plot it in the following way: far left is at the bottom, and far right is at the top. The bar is normally blue, if you overlap any panels both affected panels turn red, and if one is an overplot (see Section 4.3.6), a red outline bar only is drawn. Again, the best way to learn is to play around with the positioning arrows and see what happens.

NOTE: Drawing of all panels in the graphics area, however, is from the bottom up. The top entry in the composer ends up as the bottom panel. Since we can give no explanation for this, we just say that this is due to historical reasons!

(Remember that the panel which end up at the bottom of the plot also draws the time axis plus any additional x-axis labels. This might explain why it is at the top of the list, as it is sort of important!)

4.3.2 The panel list entries

This entry describes the panel, the entry string is made up out of default information plus a user-defined string that the user sets in his user-written module. For example:

```
2: CRRES-Spec-Plot: [ 2, 7, 2], auto-yscl
elec, diff, pa range 10
```

The first line contains the number of the plot in the order in which they were chosen (which is also the order in which they'll get plotted), a description string of the plot type, the panel vector, which has the following meaning (by index):

0: position of panel, 0 being the bottom, counted from bottom up

1: number of total panels

2: height of the panel in number of panels

So [2, 7, 2] means plot at position 2 of 7 panels with a height of 2 panels. Next is auto-yscl that means that the plot uses its own built-in user-supplied scaling. The user can select manual scaling in the panel editor for that plot type to override the automatic scaling.

The next line is the user-defined part, this string is meant to be a very brief identifier for the plot. Here we plot differential electrons of pitch angle index 10, which for CRRES is 90 degrees.

4.3.3 Positioning buttons

The white arrow buttons allow the user to manipulate the position and size of the chosen panel entry (highlighted in black). Any combination of panels can be achieved in this way, but the use of these arrows requires some practice. Watch the panel vector values change with each action, as does the schematic representation of the panels in the small left side plot window. Play around with these buttons to become familiar. If you get totally lost with your product, you can always start again from scratch by hitting the REFORM button (see Section Reformat under 4.3.5). The action of the buttons is interlinked in a logical way, and each set of buttons roughly perform the following (from top to bottom):

Position arrows: moves the panel up or down. Affects the first number in the panel vector.

Absolute size arrows: Increases or decreases the absolute size of the panel, leaving other panels unchanged. May leave gaps or overlap other panels.

Relative size arrows: Increases or decreases the size of the panel relative to all other panels.

Note: Overplots are taken into account by these actions. Any panels that exactly overlie each other are treated together. Warnings are issued if there is a conflict in the panel composer, e.g. more than one panel at the same position, only overplots at the same position, etc.
The only correct combination of plots at the same position is that all but one must be overplots!

To aid you in your panel design the graphic schematic display changes to red whenever plots start overlapping.

4.3.4 Panel schematic color logic

The graphic schematic display to the left of the panel entry list has been designed to guide and help the user in composing a panel plot. The color and fill-style of the box representing the panel is used to infrom the user of possible conflicts:

solid: Panel not an overplot.

outline: Panel is an overplot.

blue: Single panel with no overlapping or overplotting panels

cyan: Two or more panels OF THE SAME SIZE at this location (used for lining up overplots).

dark green: O.K. overplot arrangement at this location. All overlapping panels have the same size, and all but one are overplots!

red: Conflict! Not all panels are overplots, not all overlaps here are of the same size - this will possibly not create the panel plot the user intended!

4.3.5 Panel-editing

Once a product has been defined it can not only be manipulated as described above, but also edited. This is done by highlighting either of the two lines of text in the compose window associated with a given panel. To bring up that panel's panel editor window, the user can double click on the highlighted text. The other way of editing is by means of the buttons below the positioning buttons:

Edit: This button has three sub-choices, each of which affect only the highlighted panel in the panel compose window:

- **Edit Plot:** Also calls up the panels' panel editor window and is analogous to double clicking on the panel entry in the compose window.
- **Change type:** Allows you to select a different plot type altogether and calls up that plot types panel editor window.
- **Delete:** Removes the panel from the panel composer.

Reformat: Use this button to reformat all the panels in the compose window. This is useful after you have moved panels around and changed their heights and ended up with a "messy" compose window. There are four reformat options:

- **remove gaps:** Removes any empty gaps between panels. Overplots are preserved.
- **reset all:** Leaves all panels in the current order but resizes them all to height 1, with the top entry being the bottom panel. This will undo any overplots.
- **resort all:** Reorders all panels such that the bottom most panel ends up at the top of the list.
- **undo overlaps:** Any overlapping panels that are not overplots (i.e. they do not overlap exactly) are untangled, preserving the relative size of the panels. Overplots are preserved.

4.3.6 Overplotting

The remaining OVERPLOT button allows for an advanced use of PAPCO. When the highlighted panel is marked with this button the schematic representation changes to an outlined box, indicating that this plot will overplot an existing plot at the same position. Normally, when you add a panel to the composer, it is added as a new plot at the top of all the existing ones. So if you want that plot to overplot one of the previous ones already in the composer, user the arrow buttons to move that panel until it is at the required position. Panels that overlay are indicated by the color in the schematic changing to red.

An overplot adops the y-axis scaling from the plot BELOW it that isn't an overplot. Overplotting line plots onto color spectrograms is O.K, the other way around makes no sense as the plot below will be hidden. PAPCO does no checking here!

The module must be written to support overplotting, and this *should* be stated in the module's panel editor help file (if it doesn't support it the plots will simply be plotted one on top of another, which won't look very good...). Again, here PAPCO provides the functionality, but it is up to the user to provide the required support in his plot routines.

4.4 Printer defaults information

These two labels are set to the currently selected printer default output options. On the printer filename, ".ps" indicates PostScript, ".gif" indicates GIF. They are for information only, to change the default printer and/or print filename use the printer widget (see Section 4.2.6.

4.5 The bottom widget part: The Action Buttons

4.5.1 Draw

Once you have made your product, press this button and the plot is drawn, from the bottom panel up. If no Draw Window exist, a new one is created. Otherwise the plot is redrawn in the currently active Draw Window.

Any changes you make (new time range, more panels, edited panels, etc.) are only updated the next time you press Draw or Draw New.

However, any changes to the Mouse Action take immediate effect in the current Draw Window.

4.5.2 Draw New

Has the same effect as the Draw button, but here a new Draw window is opened. This window gets a new running number in the title bar and becomes the active Draw Window. Any older draw windows remain on screen for reference but are inactive.

NOTE: The last window drawn in or created is always the active draw window. It is also the window with the highest window number in the title.

This button is inactive when no Draw Windows exist. Use Draw to create the first one.

4.5.3 Print

Same action as Draw or Draw New, but graphics output is redirected to a printer or file, or data output to file, depending on the current setting of the Print Action.

If the Print Action is set to one of the Batch options, a file widget comes up first for you to choose which batch file is to be processed.

4.5.4 Annotate

Once a plot has been drawn this becomes active. IDL supplies it's own annotate tool and this has been slightly modified to work with PAPCO. This button calls up the annotate widget (Figure 4.3 which interactively allows you to annotate the currently active draw window. The normal slice functions are disabled while the annotate widget is in use.

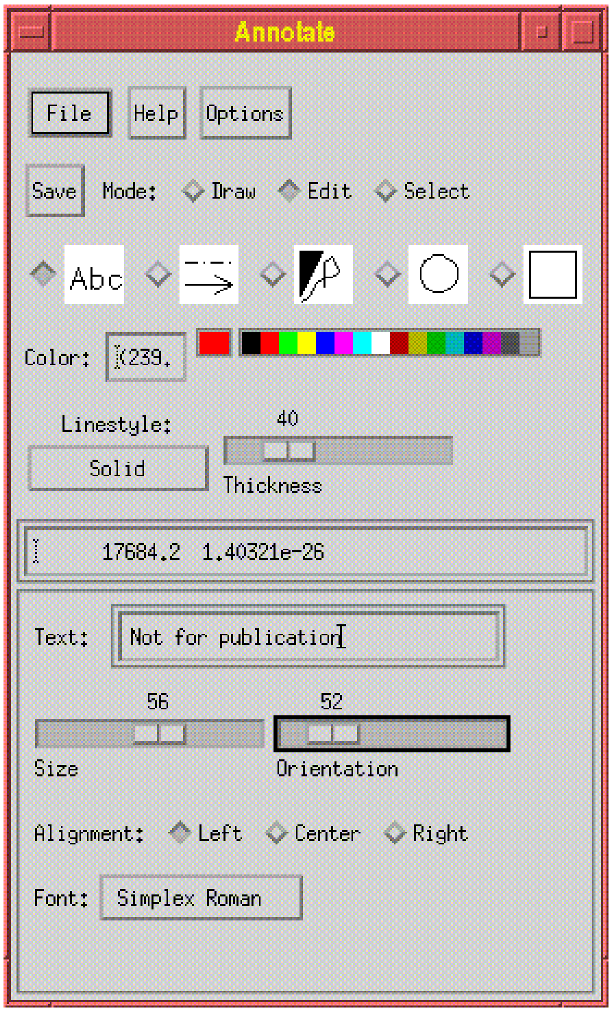


Figure 4.3: PaPCo annotation widget

This widget is basically the same as the IDL supplied version. It has been modified to use the PAPCO colors, and to save batch files into `papco_user_XX/papco_products/annotate` by default. To learn how to use it, use the widget's Help button. Here's a quick guide only:

- Any object you draw will only be "remembered" after you hit the Save button.
- Most objects can be dragged/resized interactively using the mouse in the draw window. Use the mouse to mark the initial position of an object in the draw window.
- Before you can edit an object already drawn, you need to hit select and mark it with the mouse.
- PAPCO will not "remember" any annotation you have made unless you save it to a file. Use the File button's menu choice "Save as". Make sure the file gets the extension ".dat".
- You can also load up any previously defined annotation file using "Load". Make sure you use "Clear" to clear the current annotation first.
- Don't use the "Write Postscript" and "Export Bitmap" options. PAPCO handles output of your annotation to file/printer internally. Those option will give you a copy of the annotation only.
- To finish annotating, save your annotaion to file and then hit "Exit" from the File menu.

The annotation file name becomes part of the current product and is saved along with it.

A lot of the draw/redraw activity that PAPCO needs to perform to overlay the annotation onto the current draw window depends on being able to save the bitmap of the current draw window. This is automatically done at the end of each draw loop, but this feature might not be supported on all window systems, AND it also depends on enough memory being available!

NOTE: Use this facility for SIMPLE annotation only. Mor complicated annotation should be done by using a professional package such as xFiG, and importing PAPCO output into it.

4.5.5 Output

Once a plot has been drawn this becomes active. Redraws the plot but also outputs the data to file. This functionality must be supported in the user's module, see Section C.3. It is supported by the PAPCO template `plot_type`.

4.5.6 Rm. Annot.

Once you have annotated your plot, this button becomes active. It deletes the annotation from your plot.

4.5.7 Reset

Clears the active graphics window and the panel compose list. Keeps the currently set time range.

4.5.8 Back

Load up the PREVIOUSLY drawn product. PAPCO keeps a history of all drawn products from a given session. They are all deleted upon normal exit from PAPCO.

4.5.9 Defaults

This brings up the PAPCO defaults window in which the user can set various default formating options (see Figure 4.4). Any change in the default settings will only take effect the next time the Draw button is hit, for changes to the draw window size to take effect you need to hit Draw New.

Draw character thickness: Sets the thickness of the vector drawn fonts used to annotate the plots.

Draw character size: Sets the size of the vector drawn fonts used to annotate the plots.

PostScript charsize factor: Allows additional scaling of the PostScript character size

Non encapsulated postscript: Either enables or disables non-encapsulated postscript output.

The draw area scaling: Allows the user to set a fraction of the draw window for output while keeping all the annotation size as if it was for a full draw window (they don't get scaled). This is useful for making camera ready plots of any size while keeping the annotation readable!

Draw PAPCO **Ident string:** If yes then a PAPCO identification string is plotted on the far right of your plot, with PAPCO version, date & time and product name information. Can be turned off here for publication output.

US or Europe: Set the default date time output used, and also selects A4 or letter size print output.

Screen output draw window size: Draw Window size can be given in pixel sizes and customized for your display.

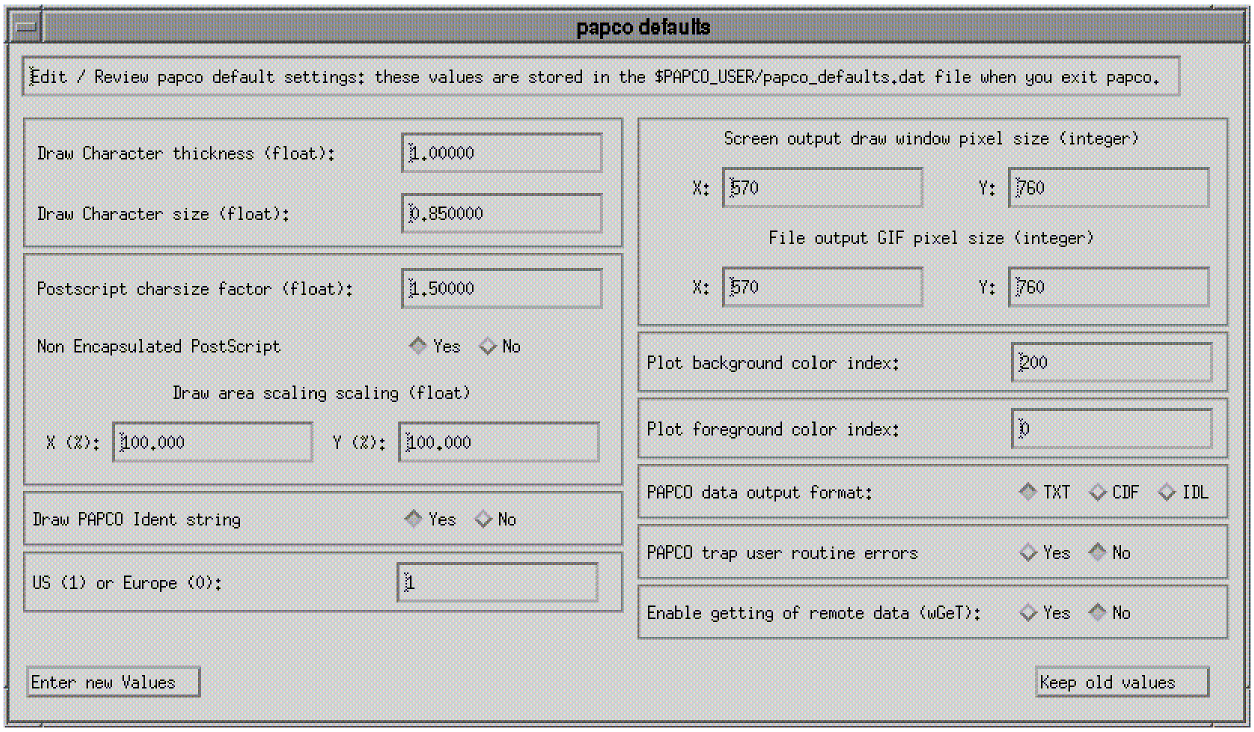


Figure 4.4: PaPCo defaults widget

File output GIF pixel size: GIF output size can be controlled here in pixel size.

Plot background color index: selects the color index for the background grey scale to be used - the values range from 0 to 255, 0 for black, 255 for white.

Plot foreground color index: The drawing color can similarly be chosen - the values range from 0 to 255, 0 for black, 255 for white.

PAPCO **Data output format:** For those modules that use the PAPCO supplied output functionality, the output file format is set here.

PAPCO **trap user errors:** Any errors occurring in user written interface and/or plot/read routines can be trapped here by PAPCO at the point they are being called. This avoids PAPCO crashing when an error in a module is encountered. turn this off while designing your own modules, in order to be able to debug your module (no error traceback beyond PAPCO is possible if PAPCO traps the error).

Enable getting of remote data: Enables or disables the remote data get facility. When enabled, PAPCO checks for the existence of wGeT on your system.

The user can modify any of the entry fields: new values are read when the Enter New Values button is pressed - you do not need to press "enter" in each modified field.

4.5.10 Font

This button calls up the IDL-supplied widget for choosing a font. The fonts you have available depend on what is installed for your X-window manager. The font you chose here is used for the labeling of all widget text in PAPCO. For the new font to take effect you have to exit PAPCO and restart.

This option is supplied for user customization. At times the PAPCO default font will be too large for the size of your screen, and the whole widget window will not fit. Choose a font of your liking for optimum size for your hardware and for best readability.

4.5.11 Help

Brings up the general help screen for the PAPCO main window.

4.5.12 About

Brings up a brief information window about PAPCO . This is mainly to avoid anyone else copyrighting the package and making us pay for it!

4.5.13 Edit Batch

Brings up a simple widget to choose and edit a PAPCO batch file. The batch files are searched for in `papco_user/papco_products/batch`.

4.5.14 Load

Loads a previously stored product file. The Load clears the current product and displays the loaded product in the compose window. Draw then displays the plot. The product is loaded together with the time range at the time of saving.

4.5.15 Save

Saves the current product as defined in the Panel Compose Area and time range to a user-selected file. Default directory is `HOME/PAPCO_products`. The saved product is also tagged with the version of PAPCO that made it. While every effort is made to ensure compatibility of these products across version changes, there is no guarantee of compatibility - we're not that professional yet. Also, if you load a product which includes panels from modules which you do not have, PAPCO will allow you to do that: Delete the unknown panel before plotting, otherwise PAPCO will crash (we hope to fix such problems in the future).

4.5.16 Save & Done

Exits PAPCO, after asking for confirmation, and saves all currently compiled procedures and functions in a file `papco_routines.save` in the user's `PAPCO_USER_XX` directory. This file is used for quick startup of PAPCO using the "Restore" feature (3.3.4).

4.5.17 Done

Exits PAPCO, after asking for confirmation. At exit, all Draw Windows are destroyed, all temporary postscript plot files are deleted and the current state of PAPCO is stored in a defaults file (stores pre-set paths for products, print files, printer options, see Section 4.5.9).

4.6 IDL window output

Although most of the work with PAPCO is done through the main level widget, it is useful to keep an eye on the output generated in the IDL command window, as most routines in PAPCO report status messages to let the user know what goes on. Its also here that you see if PAPCO has crashed; the traceback messages produced by IDL are very useful in debugging.

Normally, any crash that occurs should be traceable to a user module. If you really think that a crash is due to a fault in PAPCO core routines, then copy the traceback messages on your screen to an e-mail message to that author of this document and cry for help!

Most status reports from PAPCO start with `% name_of_routine:` and should fit onto one line in an 80-column window. This includes messages from PAPCO core routines AND user modules, so please adhere to this convention in your own modules! These status reports are simply for the information of the user, you should never be prompted to input anything on the command line - all input functions are done through the main widget and its sub-widgets.

Here is a sample output from a PAPCO drawing loop drawing a five panel plot. Here we plot the data for the `plot_type` module demonstrating some of the built-in PAPCO features such as simultaneous different color scales, line-style options and the standard `papco_text` module. Shown also is an overlay-annotation "Not for publication", constructed with the annotate widget (Section 4.5.4). The graphics output of this loop is shown in Figure 4.5.

```
****************************************************************
% papco_draw: Start of plotting loop.
% papco_files: Saving
  /n/toaster/u/friedel/mypapco/papco_products/state_00003.papco
% papco_draw: Clearing panel [       0,       2]
% papco_draw: Clearing panel [       2,       2]
% papco_draw: Clearing panel [       4,       2]
% papco_draw: Clearing panel [       6,       2]
% papco_draw: Clearing panel [       8,       1]

----------------------------------------------------------------
% papco_draw: Doing Plot No. 00, plot_type
              panelVector: (00,09,02)    orbit: 0000

% papco_autorange: Doing Logarithmic scaling, clipping 0 %
% papco_autorange: Data less than zero: 0 out of 8980
      96.1950    0.0566621      20    0.0500000
         384          897         143
% papco_plot_colorspec: time-averaging 897 time intervals down to 384.
% papco_hpc_av: smoothing allowed.

----------------------------------------------------------------
% papco_draw: Doing Plot No. 01, plot_type
              panelVector: (02,09,02)    orbit: 0000

% papco_autorange: Doing Logarithmic scaling, clipping 0 %
% papco_autorange: Data less than zero: 0 out of 8980
      96.1950    0.0566621      20    0.0500000
         384          897         143
% papco_plot_colorspec: time-averaging 897 time intervals down to 384.
% papco_hpc_av: smoothing allowed.

----------------------------------------------------------------
% papco_draw: Doing Plot No. 02, plot_type
              panelVector: (04,09,02)    orbit: 0000


----------------------------------------------------------------
% papco_draw: Doing Plot No. 03, plot_type
              panelVector: (06,09,02)    orbit: 0000


----------------------------------------------------------------
% papco_draw: Doing Plot No. 04, papco_text
              panelVector: (08,09,01)    orbit: 0000

       0           0           2           1
----------------------------------------------------------------

% papco_draw: End of plotting loop.

% papco_draw: saving draw window image bitmap...
% papco_draw: Drawing annotation
% papco_annotate: using
  /n/toaster/u/friedel/mypapco/papco_products/annotate/test.dat
****************************************************************
```

With the start of the plotting loop, the current state is saved first and then all the panels that have to be drawn/redrawn are erased.

For each panel that is drawn `papco_draw` outputs the Plot number, the name of the module, the panel vector and orbit number (0000 if not set). The messages from `papco_autorange` and `papco_plot_colorspec` are from the color plotting routines which come with the PAPCO distribution in `papco_XX/papco_lib`.

At the end of the drawing loop there are messages informing the user that the bitmap of the plot has been saved (used by some of ther slice functions), any annotation files that are used, and to which file the plot was written or to which printer queue it was sent. If the plot is to screen, there is no message.

The output plot from this drawing loop is shown in Figure 4.5, the product drawn is the one shown in the panel composer of the main PAPCO widget in 4.1.

4.7 Draw Window

When you first call up PAPCO you only get the main widget and no Draw Window: this is created one you have composed a plot, selected a time range and pressed the Draw button. From now on you have the choice of either pressing Draw, which will redraw the plot into the current draw window, or press Draw New, which will keep the existing Draw Window, deactivate it, and open a new one, which then becomes the current draw window. This is useful for comparing plots on-screen.

NOTE: Any mouse action or time selection or printing command only affects the active (last) draw window!

In the title of the Draw Window is a running number: the highest number is always the one of the active Draw Window. An example of a draw window is shown in Figure 4.5.

Below the plot area are two information labels which contain the current Product file name and Annotation file name, if set.

The Action Buttons on the Draw Window are described below. This information can also be accessed through the active Draw Window's Help button.

4.7.1 Draw, Print, Output, Back

Same as the corresponding Action Buttons Button in the PAPCO main window. See Section 4.5.

4.7.2 Help

Displays this help window for this Draw Window.

4.7.3 Delete old Draw Windows

Destroys all inactive PAPCO Draw Windows.

4.7.4 Close all Draw Windows

Closes all the PAPCO Draw-windows. A new Draw Window is created again when the Draw button is next pressed in the PAPCO main window.

4.7.5 Cancel? (entry field)

If you wish to interrupt the current draw cycle change the Cancel entry field to anything other then “N” by moving the mouse to the field and editing it. The PAPCO draw loop will then terminate AFTER the current panel has been drawn.

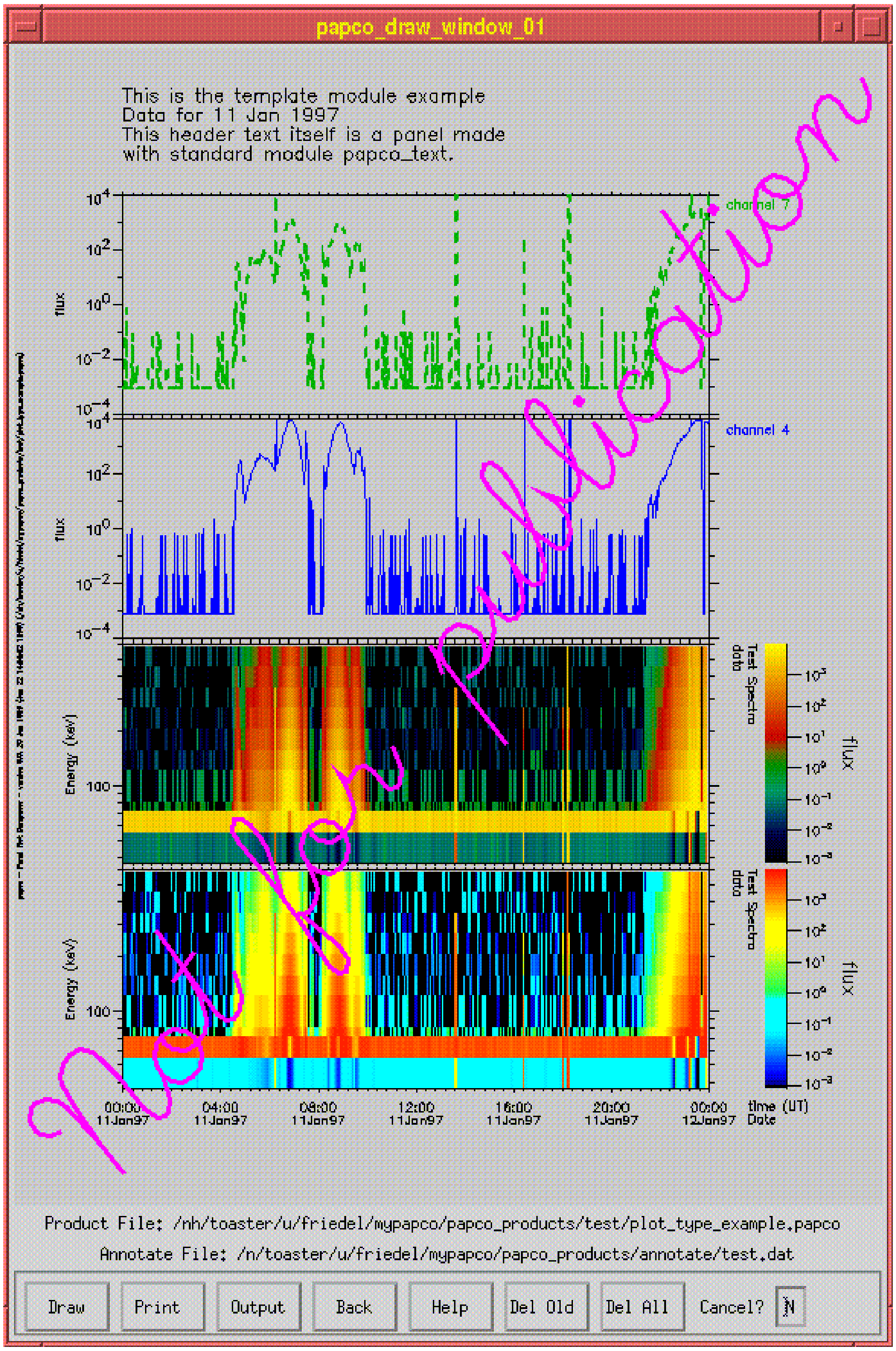


Figure 4.5: PaPCo output Draw Window for the drawing loop described.

4.8 Command-line PaPCo

Once you have some product files and/or batch processing files defined, you can use some of the command-line keyword options of PaPCo . These options are there for ease of use and speeding up things, and to enable batch processing in the background.

Note: You still need to have access to a X-Windows display from your session. Even though no widgets are displayed, they are created but never mapped.

4.8.1 Keyword PROD

If you know the product file name you want to start PAPCO with, type

`papco, PROD='product_filename'`

at the IDL prompt once PAPCO has been compiled. PAPCO will start up and automatically load that product file. It is searched for in the last product file directory that PAPCO used (which is saved in the defaults file).

You are left inside an interactive PAPCO session.

4.8.2 Keyword DRAW

If the keyword PROD is also used then the product is not only loaded, but also drawn automatically.

You are left inside an interactive PAPCO session.

4.8.3 Keyword BATCH

If you know both the product name and the batch file name, you can run PAPCO "quietly" to process a batch of files without entering into an interactive session. Type

`papco, PROD='product_filename', BATCH='batch_filename'`

PAPCO will run without opening any X-windows but still provide diagnostic output in the IDL window. It will start up, load the product and then process it for the times defined in the batch file. It outputs postscript or gif files using the last set filename template. So, before running a batch make sure that you've set the output options you want from the printer window interactively.

Note: You still need to have access to a Windows display from your session. Even though no windows will be opened, PAPCO internally still creates widgets but does not realize them. For this IDL requires access to a windows device. In other words, you can't do this from a dumb terminal...

You never enter an interactive PAPCO session.

4.8.4 Keyword OUTPUT

Used in conjunction with BATCH to also output to file if the modules used support that option.

4.8.5 Keyword SET_ONLY

Used to set up all the PAPCO common blocks and module environmental variables. Useful if you want to work in the "PAPCO environment" and have access to all the PAPCO routines and module routines for other purposes than running PAPCO .

4.9 Recovering from a crash

PAPCO core by bow is very stable and *should not crash.* PAPCO attempts to trap all errors that occur in user-writen routines (modules), using the IDL `on_error` facility. Such errors are reported by PAPCO, in the IDL transcript window and/or in the draw window in place of the panel it was busy producing. PAPCO will not crash in those cases, but the part of the module you were attempting to use will not be useable until the problem is fixed. You might need to contact the module athor to help you with that, or, if you manage to fix the problem yourself, do inform the author so that he may upgrade his module for the next release.

To produce a real crash, you need to turn off the PAPCO error trapping. This can be done with the defaults widget (4.5.9).

The crash recovery "tricks" in this section are for advanced IDL users. They are provided for developers who might encounter a lot of crashes while writing a a new module, but they are also useful for the normal user in unlikely case that the PAPCO error trapping fails.

Normally, when a program crashes you need to start again from scratch. In IDL there are ways of recovering from a crash and to continue from the point where you left off, this can substantially speed up the program development loop "edit – try out – crash – edit – try out OK!" as you don ot have to re-start PAPCO each time!

Depending on your IDL version, you can recover from a crash in the following ways:

IDL 4.x: After the error message you are back at the IDL prompt and the PAPCO widget is still up. You can try to use the IDL commands `.con` to continue, `.skip` to skip over the error producing line, or `return` to return from the crashed routine. If you crashed in your plot routine, use the PAPCO batch file `@papco_ret_plot` to continue. If you managed to continue, use the STOP button to halt PAPCO and do any edits / compiles needed, then continue.

IDL 5.x: From this version of IDL onwards any errors produced by widget programs are trapped by IDL: you get an error message, but the widget manager does not crash. Now simply close any draw windows by hitting the Dell All button in the last active draw window; this will re=activate the PAPCO main widget. Now use the STOP button to halt PAPCO and do any edits / compiles needed, then continue.

However, if get stuck or get the IDL message "can't continue from this point", then PAPCO has to be restarted (close any PAPCO windows manually first):

`retall` (use the PAPCO routine `resetx` under IDL 4.x)
`papco`

At times PAPCO may get into an unrecoverable mode and will produce strange graphics, even after `retall`. In that case exit IDL and re-start IDL and PAPCO from scratch.

Whenever you had an ungacefull exit from PAPCO, PAPCO will start up next time with the last product you were working on.

Chapter 5

Behind the scenes

Here some of the underlying conventions are explained in more detail for the user who wants to write his own modules; but the section is also useful for the normal user to understand how PAPCO does things.

No great detail of how to write your own add-in module is given here, this is covered in the section 6. Instead, we present here the general rationale that all the current modules of PAPCO adhere to, to give the user an idea of what he is letting himself in for when he attempts to add his own module.

Remember that PAPCO prescribes very little. However, if you want to produce nice output that works well in concert with other modules, then it makes some sense to adhere to some common philosophy.

If reading this section discourages you: don't be. The actual amount of work to add a new model is quite small - doing it the first time will probably take you a day. Eventually new modules can be added within an hour - although it might take longer if you want to make use of all the functionality of PAPCO.

5.1 Architectural transparency

IDL is available on most popular platforms in use today. A great amount of effort has been invested to make use of this inherent portability of IDL to have PAPCO running on a variety of platforms (see section 1.4).

IDL proper runs an many systems without any problems. The limitations in PAPCO come from all the system-dependent I/O that takes place, the use of environmental variables and system spawn calls.

The core of PAPCO has been revised now to replace all system dependent calls with "new" routines. For example, the `setnenv` call, which sets environmental variables, has been replaced by a routine `papco_setenv`, which resolves any system dependence. All the routines that had to be written in this manner are collected in the `papco_opsys.pro` file under `papco_XX/papco` and are documented there. The routines in `papco_opsys.pro` now support UNIX, VMS and Windows 95.

Note: So far, only PAPCO core has this portability implemented. All existing modules still need to be updated to make use of the support provided in `papco_opsys.pro`. All NEW modules written with this version of PAPCO should have module portability implemented - however, this is the responsibility of the module author.

5.2 Panel editor philosophy

The Panel Editor is the user-written widget which is basically an input-interface to his/her plotting routine plus more. Most normal plotting routines take some input parameters which control the behavior of the plot or select a certain subset of data. Each panel editor widget for PAPCO is a combination of some provided PAPCO-widgets and a range of user-defined buttons, sliders or whatever else the user might need.

Whenever a panel editor is called, it is passed a structure called `papco_plotinfostruct`. This contains standard tag names for PAPCO support and also tag names which are used by the user to pass his own options.

The panel editor function modifies this structure and returns it to PAPCO, which maintains an array of these structures for each panel. It is up to the user to decide which of the tags to use in the call to his plot routine; and whether he wants to support the PAPCO options of controlling color and line style of his plots, the y- and z-limits and the overplot facility. The idea again is simple: PAPCO provides the functionality, it is up to the user to write modules that support this functionality. But the choice is his/hers - you do not NEED to support all of PAPCO functionality.

5.3 Plotting philosophy

PAPCO uses a PANEL VECTOR to control the positioning of a specific plot panel. To do this, each user-written plotting routine must include the call

`setpanel(panel)`

where `panel` is the PANEL VECTOR as described in Section 4.3.2. This call simply sets the viewport for the user's plot using the IDL system variable !P.POSITION.
`panel` is one of the variables that is passed to the user's plot routine by the PAPCO.

The information contained in this vector can be used to further customize the user's plot. For example, the user can subdivide this viewport further to plot individual images, or decrease the plot height to add auxiliary x-axes.

The `plot_type` module template included in PAPCO and all the modules currently included in PAPCO adhere to these simple rules:

1. If a plot is the first panel (`panel(0)=0`), then labels for the x-axis have to be plotted. At minimum this would be a time-axis, but auxiliary information such as ephemeris information might be useful to add here.

2. Logically, if the plot is not the first one at the bottom, the plotting of an x-axis should be inhibited. However, it is up to the user to further reduce the plotting window size if he needs to plot some other axis, such as secondary ephemeris information when comparing data from two spacecraft. An example is the Los Alamos module, where the local time for each satellite is plotted together with the data even when the Los Alamos panel is not the bottom one. This is all user-defined. The x-axis titles should not be below the axis, as there might be several levels of labeling, but rather to the right of the axis.

3. The y-axis labels are generally on the left hand side and represent the physical units of the plot. The width of this label and the number of tick marks should be adjusted depending on the height of the panel compared to the total height in the plot window.

4. A short description of the plot should be placed on the right hand side of the plot, identifying the data source and product.

5. For the special case that the plot is the only panel, there should be an option for adding a header for the plot. Existing plot modules for PAPCO use the panel vector to decide if they're the top panel and then always plot a header containing at least the date for the data. Newer modules don't do this, since the text utility module (see Section 7.2) can be used to add a top panel with Title text in a more controlled way.

6. By default, all plots have the capability of auto-scaling the range of the data they plot. These user-defined routines are of varying sophistication and up to the user to supply, or he can simply hard-wire fixed limits for his plot. PAPCO supplies a few scaling routines such as `papco_autorange.pro` for color spectrograms and `ylim.pro` for simple line plots. However, any simple scaling routine will fail to yield good plotting limits at times, and for this purpose PAPCO offers the possibility to override the automatic y-limits provided by the user. For this purpose each plotting routine needs to include

a common block `common yscale, yscl` where yscl is an array of four-element vectors. There are as many array entries as there are panels, and each element has the following meaning (where * refers to the panel number):

yscl(*,0) is set to 0 or 1 indicating automatic or manual y-scaling

yscl(*,1) contains the manually set lower y-limit, or the automatically set lower y-limit depending on yscl(*,0)

yscl(*,2) contains the manually set upper y-limit, or the automatically set upper y-limit depending on yscl(*,0)

yscl(*3) is set to 0 for linear and 1 for log axis scaling.

This common block is also used by the overplot facility to know which y-scale needs to be adopted by the overplot.
In an analogous manner there is a zscale common block for scaling the third dimension of a plot.

5.3.1 Time arrays

With the old T90 format, all time was a continuous running number of seconds since Jan 1. 1990. Apart from being limiting (nothing before 1990 could be added), the plotting routines also ran into serious problem plotting in T90: the IDL plot routine internally scales everything to a float, so as T90 numbers got big rounding problems appeared. Also, converting to T90 could be time-intensive.

Consequently, a considerable amount of effort has been invested in adopting a new time standard for PAPCO: MJDT (modified Julian date plus seconds since midnight of that day). The PAPCO zooming function is now independent of whatever time axis a module actually uses. The following convention has been adopted for all new modules (the old ones using T90 are still compatible, too):

All data passed to the plotting routine has it's time array reduced to the number of seconds since the start of the start day of the plot. The existing PAPCO module template adhere to this standard, and there is some auxiliary support for its use (like functions for making time labels of various types).

5.4 PAPCO color scheme

One of the requirements that arose through the use of PAPCO was the ability to have several active color bars, to display color spectra with different color bars together on the same plot. This is not possible with the standard IDL color scheme, which has one color table of 255 colors; changing it changes the colors globally for ALL panels.

This restriction does NOT apply when you write a PostScript file, as it is a sequential file. Each time you change to a new color table this gets loaded into the PostScript file, and you can have as many color table changes as you like.

PAPCO now subdivides the IDL color table to provide up to four simultaneous color tables at the cost of lowering the number of colors available to each table - now near 50 - but this has not significantly affected the quality of the display. The scheme used is shown in Table 5.1 to 5.3.

Index	0	1
Use	background	foreground

Table 5.1: Color index assignments for drawing (foreground) and background colors.

PAPCO internal color table scheme: If nc_total are the total number of colors available to IDL, then the number available for the subtables is nc_avail=nc_total-16-2, and nc is the number of colors per subtable: nc=nc_avail/4.

Index	$2 \to 2+nc$	$2+nc \to 2+2\times nc$	$2+2\times nc \to 2+3\times nc$	$2+3\times nc \to 2+4\times nc$
Use	subtable 1	subtable 2	subtable 3	subtable 4

Table 5.2: Color subtable index assignment.

Index	$2+4\times nc \to 2+4\times n+16$
Use	Fixed primary colors for line plots

Table 5.3: Fixed prime color index assignment.

The last entry in the color table is at an index equal to !D.N_colors, the IDL system variable which contains the total number of available colors. This number is usually less than 255, since your X-window system needs some colors. PAPCO checks the number of available colors at startup: If there are too few it will exit with an error message. You then need to close other applications you have running that are "color hungry" such as NetScape, another session of IDL, etc.

The color scheme and the utility routines which allow the user to choose the color subtable and to define his own color table into one of the subtable slots is explained in more detail in the `papco_color.pro` file in the `papco_XX\PAPCO` directory. `papco_color.pro` supplies a complete set of routines to manipulate this color scheme - you can replace any of the pre-loaded color tables with your own, interrogate which subtables have been used, etc.

PAPCO by default selects the subtable 1 (GSFC rainbow). If your panel editor included the color table choice widgets (see Section 5.2) then the index of the chosen color table can be passed to your routine through the keyword SUBTABLE (see Section B.2.3).

The first color subtable (GSFC rainbow) is the papco default color table. You need to choose a different color table or load a different color table of your own in your own plot routine. Remember that your color plot routine must be able to handle the PAPCO color scheme. The color plot routine supplied with PAPCO, plot_clr.pro (in `papco_XX\papco_lib\color_plot`) does this.

PAPCO supplies a complete set of routines to manipulate this color scheme.

5.5 Slicing philosophy

Writing slices for your modules is considered advanced PAPCO. This should be attempted only once the general system is well understood.

The basic idea here is to provide extra functionality for the mouse. PAPCO provides all the required routines which will return from the mouse position the panel, the corresponding x- and y-coordinates in that panel, and to hand over this information to a user-written routine. What the user does with this information is up to him/her. Normally, this information is used to re-plot data at that time in a different form, in a separate window. Applications of this may range from simply returning the data values at the mouse position, to calling up a level-zero analysis tool for that time - e.g. for HYDRA, pointing the mouse to a plot of moments data can call up the corresponding level-zero tool to give detailed distribution functions at that time. Or to give POLAR aurora images at that time. Or anything else the user decides.

PAPCO provides some default slices which either work as they are or need a simple user-written interface routine. On top of that the user can configure PAPCO with completely self-written mouse functions.

5.6 Read-routine philosophy

Traditionally files have been kept in a variety of time formats: Per orbit, per day, per month - whatever. Originally PAPCO was written with CRRES in mind, where everything went by orbit. In fact, that is still the only way you can access CRRES data - and for CRRES, it makes little sense to plot the data any other way.

For ISTP the decision has been made to have data in files per day. Consequently, there is the need for a generic read-routine that will take as input either a time range or an orbit number; this routine will then internally and automatically select and concatenate the required files and hand back a continuous data array of the required length.

PAPCO provides the time range required in the form of a common block in MJDT format (modified Julian Date plus seconds since start of that day, a structure), containing end and start time.

`common mjdt, mjdt_start, mjdt_end`

There are conversion routines in `papco_XX\papco_lib\convert.pro` to convert this to other time formats.

All that PAPCO will provide is this common block, and the orbit number which is passed by parameter. It's up to the user to provide a read-routine that can handle this input.

5.6.1 Making things configurable

Your read routine needs to know where the data is - in which directory, or even at which site. PAPCO provides some functionality for this. In each module there is a `defaults.config` file that contains a list of environmental variables and their default settings. These are used to contain the data paths needed by the read routine, and can be interactively modified through the module's panel editor (see Section 6.4.1). So it makes sense not to hard-code any paths that are site-dependent, but rather to use environmental variables.

5.6.2 Speed of reading data

Further, it is desirable to have data in a format which can be read as quickly as possible (this problem has already been discussed in Section 1.6.1).

5.6.3 Example: Reading Los Alamos geostationary data

A good example of how this problem is solved in practice, is the use of Los Alamos geostationary data (courtesy of Dick Belian and Geoff Reeves) which was used as part of a joint study. Here CRRES data was being plotted by orbit, and the need arose to have Los Alamos data for the corresponding period. Los Alamos data is supplied as ASCII files per day, which are then further compressed using gzip to save disk space.

The following procedure was adopted and implemented in code. Zipped raw data was kept in mass storage, and IDL-binary data for fast-read in a local directory. When the routine was called to return data for a given time period the following actions were performed:

1. Use the time information to construct the file names of those files covering the interval.
2. Check if those files are available as IDL-binaries. If not, check if they are available as ASCII files. If they are still zipped, unzip them. Read those files using a slow ASCII read-routine which at the same time makes a carbon copy as an IDL-binary (which in this case is a lot smaller than the original zipped ASCII).
3. Once all required files are available as IDL-binaries, read them in succession and concatenate one data array containing the required time period of requested data.
4. Return this array to PAPCO through a common block.

Using this procedure, only wrapper routines (using existing ASCII read-routines) had to be written for the Los Alamos data to be integrated into PAPCO. The whole data compatibility problem is solved by the simple method of reading slow once only, and then fast, using IDL-binaries. This does produce an overhead and needs extra disk space. In practice, we batch-process a given data set to produce the required IDL-binaries and archive the original data.

The procedure described here, however, is not prescribed. The user may do things any which way he/she wants. The current scenario has been adopted to maximize the speed of reading in large data sets - but any read-routine which presents to the corresponding plotting routine data for the requested time interval will do. Since both are user-written, PAPCO has no place in prescribing how things are done. All we can do is show how things have been done in PAPCO history and suggest reasons for why this might have been advantageous.

5.6.4 Data portability

With PAPCO use becoming more widespread, modules need to be written with portability to other platforms / architectures in mind.

One way of doing this is to use a data format which is portable - such as CDF or IDL savesets. At least the data which are made public through a PAPCO module should be portable - or the read routine in the module should be able to read the data no matter from which platform or architecture.

This is particularly important if the module makes uses of the remote get data facility, which should be encouraged.

5.6.5 Getting data remotely

As a further extension PAPCO now provides a routine to "fetch" data from a remote site using the GNU freeware program "wget". You can now write your read routine in such a way that if the data is not found locally, it is copied via ftp from a remote site (see Section C.2)!

This feature is under development and has so far only been implemented under UNIX. The GNU wGet facility was chosen as a "vehicle" because it's free, and available for most platforms. It has so far not been tested under VMS or Windows 95.

PAPCO provides a set of routines in `papco_get_remote_data.pro` under `papco_XX/papco` to interface with the wGet program and to provide status reports to PAPCO of data being downloaded. This interface depends on interrogating the wget log files produced. The version of wGet used is: `GNU Wget/1.4.5 by Hrvoje Niksic <hniksic@srce.hr>`

As this is considered to be an extremely powerful feature of PAPCO further development in this area will take place with the aim of full portability.

Instead of obtaining data via an ftp utility such as wGet the preferred way would be to provide remote mount points for data on your system, so that a remote site "looks" like a normal directory path which you can configure your module to use.

5.7 Caveat on system resources

The current PAPCO philosophy is to keep all data for all panels in memory so that they can be plotted and manipulated together fast. The idea here is to encourage the user to use PAPCO for analysis too, where speed is required for zooming, slicing etc. - PAPCO for this reason provides to the user a variable called IsRead, which is set whenever the data for a given module is in memory, to let the part of the user-module that does the plotting know that the data does not have to be read again.

This does, however, assume that memory is not a problem. When plotting large arrays of data for long time periods for many panels, the amount of memory required might be very large. PAPCO makes no attempt at optimizing memory usage: The basic idea is that memory is cheap. PAPCO has been developed on a SUN SPARC 20 Station with 96 Mbyte of memory and no serious slow down due to swapping has been encountered so far. In practice, it makes sense to close down other applications for a serious PAPCO session, and to make sure that the swap space is adequate (we use swap=memory $\times$ 4).

Eventually, however, it makes no sense in returning 10 days worth of 6sec resolution data, since no screen can plot this in a way which allows one to see all that resolution. It is again the responsibility of the user to write "clever" read-routines, which automatically reduce the resolution of the data for long time intervals.

Chapter 6

Including your own modules

The main thing to keep in mind if you are trying to add your own module is, DON'T PANIC. The task may look daunting but the actual work involved is minimal, although the first time you try it it might take some time. We have attempted to lead you through the process by means of a series of templates, which allow you to check your process at every stage. These templates are included here in the Appendix B for your reference: They contain all the documentation you would need to use them. That's why they're so "big"; they actually contain very little code.

To use this step-by-step guide successfully, follow all the steps and read all the information included in the template files themselves. The step-by-step guide tells you what you need to do, and the information in the templates tell you how to do it. Test your modules at every step of the way, as described: in this way it's easier to track down any problems.

The templates have been written in such a way that all you need to do is to copy them and do a global rename - and there is even a utility that does that for you. If you have stuck to the naming convention then you should be able to test out the default behavior straight away. You should make sure that the PAPCO interface part of the routines works fine before you actually start programming your own plots or slices.

The module template supplied with PAPCO is actually a fully functioning module, the module "`plot_type`". This is a working module using a sample data file for Jan 11, 1997. This data file is not part of the PAPCO distribution, as the `plot_type` module supports the remote data get facility. If you don't have wGet installed, you can get the file manually via ftp. Check the `plot_type` modules config for the location of the file. Try it out, it supports all the basic PAPCO functionality. Using this module as a model it should be fairly easy for you to get a module of your own up and running quickly. The `plot_type` module can be found under `papco_XX/papco_modules/plot_type`.

Remember: DON'T PANIC. (Take your towel along.)

(That last statement is lost on you if you haven't read Douglas Adams' "The Hitchhikers Guide to the Universe").

The guide included here is a step-by-step guide for the adding of one new `plot_type` which has one data_product to plot. The best way to learn how PAPCO works is to go through this thoroughly once, which involves a fair amount of reading. But then the second module you add is a piece of cake.

I'm assuming that you already have an IDL read-routine (as described in 5.6) to put your data into common blocks, and that you know what type of plot you want to produce, and what range of input parameters your plot routine requires. Then follow the procedure described in the next sections, IN THAT ORDER.

NB: Make sure that all your filenames only user lower case letters. This makes portability a lot easier, as not all operating systems differentiate between lower and upper case letters.
This restriction does not apply to the actual IDL procedure/function names used.
Also, DO NOT use any special characters in your filenames AND your data file names, such as a period, dollar or percent sign, etc as this produces big problems with portability. Long

filenames are O.K. - unless you plan to run on older WINDOWS versions (pre Win95).

6.1 Getting into the PaPCo environment

All the tasks outlined in this chapter require you to be within an active IDL session, with all the PaPCo environment set up. In this way all the PaPCo procedures, functions and environmental variable settings are available to the module developer.

Start up your IDL session as if you were going to use PaPCo. Type

`idl papco_setup` or

`idl papco_startup` (see Section 3.6)

Now instead of going into PaPCo stay at the IDL prompt. You are now in the PaPCo environment.

6.2 make_new_module.pro

To simplify the creating of a new module a routine has been provided to automatically make the required module directory structure, copy all the template files from the `plot_type` module,and then to globally rename all the occurrences of "`plot_type`" to your new module name - in the directory and file names, and in the files themselves. The header of this procedure is:

`pro make_new_module, name, MODULE_PATH=MODULE_PATH`

and it is located in `papco_XX/papco_lib`.

name - a string, the name of your module. PaPCo convention has been to choose names less than 10 characters long, as this name become part of the name of many procedures and functions in your module. The name should indicate mission and instrument, such as `polar_mfe`, `\lanl_gps`, `\crres_meb` etc.

MODULE_PATH - a keyword, if set it contains the string of the path where new new module will be written. If you use the special string "own" then the module will be written into your `papco_user_XX/papco_user_modules` directory (this is the preferred place for new user modules). If not set, the new module is written into the standard module directory of PaPCo: `papco_XX/papco_modules`.

Using this procedure greatly simplifies the writing of a module, as most of the editing and copying tasks are performed automatically. In this way adherence to the PaPCo naming conventions for the module is also guaranteed. USE OF THIS ROUTINE IS STRONGLY ADVISED.

N.B.: This procedure makes use of a UNIX system call `sed` *and is not portable to other operating systems at this stage. An IDL only version of this routine is being planned.*

Nothing of course prevents you from doing this manually too! In the description that follows, you can omit each step 1 if you have used the automatic `make_new_module` routine.

6.3 Module directory structure

Your module can reside anywhere you want (see Section 3.3). Traditionally, all the PaPCo distribution module directories are subdirectories under `papco_XX/papco_modules`, but this is not prescribed. The best place for you module while developing is in your `papco_user_XX` space under `papco_user_XX/papco_user_modules`. Your module directory MUST, however, contain the following:

1. A subdirectory called `/papco_interface`. Place all the PaPCo specific interface files into this directory; that's panel editors, papcoadd routines and slices.

2. A subdirectory called `/plot_read`. Put all your newly written plot and read-routines here. This directory also needs to contain a file `defaults.config`, which is used to define the environmental variables used by your module.

3. An initialization file `init_module.pro`. This contains a series of `.run` commands to initialize your PAPCO interface, plot and read programs.

This structure is critical for a successful interface into PAPCO. To make life easier, the sample template module is already constructed using this file structure, and it contains all other templates already in their correct directories:
`papco_XX/papco/papco_modules/plot_type`.

Refer to the template in Appendix B for all the files that make up this module structure.

In what follows your own new module directory will be referred to as `YOUR_MODULE_DIR`.

1. copy the contents of the template module directory to `YOUR_MODULE_DIR`.

6.4 The papco_interface routines

These routines perform the actual interfacing to PAPCO. The consist of an input window for your plot data, and a series of small interfacing routines which are all called from PAPCO. The names for these routines are constructed internally in PAPCO, using your `plot_type` name plus the routine name: This is why it is so important to stick to the naming convention used.

All the templates for these interface routines reside in
`papco_XX/PAPCO/papco_modules/plot_type/papco_interface`.

6.4.1 Panel editor

The panel editor widget is a user supplied input widget for his module, and is used to pass parameters both to PAPCO and to the user's plot routine. The basic simple panel editor widget is supplied as a template in `papco_XX/papco_modules/plot_type/papco_interface/plot_type_panel.pro`. This is a working panel editor.

Standard Panel Editor widget

This panel editor widget is shown in Figure 6.1. For a given data type not all the widgets included in this template must be used, you may add in all or leave out all the ones that PAPCO provides. For instance, the color bar widget makes no sense for a line plot!

This widget provides some of the basic functionality of PAPCO. This is what each sub-window or button does:

Panel Height : A slider to set the height of the panel in panel units. This is not an absolute measure, a height of 2 means double the size of all other panels, no matter how many panels there are.

User defined section : At this point the user should set up his own widgets (buttons, sliders, text-entry etc.) to input the parameters needed for his plot. This is not shown here: Figure 6.1 shows the panel editor provided in the template only. The example here shows a slider that selects channel numbers.

Auxiliary file group : This is a general sub-widget that is widely used in PAPCO (in the Write Paneldata slice (Section 6.5.5), the `papco_event` and `papco_text` Standard Modules (Chapter 7). The user can specify or browse for a text file, and then edit that file. Can be used for any auxiliary information needed for a given panel.

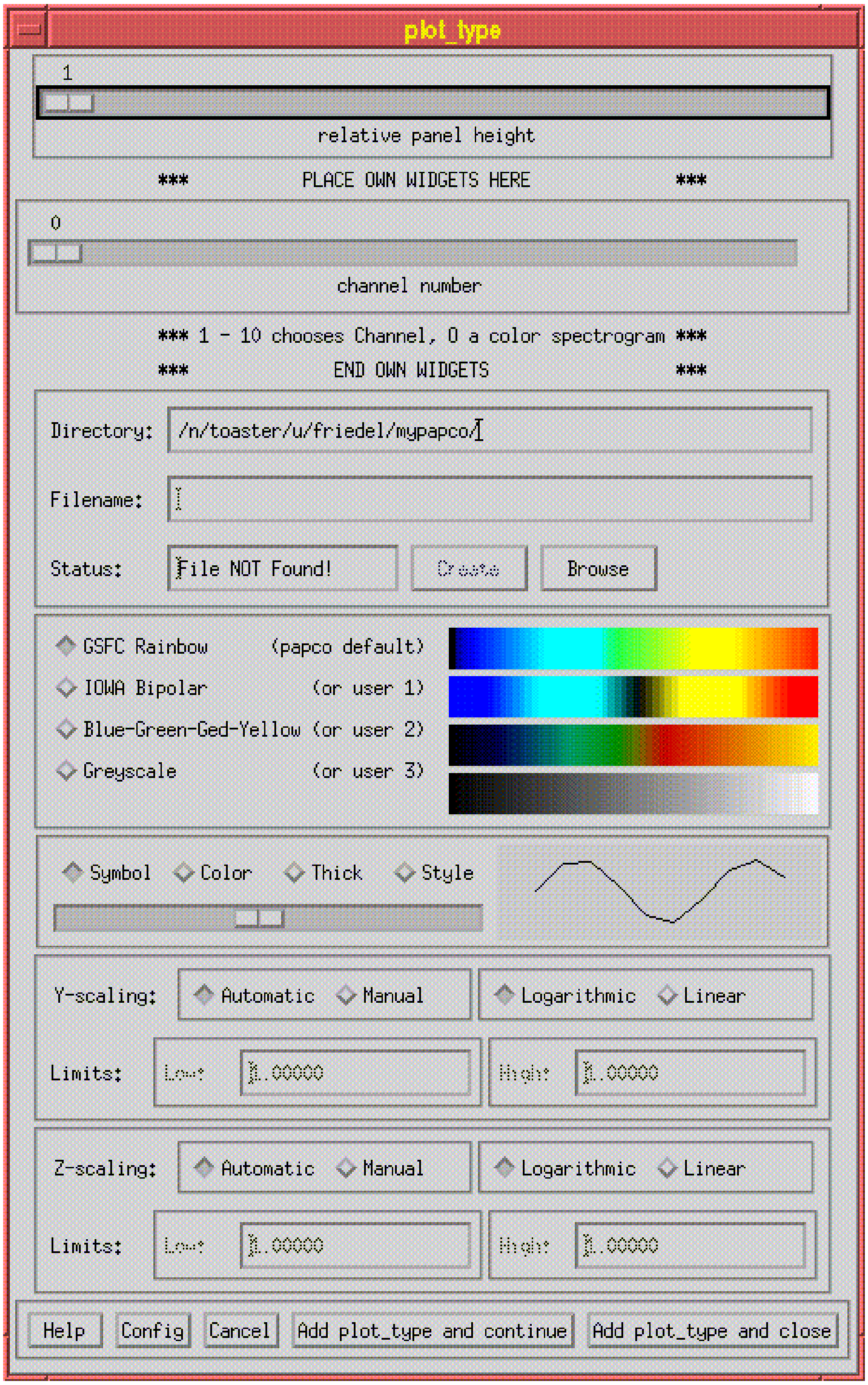


Figure 6.1: Default panel editor widget

Color Bar group : PAPCO provides the user with four color sub-tables which can be displayed at the same time. The lower three are user definable (see Section 5.4). Shown here are the currently loaded color sub-tables, and the buttons chose one of them for your plot.

Plot Style group : Useful for line-plots only, normally deleted from color spectrogram panels. Allows the user to control line style and color of his plot.

Y-scale group : Here, either automatic or manual y-scaling is set. For manual y-scaling the text entry fields become active and allow the user to enter lower and upper y-limits. Once the plot has been drawn, the automatic values are shown here.

Z-scale group : Here, either automatic or manual z-scaling is set. For manual z-scaling the text entry fields become active and allow the user to enter lower and upper z-limits. Once the plot has been drawn, the automatic values are shown here. The z-scale is normally the color scale for a spectrogram.

The action buttons : These are at the bottom of the panel, and change when the panel editor is used to edit an existing panel. The last two buttons are then replaced by **update plot_type**.

Help : Calls the help file used to describe your module.

Config : Calls up the module configuration widget. This is used to set environmental variables used by the module, e.g. data paths to your local site.

Cancel : Closes the panel editor window and does nothing.

Add plot_type and continue : Closes the panel editor window, adds the plot to the composer (right side of main PAPCO widget), and re-enters the panel editor where it left off (preserving all previous settings) and allows you to add a further panel with possibly another choice of parameters.

Add plot_type and close : Adds the plot to the composer (right side of main PAPCO widget) and closes the panel editor window.

Panel Editor Config Widget

Each panel-editor can make use of the config functionality. The idea here is to be able to customize the module via environmental variables - to set up data paths and such like.

Each module should contain a file `YOUR_MODULE/plot_read/defaults.config`. The Panel Editor Config Widget first looks for a `YOUR_MODULE.config` file under your `papco_user\papco_modules_defaults` directory, and if it can't find one it read the defaults file from the module directory itself.

You can then change these defaults to adapt to your site, the result will then be written to your `papco_user\papco_modules_defaults` directory.

An example of the Panel Editor Config Widget (for the `plot_type` template module) is given in figure 6.2.

This is what this widget does:

Current Environmental Variables : A list of the currently used environmental variables and what they are set to (usually data paths). Clicking on an entry causes the set value to appear in the "set" section for editing.

Set Environmental Variable section : Here you may manually edit the setting of a chosen entry, or use the **Browse** button to search for a path. The **Set** button then sets the new value in the list above.

The action buttons :

Done : Sets the environmental variables as currently defined in the widget. Exists the widget.

Save : Sets the environmental variables as currently defined in the widget, re-writes the user defaults file in `papco_user/papco_module_defaults`. Exits the widget.

Cancel : Does nothing. Exits the widget.

Revert : Re-reads the config file from the `papco_user\papco_modules_defaults` directory.

Default : Re-reads the original defaults.config file supplied with the module in the module directory.

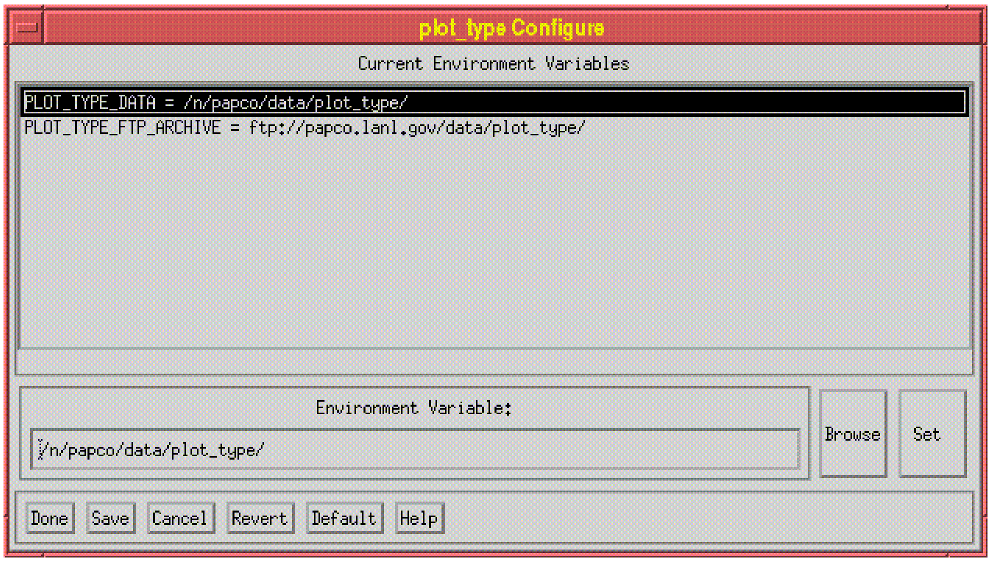


Figure 6.2: Module Configure widget

Help : A description of this widget.

To enable or disable this feature you need to make a small change to the `plot_type_panel.pro` function as described in the next section.

What you need to do

To generate your own panel editor follow these steps:

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name.
2. Compile the file and test it (see below), you should see the default panel editor.
3. You now need to add your own selection widgets (buttons, sliders, etc). If you're not too sure about IDL widget programming, just re-use and edit some of the widgets in the panel editors of other modules. The way you pass information from the panel editor to your plot routine is through the panel editor structure "PlotInfo". The panel editor is called by PAPCO with this structure as parameter. You modify this structure and pass it back to PAPCO which passes it to your draw interface routine later (see below, Section 6.4.3). Some of the tags in this structure are used by PAPCO, some are for the user:
   There are currently three user tags: `typeVector`, `switch` and `options`. The first two are there for historical reasons. We suggest that new modules are written using the options tag, which is a 20 element float array which should suffice for most uses.
   These structures are the same for ALL modules. If you absolutely cannot get by with the tags provided in this structure and need more tags added, this can be done. Contact the author of this manual for more information on this.
4. Testing: A useful thing to do is to test your panel before you proceed to see if all the input variable you set are also passed along. You can call up any of the panel editors in PAPCO (which are all functions) by passing it a default PLOTINFO structure:
   ```
   a=papco_getPlotInfoStruct()
   a=plot_type_panel(a)
   ```
   Panel editors are passed a structure containing plot information, modify and/or add to this structure, and then pass it back. So by examining `a` (`help,a,/struct`) you can test if your widget works.

5. Enable or disable the config functionality. By default it is enabled. In the file `papco_XX/papco_modules/plot_type/papco_interface/plot_type_panel.pro` find the line `widgets=papco_panel_AddActionButtons(panel_base, Action, aPlotInfo,/CONFIG)` The /CONFIG keyword enables the config button, setting this keyword to zero or deleting it disables the config button.

The testing of panels requires some built in functions of PAPCO, so you need to have started up IDL and initialized the package. The documentation in the template will lead you through what you need to do (Section B.1.1). Once your panel editor works and is tested, you can proceed to add it to PAPCO.

There are currently three such tags for the user: `typeVector`, `switch` and `options`. The first two are there for historical reasons. We suggest that new modules are written using the options tag, which is a 20 element float array which should suffice for most uses.

These structures are the same for ALL modules. If you absolutely cannot get by with the tags provided in this structure and need more tags added, this can be done. Contact the author of this manual for more information on this.

6.4.2 data_type_panel.help

This file is called by your panel editor's help button. In here you describe your data, your own widgets that you have added, and list the PAPCO functionality that your module supports. Use the template provided here (it is a simple text file, see Section B.1.4) and edit it.

6.4.3 PAPCOADD interface routines

Adding your panel editor to PAPCO is done through the file `papcoadd_plot_type.pro`, for which there is also a template. This file is the main interfacing file between modules and PAPCO, and contains all the routines that PAPCO needs to call specifically for your plot type. All these routines are short and simple. The main problems are encountered when the naming convention is not adhered to, since PAPCO uses the plot_type name you supply to construct all the calls to the procedures in this file. This is what you do:

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name. Compile the file, there should be no errors, as long as your panel editor in step 1. is called `plot_type_panel` - otherwise there is a "function not found" error.
2. Most changes which need to be made are taken care of by the global name change from plot_type to your actual name. But there are further user choices which need to be made: read through each procedure and follow the instructions in the comments.
3. The last procedure in this file is the actual bit that does the work; here your data is read and the plot call is issued.

Your read and plot routines need not exist at this time for you to test if your module works with PAPCO. Assuming your papcoadd module compiles correctly:

1. Add both files to the compile list in `YOUR_MODULE_DIR\init_module.pro`
2. Exit IDL and start-up again (`idl idl_startup_papco`). This is required since you cannot add a module to a running session of IDL which had already run a session of PAPCO: adding a module expands some of the internal structures, and IDL does not allow you to do this dynamically.
3. Make sure from the compile messages that the new module files are compiled.
4. Now start up PAPCO. Your new module should be automatically recognized and the associated button placed with the other modules at the top left of the main widget.

5. You should now have all the functionality of PAPCO available for your module (panel editor, compose window, etc.), but if you haven't got a working read or plot routine, don't try to draw anything, as PAPCO will crash.

Remember that you'll have to configure PAPCO to use your new module through the configuration widget (see Section 3.4). You'll need to exit idl for this as you cannot add a module to a running PAPCO setup.

Try out your module even if you can't plot data yet. In this way modules may be prepared ahead of time, maybe while someone else writes the read and plot routines. The documentation in the template will lead you through what you need to do (Section 6.4.3).

6.5 Simple Slices

Once you have your basic module working and can mix and play around with your data panel together with all the other data in PAPCO, then you might be thinking of adding some extra support for your data through the slice facility.

Remember, this is considered to be ADVANCED PAPCO.

In the main PAPCO widget, "Mouse Action" calls up a menu with a set of slice functions to be performed when the mouse is clicked inside a panel in the active draw window.

This uses the built-in mouse functions in PAPCO, which basically can return to the user the panel, time and y-value from a mouse-click. For the slice types already built in, PAPCO constructs a call to a slice routine you supply. This call is constructed in a logical way from your plot_type name, so again to use this facility, adherence to the naming convention is critical.

There are two basic ways in which you can add a slice. One uses PAPCO internal slices and routines which do most of the work for the user. Or the user can specify his own slice function name, write an application on his/her own, and configure PAPCO with it.

The PAPCO slices are explained here, as adding support fro them is *relatively* straightforward. Writing completely new mouse support and functionality for PAPCO is DEFINITELY ADVANCED PAPCO and somewhat more involved so that we devoted a separate Section to it (6.9).

6.5.1 PAPCO slices

The first five entries in the "Mouse Action" menu are the default PAPCO slice functions:

1. **papco Zoom** Click and drag mouse in panel to highlight a subsection of the plot to be zoomed into.

2. **papco View Slice** Calls up a widget with a plot window, for you to add whatever you want for the time of the mouse click. Useful for drawing line spectra for a given position in an energy spectrogram, etc.

3. **papco Access Level Zero** Calls a user-defined "level-zero" application for the given data type.

4. **papco Write Paneldata** Calls up a widget which allows the user to write the information of several clicks into a file. Useful for making a file of events - the same file can then be used as input for the event time standard module.

5. **papco Hair Cursor** Draws a large hair-cursor over the entire draw area and prints the x, y, and z-values to the draw window.

For the slice functions **papco View Slice**, **papco Access Level Zero** and **papco Write Paneldata** templates are provided for the required user-written procedures as outlined in Section 6.5.3.

6.5.2 The slice_type parameter

This is a 16-bit integer where each bit is used to switch a slice type on or off. This parameter needs to be set in your `draw_plot_type` procedure in the file `papcoadd_plot_type.pro`. Since each bit position corresponds to a slice type, you can add together various combinations of slices you want to support.

```
slice_type:   Selects which slices are supported and for which slice
              routines have been written.
              This is an integer logical with 16 switches. Currently
              there are 5 papco slices. You may extend this by
              further USER slices up to a total of 16 slices.
                  0 = disable all slices
                  1 = papco Zoom                 (2^0)
                  2 = papco View Slice           (2^1)
                  4 = papco Access level Zero    (2^2)
                  8 = papco Write Paneldata      (2^3)
                 16 = papco Hair Cursor          (2^4)
                 32 = USER slice A or not used   (2^5)
                 64 = USER slice B or not used   (2^6)
                 "     "      "  "  "   "    "      "
              32768 = USER slice Z or not used  (2^15)

You can support more than one type of slice by arithmetically adding the
options, eg slice_type=1+4  support Zoom and Level Zero
```

This `slice_type` parameter is also set in your `draw_plot_type` procedure in the in the PAPCO interface file `papcoadd_plot_type.pro`. Only those slice types you set there will be accessible - if your try to use a slice type in a panel from a module which does not have that slice type enabled, NOTHING will happen. A message might appear in the IDL window informing you of the function PAPCO is trying to execute, and that it can't find it.

6.5.3 Using the Draw_Slice_plot_type template

The file `plot_type_slice.pro` contains all templates for "papco View Slice", "papco Access level Zero" and "papco Write Paneldata ". These are used as follows:

The procedure followed is similar to all the others:

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name of your plot_type.

2. Enable the slice type you intend to support by changing the value of `slice_type` in your `draw_plot_type` routine in your `papcoadd_plot_type.pro`.

3. Try out the functionality. Use PAPCO to produce a plot with a panel with your plot data, select the required mouse action and click inside you panel. A vertical line should be drawn through all the panels at the time of the mouse click and the default panel should appear (see Figure 6.3) There are three standard action buttons at the bottom of the slice window:

   Help : Calls up a help window for the slice - this must be written by the module author.

   Print : Goes through the slice action again but sends whatever is plotted in the slice window to the currently selected printer action.

   Close : Exits and closes the slice widget.

4. Once the interface between your module and the default slice works you can go ahead and actually put a plot into the slice window. Follow the documentation in the template (Section B.1.3). If you get stuck, look at some of the other, previously written slices.

Figure 6.3: Default slice panel window

6.5.4 Using the levelzero_plot_type template

All this template does is provide the correct procedure name for your external application:

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name of your plot_type.

2. Enable the slice type you intend to support by changing the value of `slice_type` in your `draw_plot_type` routine in your `papcoadd_plot_type.pro`.

3. Insert the correct call to your external application and set up any parameters or common blocks you might need using the passed PAPCO time and y-position. Follow the documentation in the template (Section B.1.3).

6.5.5 Using the WriteSlice_plot_type template

PAPCO provides a standard widget for this, which by default allows the user to write time and y-position of the mouse to a user-defined file. Provision is made for the addition of further user-defined information, such as ephemeris parameters, to be added in.

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name of your plot_type.

2. Enable the slice type you intend to support by changing the value of `slice_type` in your `draw_plot_type` routine in your `papcoadd_plot_type.pro`.

3. Try out the functionality. Use PAPCO to produce a plot with a panel with your plot data, select the required mouse action and click inside you panel. A vertical line should be drawn through all the panels at the time of the mouse click and the default panel should appear (see Figure 6.4)

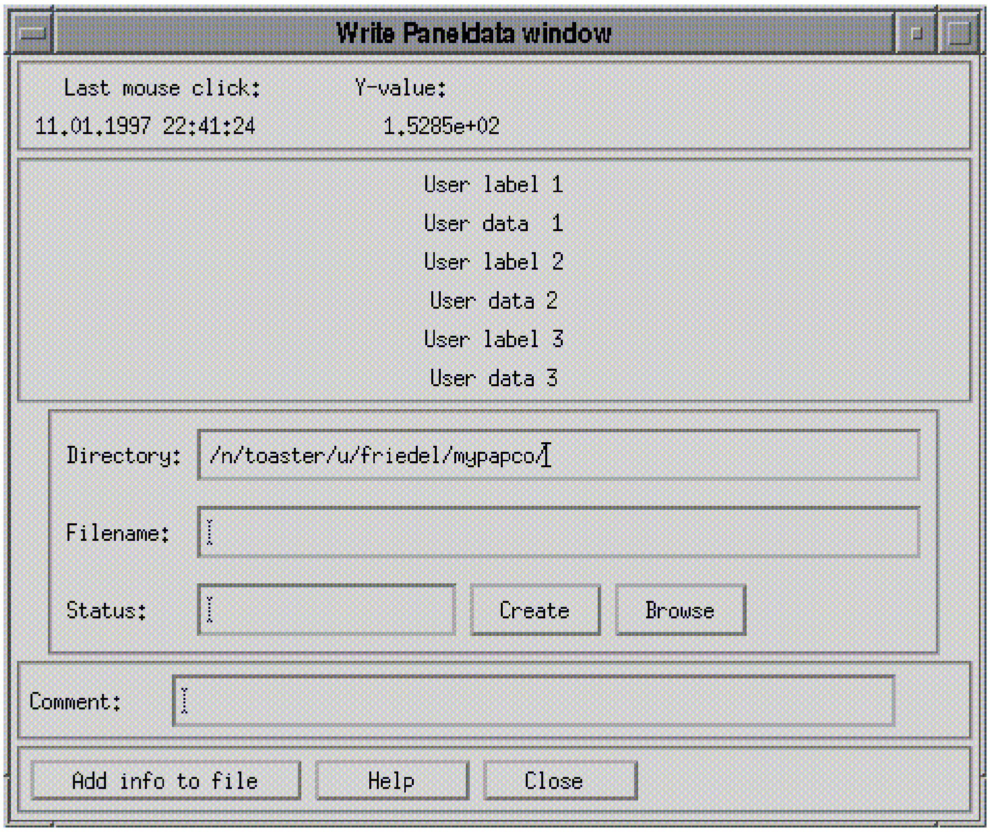


Figure 6.4: Default WriteSlice panel window

4. Once the interface between your module and the default slice works you can go ahead and add your own parameters to the window. Follow the documentation in the template (Section B.1.3). If you get stuck, look at some of the other, previously written slices.

The Write Paneldata window is an interactive one since it allows the user to define the file to which the panel data is written. This widget works thus:

Default information section : The top part gives the time and y-position of the last mouse click.

User defined section : The 6 user strings set in the WriteSlice_plot_type routine are displayed here.

File choose section : Either enter a filename manually, or browse for it. If it doesn't exist, it will be created. Once a valid file is defined the **Edit** button is activated. Press this to edit the file. The files are by default looked for and placed in you user's `papco_user/papco_product/event_time` directory, as they can also be used for the event time utility module (see Section 7.1).

Comment section : This comment string is added to the end of each line written to file.

Action buttons :

- **Add info to file** : Add the user info to the file selected. The file is appended. The format the info is written in is as string: time + y-value + user data 1 + user data 2 user data 3 + comment
- **Help** : Brings up a description of the widget.
- **Close** : Closes the window.

6.6 Plot and Read Routines

These are the main user written routines. NOTHING is prescribed here, you can read your data any which way you want, and pass it to your plot routine any which way you want. PAPCO will simply call your plot routine when it processes your panel after the DRAW button has been pressed: it calls the interface routine `draw_plot_type` from your `papcoadd_plot_type.pro` file. In that file your data is also read, so you can pass parameters to your plot routine locally through command line parameters, or globally through common blocks, which is the preferred method, as other routines (e.g. slices) can then use that data without having to read it again.

One thing is needed though. In your plot routine you need to call a PAPCO routine called `panelset` . This routine sets the viewport for your plot, i.e. the position of the panel. You can of course substitute your own routine for this, if the default PAPCO behavior does not suit you. In the Los Alamos module I have used a different routine which allows space for an extra x-axis to be plotted for each panel, even if it is not the bottom panel - to plot the local time for the data.

However, for your module to work well with all the other modules so that the output "looks good", it makes sense to stick to the PAPCO plotting and reading philosophy as described in Chapter 5.

All the plot and read templates reside in
`papco_XX/papco/papco_modules/plot_type/plot_read`.

6.6.1 Platform independence

Your read routine is the part of your module that interacts with your computer's operating system. As modules are written in order for many people to use them, you will need to pay some attention to making your read routine work under different operating systems and/or architectures.

PAPCO core has been written to be platform independent, by resolving all platform dependent calls using a set of PAPCO operating system routines. These routines are all collected in the file `papco_opsys.pro`, and are documented there. If you use only these routines, then any updates to new operating systems will become transparent to your module. Should you need routines that are not available, let me know and I'll add them to `papco_opsys.pro`.

Things you need to pay attention to are mainly filenames and file I/O. Also, if your data format is NOT one of the intrinsically portable ones (like ASCII, CDF, IDL savesets) then you need to be worried about things such as byte-swapping between Dec and SUN. You can use the IDL system variable `!version` to find out what system/architecture you are on, and take action accordingly.

```
help,!version,/stru
** Structure !VERSION, 4 tags, length=32:
   ARCH            STRING    'sparc'
   OS              STRING    'sunos'
   OS_FAMILY       STRING    'unix'
   RELEASE         STRING    '4.0'
```

6.6.2 Read-routine

Read-routines used are very much up to the user to supply as outlined in Section 1.6.1 and 5.6. To guide the user, a template read-routine is supplied which implements the expected error checking that allows PAPCO to continue should there be a problem with reading the data. It also gives an example of how to write a generic read routine that can construct data arrays that span several files, and it uses the remote get data facility. The actual techniques used for reading the data are not prescribed. However, a very effective method is using the IDL SAVE and RESTORE commands.

To use the template (`r_plot_type.pro`), this is what you do:

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name.
2. Include a common block for your data (`common plot_type_data`).
3. Adjust the error messages to your taste.
4. Write some code that constructs the file name(s) needed from the start and end times in `common mjdt, mjdt_start, mjdt_end`.
5. Write your specific file I/O.
6. Test the read-routine by setting the time common block from the command line. Use the routines in `papco_XX/papco_lib/convert.pro` to make some suitable MJDT times and load these into mjdt_start and mjdt_end before calling your read-routine. Make sure to also test correct error-handling.

The error common block is used by PAPCO to check if a valid read was done. If `get_err_no` is not zero then an empty panel is plotted which only contains the error message contained in `get_err_msg`.

The documentation in the template will lead you through what you need to do (Section B.2.1). If your system has the GNU program "wGeT" then you can use PAPCO 's remote data-get facility. See Section C.2 and B.2.4 for more details.

6.6.3 Plot routines

Once you have a panel editor, a read-routine and the papcoadd routines, you're ready to write the plot routine or adapt an old one. It might be easier to use the plot routine templates provided instead of adding your own routine, since the templates support all of the PAPCO functionality, and are designed to "stack" nicely. But the choice is yours. The procedure outlined here applies to both templates (`p_plot_type_line.pro` and `p_plot_type_spec.pro`), for line plots and for color plots.

1. Rename all occurrences of `plot_type` in the file name and in the file to the actual name.
2. Include a common block for your data.
3. If you intend to use time routines from PAPCO (for labeling x-axis), then change your time base to mjdt using PAPCO conversion routines
(can be found in `papco_XX/papco_lib\convert.pro`).
4. Change axis labeling, titles etc. to suit yourself.

5. Test the routine using either dummy or actual data by calling it from the command line. To get a full page plot set the following panel_vector:[0,1,1], and set the common time block to contain some time.

6. Once the basic plotting works, you can use PAPCO for further testing. Set the correct plot- and read-call in the draw procedure in papcoadd. From PAPCO, make yourself a product with some samples of your plot as panels (trying out all options), select a time for which you have test or real data, and save this product (if PAPCO crashes while testing, you don't have to remake the product). Now try out your plot. If you need to make editing changes to your plot routine (assuming it hasn't crashed) use the STOP button in PAPCO. Edit you routine, compile, type `.con` and redraw your product from PAPCO.

The documentation in the templates will lead you through what you need to do (Section B.2.2 and B.2.3).

6.7 Initializing your module

To make sure your module's files are compiled when PAPCO starts up, you need to make an init-file. This file HAS TO BE CALLED `YOUR_MODULE_DIR\init_module.pro` in order for PAPCO to find it. Each module has it's own `init_module.pro` file. Here is the example of what the init file for plot_type looks like:

```
print, '  --> for plot type plots -------------------------------------------'
print, '     --> read and plot routines'
.run r_plot_type
.run p_plot_type_line
.run p_plot_type_spec
print, '     --> papco support routines'
.run plot_type_panel
.run papcoadd_plot_type
.run plot_type_slice
```

As you can see, the init-file is divided into plot- and read-routines, and those that are needed to interface with PAPCO.

Each time PAPCO is started, it searches all the `YOUR_MODULE_DIR\init_module.pro` files it can find in the IDL search path and writes its own `papco_user\papco_add_init.pro` file dynamically. A new file is produced each time you start PAPCO, and you can examine it to see whether your modules have been found.

Depending on which modules you plan to add to your PAPCO session, you will need to make changes in the PAPCO configure widget (see Section 3.4). The `papco_user\papco_add_init.pro` file is automatically produced and updated.

Remember: PAPCO will add a button for all the modules that have their `papcoadd_plot_type` file in `YOUR_MODULE_DIR\papco_interface`.

6.8 Extending PAPCO read by orbit to new missions

WARNING! This section is "advanced" PAPCO!

If you are adding a module for a new mission, say EQUATOR-S, then you might want to be able to choose your time range by orbit number. This has so far been implemented for CRRES and POLAR.

With this release of PAPCO the read by orbit functionality has been rewritten and modularized, so it becomes simple to add another satellite mission to this feature.

The way this works is simple. The user needs to write a routine that can compile a list of orbit start/end times based on his/her set of ephemeris data for the mission. Typically there will be a set of files per day with ephemeris information available, like for POLAR. PAPCO provides a template for such a routine (see Section B.3). You need to put that information into a PAPCO structure, the rest is then handled by PAPCO.

For CRRES, the ephemeris is read from the CRRES ephemeris files (which are already per orbit) and the POLAR_orbits.info file resides in that directory. For POLAR, the ephemeris is read from the POLAR ephemeris file, and the CRRES_orbits.info file resides in the POLAR ephemeris data directory. So if you want to use this feature, you need to have these files, and include in your `papco_set_variables.pro` file under your `papco_user_XX` the environmental variables that point to the location of these files. For the PAPCO installation at LANL, these settings are:

```
papco_setenv, /FS, 'POLAR_ORBITS=/n/leadbelly/belly15/data/polar/eph'
papco_setenv, /FS, 'CRRES_ORBITS=/n/leadbelly/belly15/data/crres/eph'
```

Obviously, if you add a new mission to this scheme, you need to set a new environmental variable called `MISSION_ORBITS`.

The PAPCO read by orbit feature allows you to update your orbit list for you mission from within PAPCO, for ongoing missions, as new ephemeris files become available. For this to work, you of course need all the ephemeris files to be available too. Refer to the CRRES modules[1] (maintained by MPAe Lindau).

Including a new satellite in this scheme is not a difficult task, but requires changes to the PAPCO core and is considered advanced PAPCO. If you do add a new mission please let the author of this document know so that is can be included in future PAPCO releases.

The routines handling the "read by orbit" feature widgets and functionality are contained in `papco_XX/papco/papco_orbitinfo.pro`. They are written generally and should work for any mission.

The only routine that needs to be written by the user is `mission_createorbitinfo.pro` where "mission" needs to be replaced by the name of the satellite. This routine creates the file of orbit information needed by PAPCO. It has to reside in the PAPCO core directory `papco_XX/papco/`. A template for this is included under `papco_XX/papco_lib` (see Section B.3)

To use the template (`mission_createorbitinfo.pro`), this is what you do:

1. Copy the file `mission_createorbitinfo.pro` to the PAPCO core directory `papco_XX/papco`.
2. Rename all occurrences of `mission` in the file name and in the file to the actual name.
3. Insert some code that reads through your ephemeris files and construct a list of orbit start/end times.
4. Include the correct environmental variable to point to your MISSION_orbits.info file in your `papco_set_variables.pro` file under your `papco_user_XX`.
5. Add a compile line for your `mission_createorbitinfo.pro` to the `papco_init.pro` file under `papco_XX/papco` so that it gets compiled on startup.
6. Inform the author of this document that you have added a new MISSION so that it can be included in future releases.

By reading through existing `mission_createorbitinfo.pro` files and the routines contained in `papco_XX/papco/papco_orbitinfo.pro` the user should have no problem in adding a new mission.

PAPCO on startup automatically searches through it's core directory for any `mission_createorbitinfo.pro` files to include "mission" in it's list of satellites for the read by orbit function.

6.9 USER defined slices

WARNING! This section is "definitely advanced" PAPCO!

[1] http://leadbelly.lanl.gov/ccr/software/papco/papco_modules.html

The user can add his/her own slice definitions to the "Mouse Action" menu. and thus extend the functionality of his/her module. The maximum total slice functions are 16, so given the 5 PAPCO slices the user can define up to 11 slices of his/her own.

In addition the user can define what action is taken in the draw window for his slice type. This is defined as being the "show-selection". In keeping with PAPCO tradition there is a choice between specifying PAPCO supplied show-selections or writing your own.

6.9.1 Interfacing your own User Slice

Adding a new slice type is done by setting an environmental variable in `papco_set_variables.pro` in the user's `papco_user_XX` directory:

```
papco_setenv,       'USER_MOUSE_ACTIONS=View_Polar_Slice/View_Ephemeris'
```

The format of this string is `new_slice_1/new_slice_2/new_slice_3` etc., no spaces allowed, individual slice names separated by a backslash.

PAPCO uses this environmental variable to extend the "Mouse Action" menu, and to construct the calls to the user's slice routine. The calls have the format

```
user_new_slice_1_plot_type, panelNr, seconds, yValue, CLOSE=CLOSE
```

where the name of the routine is a string made up out of the new slice name and the name of the module of the panel that the mouse is clicked in; panelNr is the No. of the panel the mouse was clicked in, seconds and yValue are the x and y coordinates of the plot in data coordinates. The keyword CLOSE is set by PAPCO when a new slice type is chosen, it is up to the user to take action on this and make sure that his slice widgets are closed at that point.

Knowing the panelNr allows you to get access to the structure defining the panel that the mouse was clicked in. This is the same structure that was defined with the panel editor for that panel. PAPCO keeps an array of all these structures in it's own internal main structure, which is put into a common block so that it accessible to the rest of PAPCO and to the user:

```
COMMON PLOT_COMPOSER, widgetData
```

You can interrogate this structure using the `help, widgetData, /stru` command on the IDL command line. You can always get at the current state of `widgetData` when in PAPCO: Press the STOP button on the main widget, that get's you to the IDL command line.
`.con` then resumes PAPCO.

The tag name in `widgetData` containing the array of all currently defined panelplots is called `plotsDrawn`. To get access to this in your slice routine, you need to include the common block `PLOT_COMPOSER`, then

```
thisplotinfo=widgetData.plotsDrawn(panelNr)
```

returns you the plot info structure of the panel you clicked in.

Often a slice function that you write needs to get at the data plotted in the panel you clicked, and this data might not always be available in a common block - so you need to know how to re-read the data. It is for this reason that you defined the sting `get_data_call` in your `draw_plot_type` procedure in the PAPCO interface file `papcoadd_plot_type`. This string is also available in the structure:

```
get_data_call=thisplotinfo.get_data_call
```

If you set up your `get_data_call` correctly in your PAPCO interface routine, then it should contain exactly the call you used to obtain the data that was plotted in the panel, so you can re-read the data by executing that string again:

```
IF NOT EXECUTE(widgetData.plotsDrawn(panelNr).get_data_call) THEN BEGIN
  dummy=messageBox(['I cannot read the data for the panel'], ['Ok'])
  RETURN
ENDIF
```

This little bit of code tries to execute your data call, and pops up a message widget if it was not successful.

NB. Please, if you have written a user slice for your data, include some documentation on that slice in your module's help file (Thats the one that gets called up when you hit the help button in the panel editor for that module). Also, if your slice is another widget, make sure it has a help button too that explains its use!

6.9.2 Selecting the show-selection type for your User Slice

The show-selection here is defined as the action taken in the current draw window for a given Mouse Action. For ZOOM, the show selection is drawing a box, for SLICE it's drawing a vertical line etc. These show-selections are now also available for User defined slices.

PAPCO provides routines for several built-in show-selection types. These types are identified by the following names:

"vert_line"	-	draws a vertical line over all panels at mouse click
"horiz_line"	-	draws a horizontal line through panel at mouse click
"hair_cursor"	-	draws a hair cursor made up of "vert_line" and "horiz_line"
"box"	-	draws an inverted box in the panel when dragging the mouse

The PAPCO mouse actions use these show-selection types and are pre-configured in the following way (and cannot be changed):

ZOOM	uses box
VIEW SLICE	uses vert_line
ACCESS LEVEL ZERO	uses vert_line
WRITE PANELDATA	uses vert_line
HAIR CURSOR	uses hair_cursor

While the user was able to add in his/her own USER slice type by setting the environmental variable `USER_MOUSE_ACTIONS`, the show-selection had been hard-wired to vertical line only. This is still the default action when `USER_SHOW_ACTIONS` is not defined.

The user can now also define an environmental variable `USER_SHOW_ACTIONS` to select the show-selection for his/her User Mouse Action. This is done be setting `USER_SHOW_ACTIONS` to the show-selection type names chosen. If you have added 3 mouse actions of your own, then you have set

`USER_MOUSE_ACTIONS = new_slice_1/new_slice_2/new_slice_3`

To specify which show selection type each user mouse action should use, set

`USER_SHOW_ACTIONS = show_1/show_2/show_3`

The association between Mouse Action and Show Action is by order in these two system variables. Replace `show_1` etc by valid show-selection names (see above).

6.9.3 Getting at information

All the routines that implement the show-selection share a common block, which is available to the user:

`COMMON MOUSE_SELECT,old,click,pnlNr,n_x,n_y`

where

`click`	2-element float	Contains the last x y values of where the mouse was last clicked. If you drag the mouse, values are from when the mouse was last released. Normal coordinates.
`old`	2-element float	Contains the previous last clicked coordinates. When mouse was dragged, contains the start of the drag. Normal coordinates.
`pnlNr`	an integer	The panel Nr where the mouse was clicked. Useful for getting at that panels' plotinfo structure. All the plotted panels have their plotinfo structures (thats the same structure as returned by that modules panel editor) stored in an array in the main papco common block `COMMON PLOT_COMPOSER, widgetData`. This array is under `widgetdata.plotsdrawn`, use pnlNr to get at that panel's plotinfo: `myplotinfo=widgetdata.plotsdrawn(pnlNr)`
`n_x,n_y`	2-element float	Contains the current mouse coordinates independent of what was clicked. Normal coordinates.

These data are in normal coordinates as PAPCO doesn't know or care about your actual coordinates used. For drawing show-selections, you don't need data coordinates anyway.

6.9.4 Writing your own show-selection

Given the information in the common block above the user can write his/her own show-selection routine.

For this routine to interface into PAPCO , it MUST adhere to a naming convention. Look a the built-in show-selection routines in `papco_cursor.pro`.

You'll have to give your own show-selection type a name. Lets assume you've called it "my_box", then you will need to write a routine called:

```
pro papco_cursor_showselection_my_box, ERASE=ERASE, DRAW=DRAW

  COMMON PLOT_COMPOSER, widgetData
  COMMON MOUSE_SELECT,old,click,pnlNr,n_x,n_y

; put in here what your show-selection does!

end
```

I.e., papco will look for a routine whose name is made out of "papco_cursor_showselection_" + your selection name.

The best place for this is in your modules slice routines. In that way it'll get compiled up at startup automatically.

You can the select your show-selection type by setting `USER_SHOW_ACTIONS` accordingly, as explained in Section 1.

Note: To keep with PAPCO philosophy any show-selection you write should ONLY use the info supplied in the common block `MOUSE_SELECT`, and should do everything in normal coordinates, so that other mouse actions may also use it! As more show-selections get written we can also include them in PAPCO core to make them available outside your module.

6.9.5 Getting more info for your USER slices

While normal coordinates are all that's needed for drawing show-selections, you might want more information for actually constructing a user slice. When you have configured PAPCO with your USER slice, papco calls your slice with three parameter: the panel Nr and the data values for the x and y positions of the click, seconds and yvalue.

"seconds" will be in whatever format you've chosen for the time axis. There is a papco_cursor utility routine that converts that to a verbose description of the time:

```
FUNCTION papco_cursor_time,user_time,panelNr,MJDT=MJDT
```

If MJDT is set, the time gets returned in mjdt format.

If your show-selection is a box-style, enclosing an area of the panel, you'll need to get hold of the box coordinates. An example of this can be found the the `papco_Cursor_Rect` routine in `papco_cursor.pro`. But the important code to use is the following:

```
papco_get_data_box, starttime, endtime, startData, endData
```

which is a papco_cursor routine that returns to you, in data coordinates, the x and y start/end points (corresponding to the time and data axes) of the area enclosed by your mouse drag in the current panel.

6.9.6 Multiple mouse button support

IDL supports a three-button mouse, which is available on most systems and can be emulated if needed (under W95 pressing both buttons on an MS Mouse can emulate the third button).

Up to now, all mouse actions where controlled by the left button, and this is still the default mode. Pressing the left button in a panel will call up the selected Mouse Action and perform it, if that action is enabled for that panel.

Support has now been added for the other two mouse button. The configuration is hard-wired, at present it is not planned to make the mouse buttons configurable (although this is the logical extension and in principle not difficult!)

This is what is currently implemented:

left button	-	default mouse action as defined by Mouse Action in PAPCO
	-	main Window.
middle button	-	calls hair_cursor mouse action
right button	-	calls zoom mouse function.

Whenever the middle or left button is pressed, that function is called, keeping the default action active too. Going back to the left mouse button returns you to the default action and erases the hair_cursor / zoom windows.

6.9.7 Assigning slice option numbers

As PAPCO cannot know in advance what user slices will be added, and the user cannot know how many other user slices might be used together with his/hers, there needs to be a mechanism to interrogate PAPCO which slice number was assigned to a given user slice.

The first 5 slice types are preset with values 1, 2, 4, 8 and 16. User slices are assigned consecutively (in powers of two) in the order they appear in the `USER_MOUSE_ACTIONS` environmental variable. PAPCO provides a routine which returns the slice value given the user slice name:

```
slice_assigned_value=PAPCO_slice_value('USER Slice Name')
```

Note that `USER Slice Name` is specified exactly the way it appears in the "Mouse Action" menu.

So, if you want to set up you module to use Zoom, Hair Cursor and your own slice "Do my own thing" you need to set up `slice_type` as:

```
slice_type= 1 + 16 +  PAPCO_slice_value('USER Do my own thing')
```

6.9.8 Keeping things general

When designing your own slice function try and keep your code general and portable. If you set out from the beginning having the idea that other modules could also use your slide in mind, then there is no real extra effort involved.

PLEASE comment your code thoroughly and provide ample documentation about it it your modules help file (the one accessed from your panel editor) or provide your slice window with it's own help button.

Any user written slice that becomes useful to more than one module is a bonus and can then be easily included as a "PAPCO slice" in the next release.

Chapter 7

Standard Modules

As PaPCo grows the need for more functionality arises. Whenever possible, such functionality should be added in the form of a module, to minimize the changes needed to be made to the PaPCo core.

To date three such modules of general use have been written. Users who write further modules of general use should please contact the author of this document so that they can be included in the PaPCo documentation and be made available in the distribution.

7.1 PaPCo Event Module

This module is in the PaPCo distribution under `papco_XX/papco_modules/papco_event`. It grew out of the need to annotate graphs with event time markers to indicate the position in time of a specific feature of the plot, or to plot related event times (such as the times of Sudden Storm Commencements). The Panel Editor for this module is shown in Figure 7.1

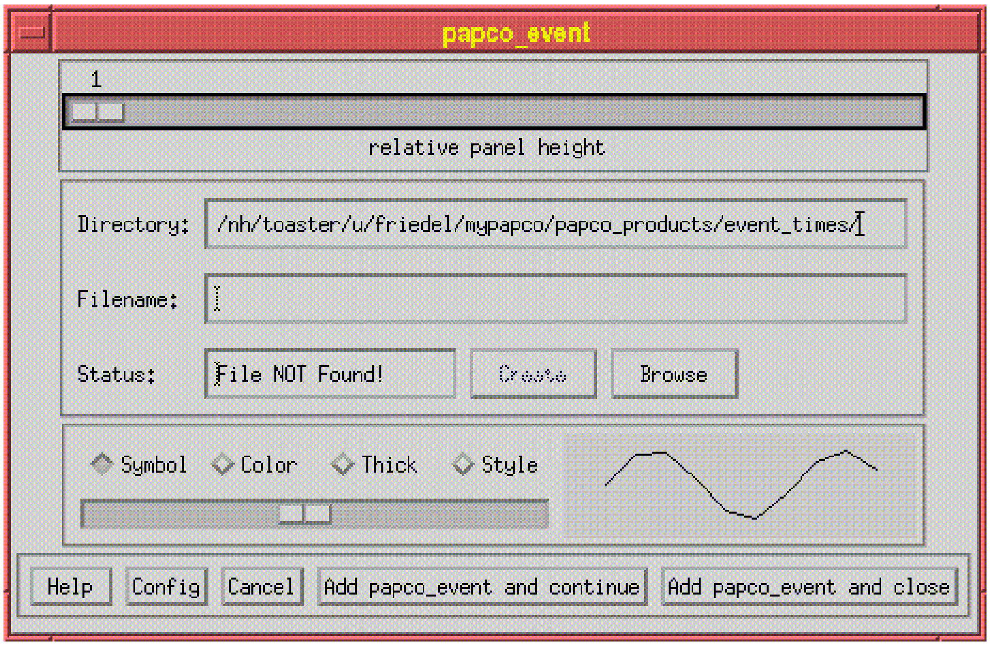


Figure 7.1: PaPCo Event Panel Editor

The module allows one to define a panel in which symbols to mark the event times are drawn in the middle of the panel, or vertical lines are drawn at the event times. Using the PaPCo Plot Style Group sub-widget allows the user to have full control over the type of line used. By using the Panel Compose Area buttons this panel can be made to overplot some other or all other panels, providing a lot of flexibility.

This module allows the user to define his own file of event times interactively. These are stored in text files with the extension `.events` in the directory `papco_user_XX/papco_products/event_times` and can also be edited outside PaPCo. The time format can be any of the formats accepted by PaPCo.

The Write Paneldata slice, which is used to interactively write information at a mouse click in a panel to file, also produces text files which have lines of information in them, each line starting with a time. These files can be used as input to this module, which allows for further flexibility. For instance, you can use one type of plot to get events from which you then overplot onto another type of plot.

For further information on the use of this module refer to the PaPCo Event Panel Editor help.

To use this module you need to configure PaPCo with it like any other module (see Section 3.5). The first time you use it, you also need to configure the module itself; the default location for all the event files is `papco_user_XX/papco_products/event_times/` and all the efent files have extension `.event`.

7.2 PaPCo Text Module

This module is in the PaPCo distribution under `papco_XX/papco_modules/papco_text`. It grew out of the limitations which the old way of supplying titles to the plots imposed: The top panel controlled the title. With this module any title can be defined as a panel and placed anywhere. The Panel Editor for this module is shown in Figure 7.2

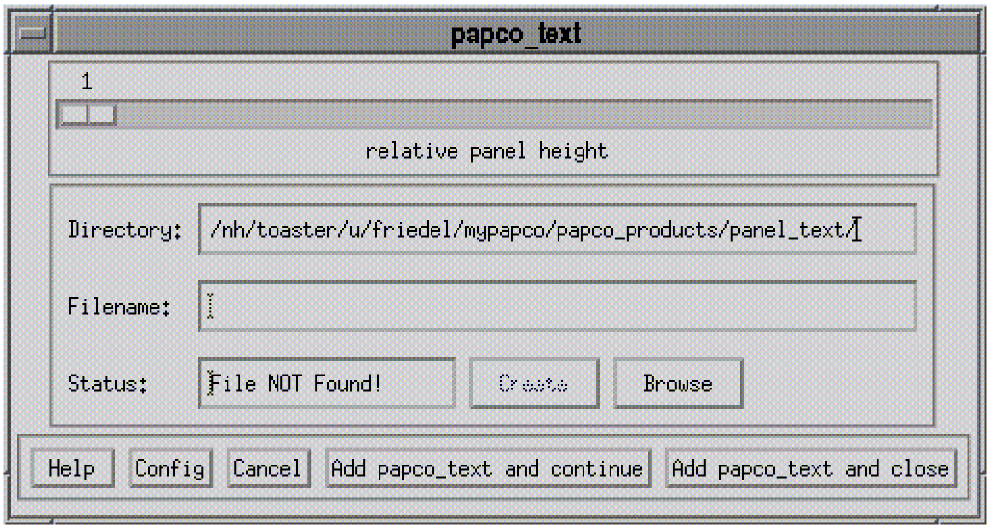


Figure 7.2: PaPCo Text Panel Editor

This module allows the user to interactively create a simple text file or to use a pre-existing one, which will be plotted into a panel line by line using the IDL xyouts procedure, thus allowing the use of all IDL font and positioning commands. These are stored in text files with the extension `.text` in the directory `papco_user_XX/papco_products/panel_text` and can also be edited outside PaPCo.

By using the Panel Compose Area buttons this panel can be made to overplot some other or all other panels or be put between any panels, providing a lot of flexibility beyond the task of providing a title for the whole plot!

For further information on the use of this module refer to the PaPCo Text Panel Editor help.

To use this module you need to configure PaPCo with it like any other module (see Section 3.5). The first time you use it, you also need to configure the module itself.

7.3 plot_type Module

This module is in the PAPCO distribution under `papco_XX/papco_modules/plot_type`. This is a fully functional PAPCO module designed to be a developer's module. It also serves as a vehicle for trying out and demonstrating all of PAPCO functionality using dummy data for a single day (11. January 1997).

This module is the only one which is part of PAPCO *core responsibility, and as such will always represent cutting edge* PAPCO*. If you need to find out how to make use of some new* PAPCO *features, look at their use in this module.*

To design your own module start with this working module and change it to your own requirements! See Chapter 6 for details.

Chapter 8

Future PaPCo, beyond PaPCo

PAPCO was written because of the basic need to have an easily extendible package for the scientific analysis of a great variety of data.

However, PaPCo is not only a stand-alone application, but part of a larger philosophy of data analysis and dissemination, which is aimed at making data available to a large audience. This has now grown to make PaPCo a full multi-purpose data display AND analysis package, while still keeping to the modular philosophy, allowing the user full power withour prescribing anything.

It has been said that if one gives a scientist any data tool to use, it will take him 5 minutes to find out what he CANNOT do with it. The basic idea behind PaPCo is to answer the question “Can you do such and such?” with “YES! But you will have to do it!”

8.1 Sharing PaPCo products

Once data is on-line and has been added in the form of modules, the user can define a set of products which he/she wants to make available to the scientific community. This can be in the form of overview plots. Or, the user needs to get together a product for a select study on which he/she collaborates with a range of international colleagues. Once these products have been defined, the product files only need to be “passed around” - as long as everyone used PaPCo and has the same data available! Even if not everybody has the data available, as the use of the get remote data facility grows, this does not become a problem.

Alternatively the user can create a set if gif files to be distributed via the Web. For this the batch facility in PaPCo was provided. Given a product and a batch file of times, a series of plots can be produced automatically. This can be done from command line PaPCo.

8.2 Using a graphical data base

Such a set of PostScript files could then form the basis for the next step. As PostScript is an unhandy way of distributing large amounts of data, we use the QuickView package provided as part of the *Archimede* Project of CNES, Toulouse, France.

QuickView itself is freeware that can be obtained from CNES via ftp. Contact Alain Caranana at CNES (`caranana@sc2000.cst.cnes.fr`) for details. QuickView is a dedicated graphical viewing package that is used to “page” through a large amount of pre-defined graphs in GIF format. It uses a data base like Graphical Data Distribution Standard (GDDS) which keeps each plot in a range of resolutions, allowing some zooming capabilities, and allows the user to group and search his data by user-defined tags. QuickView is meant to replace cupboards full of summary plots by electronic means, typically a set of GIFs on CDROM. It is the tool that has been selected for the Interball mission and would also have been used by the French CLUSTER community (CIS).

CNES also provides a set of tools which make handling SDDG very easy. SDDG_tools allows one to create SDDG data structures from a directory full of PostScript files, using freeware products such as ghostview. Further tools allow one to add and merge SDDG structures.

These can subsequently be viewed using QuickView, for which the package needs to be installed locally. However, if one does not want to do that, there is a further tool called SDDG_Web, which translates the SDDG structure to html, preserving most of the QucikView functionality. This makes the data instantly available to a large community via WWW.

The whole process can easily be automated into chain production, and it is hoped that SDDG and QUICK-View will evolve into a widely used standard. As this package takes PostScript as input, you do not even need to use PAPCO as the front end, any plotting package which can produce a set of PostScript files can be used.

8.3 Web PAPCO

Given a PAPCO product and a batch file of times PAPCO can be called from the command line. This opens up the possiblility of having a WWW interface, which allows the user to select a pre-defined PAPCO product and a time range, and to submit a PAPCO -job on-line. PAPCO output in form of postscribt and/or gif are then available via WWW to download or just view.

Such a prototype PAPCO Web interface is in existence at LANL. Further development on this aim to provide some of the panel compose functionality via the Web too.

Appendix A

Running Logs

Given here are a set of sample outputs from a PAPCO session that you might expect to see, when IDL is started, when PAPCO is started and when PAPCO is exited. The output here is from the installation at LANL.

A.1 IDL startup log

When you startup IDL with PAPCO, a lot of status messages are printed to the terminal window in which you started IDL. These messages include a list of environmental variables which have been set and a list of programs which are compiled. Assuming you've started PAPCO with the command line

`idl papco_setup`

this is what you should see: (papco_sample_startup_a.text)

```
papco->papco_user% idl papco_setup
IDL Version 5.2 (sunos sparc). Research Systems, Inc.
Installation number: 679-1.
Licensed for use by: NIS, LANL

For basic information, enter "IDLInfo" at the IDL> prompt.


=====================================================================
--- W E L C O M E   T O   P A P C O ! -------------------------------
=====================================================================

--- Start papco_set_variables ---------------------------------------

--- Expanding IDL search path for PAPCO core, Lib, Home -------------

--- Compile PAPCO routines needed for startup -----------------------

=====================================================================
--- papco - Panel Plot Composer - version 9.0, 20 Aug 1998 ----------
=====================================================================

--- Environmental variables now set: --------------------------------

PAPCO_QUIET:        0
PAPCO_STARTUP_MODE: COMPILE
PAPCO_ADDINS:       NO
PAPCO_HOME:         /n/toaster/u/friedel/papco_user
PAPCO_PATH:         /n/leadbelly/belly1/home/papco/papco/papco
PAPCO_MODULES:      /n/leadbelly/belly1/home/papco/papco/papco_modules
PAPCO_DOWNLOAD:     /n/toaster/u/friedel/papco_user/papco_download_temp
TIME_CONV:          /n/leadbelly/belly1/home/papco/papco/papco_lib/cdssw/time
POLAR_ORBITS:       /n/papco/data/polar/eph
CRRES_ORBITS:       /n/papco/data/crres/eph
USER_MOUSE_ACTIONS:
                    View_Polar_Slice
                    View_Ephemeris
```

```
                    Distribution
USER_SHOW_ACTIONS:
                    hair_cursor
                    hair_cursor
                    vert_line

--- Done papco_set_variables ----------------------------------------

--- Running papco Configure widget ----------------------------------

% Compiled module: XMANAGER.
% Compiled module: XREGISTERED.
```

After this the configure widget should appear. Once you've made your choices of modules, chosen your startup mode and exited this widget, startup will continue. (This step is skipped if you've started PaPCo with `idl papco_startup`.)

The example here is for choosing the modules "plot type", "papco text" and "papco event", and the startup mode "Compile on startup", "Do not include add-Ins on startup" and "Verbose compile on startup" (all the "% Compiled Module:..." messages are suppressed if you choose "Quiet compile on startup"). The startup should continue with: (papco_sample_startup_b.text)

```
--- Start papco_compile ---------------------------------------------

--- Start papco-init ------------------------------------------------

--- Time utilities --------------------------------------------------
% Compiled module: CONVERT_TZERO_TO_T90.
% Compiled module: CONVERT_T90_TO_TZERO.
% Compiled module: CONVERT_T90_TO_DATE.
% Compiled module: CONVERT_DATE_TO_T90.
% Compiled module: CONVERT_SECONDSTOTIME.
% Compiled module: CONVERT_TIMETOSECONDS.
% Compiled module: CONVERT_DAYTODATE.
% Compiled module: CONVERT_DATETODAY.
% Compiled module: ADDSUBTIME.
% Compiled module: UTC2TAI.
% Compiled module: TAI2UTC.
% Compiled module: INT2UTC.
% Compiled module: GET_LEAP_SEC.
% Compiled module: GET_UTC.
% Compiled module: ANYTIM2CAL.
% Compiled module: ANYTIM2UTC.
% Compiled module: MJD2DATE.
% Compiled module: CHECK_EXT_TIME.
% Compiled module: UTC2INT.
% Compiled module: DATE2MJD.
% Compiled module: CHECK_INT_TIME.

--- Satellite by orbit utilities ------------------------------------
% Compiled module: PAPCO_GETORBITINFOSTRUCT.
% Compiled module: PAPCO_ORBITDESCRIPTIONLIST.
% Compiled module: PAPCO_WRITEORBITINFO.
% Compiled module: PAPCO_READORBITINFO.
% Compiled module: PAPCO_GETORBITINFOFOR.
% Compiled module: PAPCO_SELECTORBITS_REFRESH.
% Compiled module: PAPCO_SELECTORBITS_CALCBOUNDS.
% Compiled module: PAPCO_SELECTORBITS_EVENT.
% Compiled module: PAPCO_SELECTORBIT.
% Compiled module: POLAR_CREATEORBITINFO.
% Compiled module: CRRES_CREATEORBITINFO.

--- Other routines for papco ----------------------------------------
% Compiled module: DEALLOC_LUN.
% Compiled module: XMGR_FAKE.
% Compiled module: B_BUTTON.
% Compiled module: ANN_CLOSEST.
% Compiled module: ANN_CLOSEST_OBJ.
% Compiled module: ANN_GET_NUM_EVENT.
% Compiled module: CW_CONF_EVENT.
% Compiled module: CW_CONFIRM.
% Compiled module: ANN_GET_NUM.
% Compiled module: ANN_RESET_MODE.
% Compiled module: ANN_ADD_OBJECT.
% Compiled module: ANN_SET_CONTROLS.
% Compiled module: ANN_REFRESH_LIST.
```

```
% Compiled module: ANN_LOAD_OBJ.
% Compiled module: ANN_XFER_FILE.
% Compiled module: PAPCO_ANNOTATE_PS.
% Compiled module: PAPCO_ANNOTATE_EVENT.
% Compiled module: ANN_MOVE_RESIZE.
% Compiled module: ANN_DRAW_EVENT.
% Compiled module: ANN_DRAW_OBJECT.
% Compiled module: PAPCO_ANNOTATE.
% Compiled module: PAPCO_MODULE_CONFIGURE.
% Compiled module: PAPCO_MODULE_CONFIGURE_EVENT.
% Compiled module: R_MODULE_CONFIG.
% Compiled module: R_MODULE_DEFAULT.
% Compiled module: S_MODULE_CONFIG.
% Compiled module: PAPCO_DRAW_WINDOW.
% Compiled module: PAPCO_DRAW_WINDOW_EVENT.
% Compiled module: PAPCO_GETPRINTERINFOSTRUCT.
% Compiled module: PAPCO_PRINTER_REFRESH.
% Compiled module: PAPCO_PRINTER_EVENT.
% Compiled module: PAPCO_PRINTER.
% Compiled module: ENTRYBOX_EVENT.
% Compiled module: ENTRYBOX.
% Compiled module: MESSAGEBOX_EVENT.
% Compiled module: MESSAGEBOX.

--- PAPCO-main-program ----------------------------------------------
% Compiled module: PAPCO_SETCOLORTABLE.
% Compiled module: PAPCO_LOAD_SUBTABLE.
% Compiled module: PAPCO_SET_SUBTABLE.
% Compiled module: PAPCO_RELEASE_SUBTABLE.
% Compiled module: PAPCO_SUBTABLE_USED.
% Compiled module: PAPCO_OCCUPY_SUBTABLE.
% Compiled module: PAPCO_GET_COLORINDICES.
% Compiled module: PAPCO_GETCOLOR_TABLE.
% Compiled module: PAPCO_GET_SUBTABLE_INDEX.
% Compiled module: PAPCO_GET_ZVALUE.
% Compiled module: GET_PLOTSTYLE_VALUE.
% Compiled module: GET_USER_SYMBOL.
% Compiled module: PAPCO_COLORTHICKNESS_CODER.
% Compiled module: PAPCO_STYLE_CODER.
% Compiled module: PAPCO_PANEL_ADDHEIGHTWIDGETS.
% Compiled module: PAPCO_PANEL_RETURNDATA.
% Compiled module: PAPCO_PANEL_ADD_PLOTSTYLEBUTTONS.
% Compiled module: PAPCO_PANEL_IS_SCATTERPLOT.
% Compiled module: PAPCO_PANEL_ADDACTIONBUTTONS.
% Compiled module: PAPCO_PANEL_CREATEWIDGETDATA.
% Compiled module: PAPCO_PANEL_REALIZE.
% Compiled module: PAPCO_PANEL_ADDYSCALE_WIDGETS.
% Compiled module: PAPCO_PANEL_ADDZSCALE_WIDGETS.
% Compiled module: PAPCO_PANEL_ADDCOLORBAR_WIDGETS.
% Compiled module: PAPCO_PANEL_ADDFILE_WIDGETS.
% Compiled module: PAPCO_PANEL_ADDFILE_EDITFILE.
% Compiled module: PAPCO_PANEL_ADDFILE_EDITFILE_EVENT.
% Compiled module: PAPCO_PANEL_REFRESH.
% Compiled module: PAPCO_PANEL_EVENT.
% Compiled module: PAPCO_CURSOR_TIME.
% Compiled module: PAPCO_NORMAL_TO_DATA.
% Compiled module: PAPCO_DATA_TO_NORMAL.
% Compiled module: PAPCO_CURSOR_WHICHPANEL.
% Compiled module: PAPCO_WRITE_PANELDATA_EVENT.
% Compiled module: PAPCO_WRITE_PANELDATA.
% Compiled module: PAPCO_DRAWSLICE_EVENT.
% Compiled module: PAPCO_DRAWSLICE.
% Compiled module: PAPCO_DRAW_HORIZ_SLICE_EVENT.
% Compiled module: PAPCO_DRAW_HORIZ_SLICE.
% Compiled module: PAPCO_CURSOR_UPDATEDATA.
% Compiled module: PAPCO_CURSOR_RECT_EVENT.
% Compiled module: PAPCO_CURSOR_RECT.
% Compiled module: PAPCO_GET_DATA_BOX.
% Compiled module: PAPCO_CURSOR_SETPOINTER.
% Compiled module: PAPCO_CURSOR.
% Compiled module: PAPCO_CURSOR_SHOWSELECTION.
% Compiled module: PAPCO_CURSOR_SHOWSELECTION_VERT_LINE.
% Compiled module: PAPCO_CURSOR_SHOWSELECTION_HORZ_LINE.
% Compiled module: PAPCO_CURSOR_SHOWSELECTION_HAIR_CURSOR.
% Compiled module: PAPCO_CURSOR_SHOWSELECTION_BOX.
% Compiled module: PAPCO_CURSOR_SETMOUSEDATASTRUCT.
% Compiled module: PAPCO_CURSOR_HAIRCURSOR.
% Compiled module: PAPCO_CURSOR_RESET.
% Compiled module: PAPCO_SLICE_VALUE.
```

```
% Compiled module: PAPCO_EDIT_TYPE.
% Compiled module: PAPCO_EDIT_TYPE_EVENT.
% Compiled module: PAPCO_CLEAR_DRAWING.
% Compiled module: PAPCO_PRINTERDESCRIPTION.
% Compiled module: PAPCO_CANYOUPLOT.
% Compiled module: PAPCO_DRAWPLOTS.
% Compiled module: PAPCO_PRINTPLOTS.
% Compiled module: PAPCO_DRAW_CHECKRESULT.
% Compiled module: PAPCO_DRAW_ERRORPANEL.
% Compiled module: PAPCO_DRAW_IDENTSTAMP.
% Compiled module: PAPCO_DRAW_BORDER.
% Compiled module: PAPCO_CHECK_PLOTINFOS.
% Compiled module: PAPCO_CLEAR_PANEL.
% Compiled module: PAPCO_DRAWPLOTS_DRAW.
% Compiled module: PAPCO_SETORBIT.
% Compiled module: SET_MANUAL_TIME.
% Compiled module: PAPCO_ORBIT_SELECTED.
% Compiled module: PAPCO_SETTIME.
% Compiled module: PAPCO_DRAWTIMERANGE.
% Compiled module: PAPCO_SAVESTATE.
% Compiled module: PAPCO_LOADSTATE.
% Compiled module: PAPCO_RESETSTATE.
% Compiled module: PAPCO_READPRINTERCONFIG.
% Compiled module: PAPCO_READDEFAULTSETTINGS.
% Compiled module: PAPCO_WRITEDEFAULTSETTINGS.
% Compiled module: PAPCO_BATCH.
% Compiled module: PAPCO_EDIT_BATCH.
% Compiled module: PAPCO_EDIT_BATCH_EVENT.
% Compiled module: PAPCO_SAVE_COMPILED.
% Compiled module: PAPCO_SAVE_CURRENT_STATE.
% Compiled module: PAPCO_LOAD_PREVIOUS_STATE.
% Compiled module: PAPCO_LOAD_LAST_STATE.
% Compiled module: PAPCO_DESCRIPTIONFOR.
% Compiled module: PAPCO_SETPLOTDATA.
% Compiled module: PAPCO_EDITPLOT.
% Compiled module: PAPCO_DELETEPLOT.
% Compiled module: PAPCO_CHECK_DEPLO_TIME.
% Compiled module: PAPCO_ADD_PLOT.
% Compiled module: PAPCO_STRIPEMPTYSLOTS.
% Compiled module: PAPCO_RESETSLOTS.
% Compiled module: PAPCO_RESORTSLOTS.
% Compiled module: PAPCO_ORDERSLOTS.
% Compiled module: PAPCO_DEFAULTS_EVENT.
% Compiled module: PAPCO_DEFAULTS.
% Compiled module: PAPCO_GET_REMOTE_HOST_NAME.
% Compiled module: PAPCO_GET_REMOTE_DATA.
% Compiled module: PROCESS_ACTIVE.
% Compiled module: DOWNLOAD_STATUS_GET_FILENAME.
% Compiled module: DOWNLOAD_STATUS_GET_FILELENGTH.
% Compiled module: DOWNLOAD_STATUS_GET_BYTESREAD.
% Compiled module: LIMIT_DOWNLOAD.
% Compiled module: DOWNLOAD_STATUS.
% Compiled module: PAPCO_GET_REMOTE_DATA_EVENT.
% Compiled module: PAPCO_GET_REMOTE_FILE_LIST_EVENT.
% Compiled module: PAPCO_GET_REMOTE_FILE_LIST.
% Compiled module: PAPCO_GET_REMOTE_ACCOUNT_INFO.
% Compiled module: PAPCO_GET_REMOTE_ACCOUNT_INFO_EVENT.
% Compiled module: PAPCO_UNFORMAT_IDLSTRING.
% Compiled module: PAPCO_WRITE_DATA.
% Compiled module: PAPCO_ERROR_HANDLER.
% Compiled module: PAPCO_GETPLOTINFOSTRUCT.
% Compiled module: PAPCO_GETDRAWNPLOTINFOSTRUCT.
% Compiled module: PAPCO_PRINTMENU_EVENT.
% Compiled module: PAPCO_MAIN_EVENT.
% Compiled module: PAPCO_REFRESH.
% Compiled module: PAPCO_REFRESHLIST.
% Compiled module: PAPCO_INPUT.
% Compiled module: PAPCO.

--- General Plot Initialization -------------------------------------
% Compiled module: PANELSET.
% Compiled module: CNVRTTIM.
% Compiled module: MLTTICKS.
% Compiled module: ILATTICKS.
% Compiled module: LSHELTICKS.
% Compiled module: RADTICKS.
% Compiled module: TIMETICKS.
% Compiled module: TIMETICKS_HR.
% Compiled module: NOFIRST2.
```

```
% Compiled module: NOTICKS.
% Compiled module: ORBITTICKS.
% Compiled module: NOFIRST1.
% Compiled module: MAGTICKS.
% Compiled module: CRRES_ORB_TICKS.
% Compiled module: CRRES_MLT_TICKS.
% Compiled module: NOZERO.
% Compiled module: NOZERO2.
% Compiled module: NOZERO3.
% Compiled module: MJDT_DATE_1.
% Compiled module: MJDT_DATE_2.
% Compiled module: MJDT_DATE_3.
% Compiled module: MJDT_DATE_4.
% Compiled module: MJDT_DATE_5.
% Compiled module: MJD_DEC_1.
% Compiled module: MJD_DEC_1A.
% Compiled module: MJD_DEC_2.
% Compiled module: TIME_LABEL.
% Compiled module: XTICKS.
% Compiled module: XTICKSDATE.
% Compiled module: XTICKSDATE2.
% Compiled module: XTICKSDATE3.
% Compiled module: YLIM.
% Compiled module: YLIM2.
% Compiled module: YLIM3.
% Compiled module: YLIM4.
% Compiled module: SET_DEVICE.

--- Color plot routines for papco -----------------------------------
% Compiled module: AUTORANGE.
% Compiled module: AUTORANGE_LP.
% Compiled module: SETMAXMIN.
% Compiled module: YRES.
% Compiled module: CALCULATEHEIGHT.
% Compiled module: PLOT_CLR.
% Compiled module: PLOT_CLR_ALT.
% Compiled module: GCD1.
% Compiled module: GCD.
% Compiled module: PAPCO_HPC_AV.
% Compiled module: PAPCO_PLOT_COLORSPEC.
% Compiled module: PAPCO_COLORBAR.
% Compiled module: PAPCO_COLORBAR_LOG_TICKS.
% Compiled module: PAPCO_AUTORANGE.

--- End papco_init: PAPCO Core compiled ------------------------------

--- Compile add-in modules: only those found in module paths: --------

% papco_find_modules: The following module directories were found:

/n/leadbelly/belly1/home/papco/papco/papco_modules/papco_event/
/n/leadbelly/belly1/home/papco/papco/papco_modules/papco_text/
/n/leadbelly/belly1/home/papco/papco/papco_modules/plot_type/

% papco_find_modules: Writing file
  /n/toaster/u/friedel/papco_user/papco_add_init.pro

% papco_add_init: Compiling found modules:

  --> for papco events ----------------------------------------------
     --> read and plot routines
% Compiled module: R_PAPCO_EVENT.
% Compiled module: READUSEREVENTS.
% Compiled module: P_PAPCO_EVENT.
     --> papco support routines
% Compiled module: PAPCO_EVENT_PANEL.
% Compiled module: PAPCO_EVENT_PANEL_EVENT.
% Compiled module: ADDPLOTTYPE_PAPCO_EVENT.
% Compiled module: ADDPANEL_PAPCO_EVENT.
% Compiled module: PLOTTABLE_PAPCO_EVENT.
% Compiled module: DESCRIPTOR_PAPCO_EVENT.
% Compiled module: EDITOR_PAPCO_EVENT.
% Compiled module: DRAW_PAPCO_EVENT.
  --> for papco text plots ------------------------------------------
     --> read and plot routines
% Compiled module: P_PAPCO_TEXT.
% Compiled module: R_PAPCO_TEXT.
     --> papco support routines
% Compiled module: PAPCO_TEXT_PANEL.
```

```
% Compiled module: PAPCO_TEXT_PANEL_EVENT.
% Compiled module: ADDPLOTTYPE_PAPCO_TEXT.
% Compiled module: ADDPANEL_PAPCO_TEXT.
% Compiled module: PLOTTABLE_PAPCO_TEXT.
% Compiled module: CANGET_PAPCO_TEXT.
% Compiled module: DESCRIPTOR_PAPCO_TEXT.
% Compiled module: EDITOR_PAPCO_TEXT.
% Compiled module: DRAW_PAPCO_TEXT.
  --> for plot type plots -----------------------------------------
     --> read and plot routines
% Compiled module: R_PLOT_TYPE.
% Compiled module: R_PLOT_TYPE_ONEFILE.
% Compiled module: R_PLOT_TYPE_ONEFILE_ORIGINAL.
% Compiled module: P_PLOT_TYPE_LINE.
% Compiled module: P_PLOT_TYPE_SPEC.
     --> papco support routines
% Compiled module: PLOT_TYPE_PANEL.
% Compiled module: PLOT_TYPE_PANEL_EVENT.
% Compiled module: ADDPLOTTYPE_PLOT_TYPE.
% Compiled module: ADDPANEL_PLOT_TYPE.
% Compiled module: PLOTTABLE_PLOT_TYPE.
% Compiled module: CANGET_PLOT_TYPE.
% Compiled module: DESCRIPTOR_PLOT_TYPE.
% Compiled module: EDITOR_PLOT_TYPE.
% Compiled module: DRAW_PLOT_TYPE.
% Compiled module: DRAWSLICE_PLOT_TYPE.
% Compiled module: LEVELZERO_PLOT_TYPE.
% Compiled module: WRITESLICE_PLOT_TYPE.

% papco_add_init: done

--- End papco_compile ---------------------------------------------

--- Environmental variables now set: --------------------------------

PAPCO_QUIET:        0
PAPCO_STARTUP_MODE: COMPILE
PAPCO_ADDINS:       NO
PAPCO_HOME:         /n/toaster/u/friedel/papco_user
PAPCO_PATH:         /n/leadbelly/belly1/home/papco/papco/papco
PAPCO_MODULES:      /n/leadbelly/belly1/home/papco/papco/papco_modules
PAPCO_DOWNLOAD:     /n/toaster/u/friedel/papco_user/papco_download_temp
TIME_CONV:          /n/leadbelly/belly1/home/papco/papco/papco_lib/cdssw/time
POLAR_ORBITS:       /n/papco/data/polar/eph
CRRES_ORBITS:       /n/papco/data/crres/eph
USER_MOUSE_ACTIONS:
                    View_Polar_Slice
                    View_Ephemeris
                    Distribution
USER_SHOW_ACTIONS:
                    hair_cursor
                    hair_cursor
                    vert_line

--- Done papco_set_variables -----------------------------------------
IDL>
```

Once your setup is static and has been saved (using the "Save & Done" way of exiting PAPCO), you can use the "Restore on startup" and "Quiet compile on startup"option and bypass the configuration widget by using `idl papco_startup`. Things should go much quicker, and the output is much shorter: (papco_sample_restore_startup.text)

```
papco_user% idl papco_setup
IDL Version 5.2 (sunos sparc). Research Systems, Inc.
Installation number: 679-1.
Licensed for use by: NIS, LANL

For basic information, enter "IDLInfo" at the IDL> prompt.


=====================================================================
--- W E L C O M E   T O   P A P C O ! --------------------------------
=====================================================================

--- Start papco_set_variables ----------------------------------------
```

```
--- Expanding IDL search path for PAPCO core, Lib, Home --------------

--- Compile PAPCO routines needed for startup ------------------------

======================================================================
--- papco - Panel Plot Composer - version 9.0, 20 Aug 1998 -----------
======================================================================

--- Environmental variables now set: ---------------------------------

PAPCO_QUIET:        0
PAPCO_STARTUP_MODE: COMPILE
PAPCO_ADDINS:       NO
PAPCO_HOME:         /n/toaster/u/friedel/papco_user
PAPCO_PATH:         /n/leadbelly/belly1/home/papco/papco/papco
PAPCO_MODULES:      /n/leadbelly/belly1/home/papco/papco/papco_modules
PAPCO_DOWNLOAD:     /n/toaster/u/friedel/papco_user/papco_download_temp
TIME_CONV:          /n/leadbelly/belly1/home/papco/papco/papco_lib/cdssw/time
POLAR_ORBITS:       /n/papco/data/polar/eph
CRRES_ORBITS:       /n/papco/data/crres/eph
USER_MOUSE_ACTIONS:
                    View_Polar_Slice
                    View_Ephemeris
                    Distribution
USER_SHOW_ACTIONS:
                    hair_cursor
                    hair_cursor
                    vert_line

--- Done papco_set_variables -----------------------------------------

--- Running papco Configure widget -----------------------------------

% Compiled module: XMANAGER.

--- Done papco Configure ---------------------------------------------

======================================================================
--- W E L C O M E   T O   P A P C O ! --------------------------------
======================================================================

--- Start papco_set_variables ----------------------------------------

--- Expanding IDL search path for PAPCO core, Lib, Home --------------

--- Compile PAPCO routines needed for startup ------------------------

======================================================================
--- papco - Panel Plot Composer - version 9.0, 20 Aug 1998 -----------
======================================================================

--- Environmental variables now set: ---------------------------------

PAPCO_QUIET:        1
PAPCO_STARTUP_MODE: RESTORE
PAPCO_ADDINS:       NO
PAPCO_HOME:         /n/toaster/u/friedel/papco_user
PAPCO_PATH:         /n/leadbelly/belly1/home/papco/papco/papco
PAPCO_MODULES:      /n/leadbelly/belly1/home/papco/papco/papco_modules
PAPCO_DOWNLOAD:     /n/toaster/u/friedel/papco_user/papco_download_temp
TIME_CONV:          /n/leadbelly/belly1/home/papco/papco/papco_lib/cdssw/time
POLAR_ORBITS:       /n/papco/data/polar/eph
CRRES_ORBITS:       /n/papco/data/crres/eph
USER_MOUSE_ACTIONS:
                    View_Polar_Slice
                    View_Ephemeris
                    Distribution
USER_SHOW_ACTIONS:
                    hair_cursor
                    hair_cursor
                    vert_line

--- Done papco_set_variables -----------------------------------------

--- Start papco_startup ----------------------------------------------

--- Module Paths and module data path env. var. settings -------------

Adding papco event module...
```

```
Adding papco text module...
Adding plot type module...

--- Start papco_restore ---------------------------------------------

--- Restoring procedures from --------------------------------------
    /n/toaster/u/friedel/papco_user/papco_routines.save

% papco_find_modules: The following module directories were found:

/n/leadbelly/belly1/home/papco/papco/papco_modules/papco_event/
/n/leadbelly/belly1/home/papco/papco/papco_modules/papco_text/
/n/leadbelly/belly1/home/papco/papco/papco_modules/plot_type/

% papco_find_modules: Writing file
  /n/toaster/u/friedel/papco_user/papco_add_init.pro

--- End papco_restore -----------------------------------------------

--- Environmental variables now set: --------------------------------

PAPCO_QUIET:        1
PAPCO_STARTUP_MODE: RESTORE
PAPCO_ADDINS:       NO
PAPCO_HOME:         /n/toaster/u/friedel/papco_user
PAPCO_PATH:         /n/leadbelly/belly1/home/papco/papco/papco
PAPCO_MODULES:      /n/leadbelly/belly1/home/papco/papco/papco_modules
PAPCO_DOWNLOAD:     /n/toaster/u/friedel/papco_user/papco_download_temp
TIME_CONV:          /n/leadbelly/belly1/home/papco/papco/papco_lib/cdssw/time
POLAR_ORBITS:       /n/papco/data/polar/eph
CRRES_ORBITS:       /n/papco/data/crres/eph
USER_MOUSE_ACTIONS:
                    View_Polar_Slice
                    View_Ephemeris
                    Distribution
USER_SHOW_ACTIONS:
                    hair_cursor
                    hair_cursor
                    vert_line

--- Done papco_set_variables ----------------------------------------
IDL>
```

A.2 PAPCO startup log

Once you are in IDL and all PAPCO modules have been compiled, you start PAPCO by simply typing `papco`. Here is the output generated by PAPCO when it starts. What you should see when you start PAPCO is a similar list depending on the modules you have loaded: (papco_sample_startup.text)

```
IDL> papco

% r_module_config: papco_event settings have been read.
% r_module_config: papco_text settings have been read.
% r_module_config: plot_type settings have been read.
% papco_files: Now reading font default settings
% papco_files: Now reading error catch default settings

;****************************************************************************
;* Adding buttons for modules
;*       71   addplottype_papco_event
;*       75   addplottype_papco_text
;*       79   addplottype_plot_type
;****************************************************************************
;* checking for sub plot_types
;****************************************************************************
;* file papcoadd_papco_event
;*      papcoadd_papco_event
;* end
;****************************************************************************
;* file papcoadd_papco_text
;*      papcoadd_papco_text
;* end
;****************************************************************************
;* file papcoadd_plot_type
```

```
;*      papcoadd_plot_type
;* end
;****************************************************************************
;* All Add-in modules read
;****************************************************************************

% papco: Added user slice USER View Polar Slice          slice =      32
% papco: Added user slice USER View Ephemeris            slice =      64
% papco: Added user slice USER Distribution              slice =     128

% papco_files: Reading printer configuration file
  /n/toaster/u/friedel/papco_user/papco_printer_config.dat
  Added printer: Phaser 550 w108
      printcall: lpr -P w108rgb
  Added printer: Phaser 550 w108 (transparency)
      printcall: lpr -P w108rgb_t
  Added printer: Phaser 550 n114
      printcall: lpr -P n114rgb
  Added printer: Phaser 550 n114 (transparency)
      printcall: lpr -P n114rgb_t
  Added printer: Phaser 450 4-color (CMYK) dye-sublimation s110
      printcall: lpr -P s110dye
  Added printer: Phaser 450 4-color (CMYK) dye-sublimation s110 (transparency)
      printcall: lpr -P s110dye_t
  Added printer: Dye Sublimation CR
      printcall: lpr -P n114dye
  Added printer: QMS 410 (laser b&w) w108a
      printcall: lpr -Pw108abw
% papco_files: Reading all default settings from file
  /n/toaster/u/friedel/papco_user/papco_default.dat
% papco_files: No last state saved
```

A.3 PAPCO finishing log

When you exit PaPCo a default file is written and any temporary plot files are deleted. You should see something like this: (papco_sample_finishup.text)

```
% papco: Cleaning up

% papco_files: Writing default settings to file:
%              /n/toaster/u/friedel/papco_user/papco_default.dat

% papco: Terminated normally
```

Appendix B

Templates for PaPCo

The PaPCo distribution contains a template of a standard module directory structure with all the files needed to construct a simple module, to make the design of new modules easier. This template resides in the PaPCo module directory, in `papco_XX/papco_modules/plot_type`.

The general idea is that you copy the template directory structure, rename all the occurrences of `plot_type` with the name of your own module (including the directory name, the filename and the contents of the files) and then add any specifics you need for your own plot type.

If you're working under UNIX and have the `sed` utility available, you can use the PaPCo routine `make_new_module.pro` to automatically create a correctly named module directory & file structure for you (see Section 6.2).

The templates are extensively commented in the files themselves (any line starting with a semicolon). It is probably a good idea to have a look at some other simple existing modules too, such as the OMNI module (in `papco_XX/papco_modules/omni`).

The template as it stands is a fully functioning module which works with one single data file included in the module (for 11. January 1997). When you install PaPCo this template module is installed along with it and you can play with it to test out its functionality by setting the time to 11.01.97 and pressing the plot_type button to add some panels. Remember to set the correct data path using the panel editor config button. The data file will be fetched using the remote get data facility.

Included with the distribution are some sample product files. You can load these using the LOAD button, those whose name includes "plot_type" are for the plot_type module template.

This template does not include examples of slices beyond the simple zoom functionality: this will be added for a later release...

The `plot_type_panel.help` template (Section B.1.4) is not an IDL routine put a sample help file which is called by the panel editor help button. It resides in the module directory for the given module in the subdirectory `papco_interface`.

Please write such a help file for your module using this template, it is of great help to potential other users of your module.

As the use of these templates is quite pivotal to the successful customization of PaPCo *by any user their quality and utility are very important. If you have any comments or suggestions or criticisms about these templates and would like to suggest improvements, please contact the author of this document.*

B.1 PAPCO interface templates

These routines are needed to interface your module into PAPCO .

B.1.1 plot_type_panel.pro

This file contains the panel editor widget for `plot_type`.

```
;******************************************************************************
;* FILE: plot_type_panel.pro
;*
;* DESCRIPTION:
;*    This is a demonstration on how to implement a papco-panel-editor.
;*    It makes use of all the features provided by $papco_XX/papco_panel.pro
;*    (a collection of standard papco widgets). It constructs an editor for a
;*    very simple papco-plot, that has all the default-functionality.
;*
;* COMMON-BLOCKS:
;*    plot_type_panel_data   contains the window-information
;*
;* STRUCTURES:
;*    This widget inherits a PlotInfo structure as an argument. This
;*    structure is then modified by the user according to his/her needs:
;*    Either by including standard PAPCO widgets from papco_panel.pro,
;*    which extend this structure, or by modifying its tags. The modified
;*    structure is then handed back to PAPCO.
;*
;*    The PlotInfo structure is created by a call to papco_getPlotInfoStruct
;*    and contains a series of tags which are used to describe a given panel.
;*    These tags come in three categories:
;*       1. Tags used by PAPCO internally - don't touch!
;*       2. Tags added by including standard PAPCO widgets. You can edit
;*          or preset these tags if you wish.
;*       3. User Tags for control of user-specified features. There
;*          are enough user tags specified to handle most needs.
;*    The main limitation of the User Tags is that you're stuck with
;*    what has been defined in PAPCO. If you REALLY need additional tags,
;*    this entails changing the PAPCO core, and should be done by
;*    whoever maintains the core. You can do this yourself too, by editing
;*    pro papco_getPlotInfoStruct contained in $papco_XX/papco/papco.pro
;*    but your changes will then not propagate to the next PAPCO version.
;*
;*    The PAPCO 09 set of user tags is:
;*
;*       typeVector:intarr(4), $ ; user keyword (old, for compatibility)
;*       switch:lonarr(3),     $ ; user keyword (old, for compatibility)
;*       options:fltarr(20),   $ ; user keyword (float options)
;*       ioptions:lonarr(16),  $ ; user keyword (long options)
;*       soptions:strarr(16)   $ ; user keyword (string options)
;*
;*    NB! If you need more variables to commincate between your panel
;*    editor and your plot routine then you can use your own common block
;*
;*    For a more complete description of the other PlotInfo tags and the
;*    tags used by PAPCO supplied widget refer to
;*    $papco_doc/papco_variables.text
;*
;* OVERVIEW:
;*    This function creates the editor-window:
;*       FUNCTION  plot_type_panel, aPlotInfo, GROUP=group, ACTION=ACTION
;*    To test you might try this to get a stand-alone widget:
;*       a=papco_getPlotInfoStruct()   which creates the structure
;*                                     containing the plot information
;*  a=plot_type_panel(a)          which calls up the panel window.
;*    Interrogating a with help,a,/str shows the contents.
;*
;* MODIFICATION HISTORY:
;*     august 1995, written by A.Keese
;*     december 1995, modified by R.Friedel at Goddard
;*     april 1996, modified by R.Friedel
;*     march 1997, modified by R.Friedel
;*     August 1998, modified by R.Friedel
;******************************************************************************
```

```
;**************************************************************************
;* FUNCTION:
;*      FUNCTION  plot_type_panel, aPlotInfo, GROUP=group, ACTION=ACTION
;*
;* DESCRIPTION:
;* Open an editor widget for a panel, using the methods defined in the
;* file $papco_XX/papco_panel.pro .
;* When the user presses the 'Done'- or the 'Cancel'-Button, the values
;* changed by the event-methods of papco_panel.pro are returned.
;* The returned value is a structure containing those values and a flag,
;* 'canceled', which the caller may read out to check, whether the
;* window was closed using the 'Cancel' or the 'Done'-Button.
;*
;* INPUTS:
;*      aPlotInfo       a structure of the papco_plotinfo-type
;*                      this structure is edited in the panel editor window.
;* KEYWORDS:
;*      GROUP           a WIDGET_BASE
;*                 when specified, this is used as GROUP_LEADER
;*      ACTION          a string
;*                 when specified, the 'Done'-Button is labeled with it
;*         if not specified, the 'Done'-Button is labeled 'Done'
;*
;* OUTPUT:
;*      a default-structure of type papco_panel_editor_return. Though this
;*      type is nowhere defined, it is a virtually existing data-type.
;*      All the functions in $papco_XX/papco.pro that edit plots or add new
;*      ones, expect the opened editor-windows to return a structure containing
;*      at least some fields. Typically this structure contains fields
;*      used by the papco_panel widgets, and some fields "inherited"
;*      from aplotinfo but modified by the user.
;*
;* CALLING SEQUENCE:
;*      newPlotData=papco_PANEL_TEMPLATE(aplotInfo, ACTION=, GROUP=)
;**************************************************************************
FUNCTION  plot_type_panel, aPlotInfo, GROUP=group, ACTION=ACTION

  COMMON plot_type_panel_data, widgetData

;--- if GROUP is not specified, the group-leader will be the desktop
  IF NOT KEYWORD_SET(GROUP) THEN GROUP=0

;--- set the label for the action-button
  IF NOT KEYWORD_SET(ACTION) THEN ACTION='Done'

;--- the routines used by papco_panel.pro define tags for a structure. Those
;    tags will be stored in the structure 'additionalWData'. When you realize
;    the window, you have to add your own window-specific data to that
;    structure and afterwards assign it to widgetData.
  additionalWData={canceled:1}

;--- create the base-window ----------------------------------------------------
  panel_base=papco_widget_base(title=aPlotInfo.panelKind, column=1, /modal, $
                               GROUP_LEADER=group)

;--- All papco_panel_add routines are papco panel widgets that can be included
;    in the panel editor or not. They provide functionality that papco
;    supports. They dynamically extend the panel editors widget structure, and
;    all the events they generate are supported by papco_panel_event,
;    transparent to the user.

;--- the first part of the window will consist of the widget that -------------
;    controls the height of that panel. This is needed for all panel editors.
  widgets=papco_panel_AddHeightWidgets(panel_base, aPlotInfo)
  additionalWData=create_struct(widgets, additionalWData)


;
;---  <<<<<<<<<<  insert your widgets below here!  >>>>>>>>>
;
  type_base=WIDGET_LABEL(panel_base, $
  VALUE='***           PLACE OWN WIDGETS HERE             ***')

;--- this example uses a simple slider to set channel number. The channel
;    number is passed out in the  typevector(0) tag of structure aplotInfo.
;    channel numbers go from 1 to 10, if 0 is chosen the data is plotted in the
;    form of a color spectrogram.

  channel_base=WIDGET_BASE(panel_base, /FRAME, ROW=1)
  sl_channel=WIDGET_SLIDER(channel_base, XSIZE=450, /DRAG, $
```

```
                         TITLE=strcenter('channel number',75), $
                         VALUE=aPlotInfo.typevector(0) > 0 < 10, $
                         UVALUE='sl_channel',$
                         MINIMUM=0, MAXIMUM=10)

  type_base=WIDGET_LABEL(panel_base, $
  VALUE='*** 1 - 10 chooses Channel, 0 a color spectrogram ***')
  type_base=WIDGET_LABEL(panel_base, $
  VALUE='***              END OWN WIDGETS                 ***')
;
;---  <<<<<<<<<<  insert your widgets above here!  >>>>>>>>>
;

;--- for examples of standard buttons or sliders look at other panel editors.
;    some standard buttons which have been recycled before can be found in
;    papco_XX/papco_lib/more_panel_widgets.

;--- As the same panel is called when you edit a plot, interrogate the tags in
;    the plotInfoStruct (aplotInfo) passed in and set your widgets accordingly.
;    In this way the widget is always reinitialized to its last state.

;--- The next five papco panel widgets will be added only if you want to
;    support that functionality, and if they make sense for your
;    plot_type. If you do not want to use them, just comment them out.

;--- some panels might need to read/input to text files (event, text ----------
;    module). So a generic papco widget is supplied for this.
  widgets=papco_panel_AddFile_Widgets(panel_base, aPlotInfo)
  additionalWData=create_struct(widgets, additionalWData)
;--- add the papco panel widget for choosing the color bar for color plots ----
  widgets=papco_panel_Addcolorbar_Widgets(panel_base, aPlotInfo)
  additionalWData=create_struct(widgets, additionalWData)
;--- add the  papco panel widget for choosing the plot-style for line plots ---
  widgets=papco_panel_add_PlotStyleButtons(panel_base, aPlotInfo, 0)
  additionalWData=create_struct(widgets, additionalWData)
;--- add the papco panel widget for choosing the y-axis-ranges ----------------
  widgets=papco_panel_AddYScale_Widgets(panel_base, aPlotInfo)
  additionalWData=create_struct(widgets, additionalWData)
;--- add the papco panel widgets for choosing the z-axis-ranges ---------------
  widgets=papco_panel_AddZScale_Widgets(panel_base, aPlotInfo)
  additionalWData=create_struct(widgets, additionalWData)

;--- finally, the Actionbuttons (help, cancel, done) are added to the ---------
;    editor window. This is needed for all panel editors. If you want to
;    make use of the configure feature, then add the keyword CONFIG
  widgets=papco_panel_AddActionButtons(panel_base, Action, aPlotInfo, /CONFIG)
  additionalWData=create_struct(widgets, additionalWData)

;--- now, create the structure that keeps the panel-window data. You always
;    need to specify the help-file, and your structure must contain the
;    'additionalWData'. You should create the helpfile in the papco_interface
;    directory of your module (a template is in the plot_type example module)
;    Also add the ID's of your widgets so that you can perform widget_control
;    actions in the events routine or in a refresh routine if needed.
  thisWidgetsData={helpfile:'plot_type_panel.help', $
                   typevector:aPlotInfo.typevector}
  widgetData=papco_panel_CreateWidgetData(thisWidgetsData, additionalWData)

;--- realize the window -------------------------------------------------------
  papco_Panel_Realize, panel_base, widgetData
  papco_xmanager, 'plot_type_panel', panel_base

;--- the returned data consists of the standard-data inherited from
;    papco_panel.pro and your new data. Stick to the tags provided in
;    aPlotInfo, there are ample variables for the user
  returnData={typevector:widgetData.typevector}
  additionalRData=papco_panel_ReturnData(widgetData)

  return, create_struct(ReturnData, additionalRData)

END

;******************************************************************************
;* PROCEDURE:
;*      PRO plot_type_panel_event, event
;*
;* DESCRIPTION:
;*      This processes events from the XMANAGER.
;*      As this example does not extend the basic functionality of plot-
```

```
;*      editors, it directly calls the default-handler, papco_PANEL_Event
;*
;* INPUTS:
;*      event            an XManager event
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      called by XMANAGER
;******************************************************************************
PRO plot_type_panel_event, event

  COMMON plot_type_panel_data, widgetData

  WIDGET_CONTROL,GET_UVALUE=uval, event.id

; in this editor, we only need to process the channel slider events. events
; other than those from your own widgets need to be passed on to
; 'papco_panel_Event', which handles the events generated
; by the papco-provided panel widgets. Your own added panel widgets events
; need to be processed here:  you should check 'event' for your events and call
; 'papco_panel_Event', if event was not generated by your own panel widgets.
; The best way to learn is to look at some of the other module's panel editors.

  if uval eq 'sl_channel' then begin
    widgetData.typevector(0)=event.value
  endif else begin
    widgetData=papco_panel_Event(event, widgetData)
  endelse

END
```

B.1.2 papcoadd_plot_type.pro

This file contains all the interface routines between PAPCO and the module `plot_type`.

```
;******************************************************************************
;* file papcoadd_plot_type
;*      papcoadd_plot_type
;* end
;*
;* this files contains all the collected routines which have to be added in
;* for a plot_type.
;*
;* THIS IS A WORKING EXAMPLE MODULE WHICH CAN BE INCLUDED INTO papco AS IS
;* (it doesn't plot very exiting data though!)
;*
;* THIS TEMPLATE IS INTENDED AS THE STARTING POINT FOR DEVELOPING OWN MODULES.
;* COPY THE WHOLE plot_type DIRECTORY TREE AND RENAME EVERY OCCURRENCE OF
;* "plot_type" WITH YOUR MODULE NAME, IN ALL THE FILES AND FILENAMES.
;*
;* If the file starting with "papcoadd" exist in the module directory, then the
;* routines contained thereinare utilized to add that plot type into PAPCO.
;* This makes papco "configurable" by adding or leaving out modules.
;*
;* ALL that is needed to include the module is to include the path to the
;* module directory tree in the IDL search path. This is done automatically
;* by the PAPCO configure widget when you start PAPCO with
;*
;* idl papco_setup
;*
;* If the plot type is in itself a sub-menu containing more plot_types, then
;* these must be listed at the top of this file, as it is read by papco.
;* the entries for each sub-plot-type must come after the line
;* "file papcoadd_plot_type" and before the "end" line, one entry per line.
;* Some of the routines in this file must then be duplicated for each
;* sub-plot-type used: see some of the exisiting modules for an example.
;*
;* Apart from editing this template the user has to write a panel
;* editor widget for his plot_type. Use the file
;*    $papco_XX/papco_modules/plot_type/papco_interface/plot_type_panel.pro
;* as the template.
;*
;* If slice functions are to be supported a slice program needs to be written
;* too. Use the template file
;*    $papco_XX/papco_modules/plot_type/papco_interface/plot_type_slice.pro.
;*
;* Finally, a data read and data plot routine are required as well. The data
;* read routine will be specific for each plot_type. An example for a standard
;* line plot is given in the file
;*    $papco_XX/papco_modules/plot_type/plot_read/p_data_type_line.pro
;* and for a standard color spectrogram in
;*    $papco_XX/papco_modules/plot_type/plot_read/p_data_type_spec.pro.
;* These templates can also be used as a basis for more complex plots.
;*
;*
;* The Way these add-in routines work:
;* ===================================
;*
;* On initializing IDL for PAPCO, the routine find_modules.pro is used to
;* parse the IDL search path, and all the init_module.pro batch files found in
;* the module sub-directories \papco_interface are written to the
;* papco_add_init.pro batch file and then executed: In this way the user can
;* specify all the routines that need to be compiled for his module at startup.
;*
;* On starting papco, all module sub-directories \papco_interface are searched
;* for a file beginning with "papcoadd_". From the names of this file the
;* plot_type name is stripped off and used to construct all the procedure
;* names contained in the file. The first routine, "addplottype_plot_type", is
;* used to add the button for this plot_type to the papco main window. Since
;* "addplottype_plot_type" simply returns widget ID of the button, the button
;* itself can contain further sub-menus for plot sub-types.
;*
;* To inform papco of all these sub-plot types (and to avoid having to write a
;* separate file for each sub-plot type) all the types have to be listed in the
;* header of the file papcoadd_plot_type.pro. These entries have to finish off
;* with the line ";* end". All plot_type names are stored in the string array
;* plot_types and are placed in a common block:
;*
;*    COMMON ADDINS, plot_types
```

```
;*
;* If there are no sub-plot types, then the plot_type as used in the file name
;* for papcoadd_plot_type.pro is used to construct the calls to the procedures
;* in that file. The convention for naming these procedures is
;* "procedure_name_plot_type" , so that papco can always construct the right
;* procedure call when needed.
;*
;* When there are sub-plot types, their names get used to construct the
;* required calls. There are 6 procedures that need to be called for each
;* plot_type, so that a set of these 6 procedures has to be written for each
;* sub-plot type. In the same way, you need to write a panel editor for each
;* plot type (sometimes several plot sub-types may share the same editor,
;* e.g. multiple line plots or color spectra, see papcoadd_crres_plots).
;*
;* In PAPCO, whenever an action for a plot_type is taken, the information
;* stored in the structure widgetData is queried. In this structure there
;* is an array of PlotInfo structures, which contain all the relevant
;* information for a chosen panel. The field plotInfo.panelkind is loaded
;* with the name of the plot_type. This is set by the user in this file in
;* the routine addpanel_plot_type ( "panelKind='plot_type'" ). THIS MUST
;* BE SET CORRECTLY AS ALL THE ROUTINE NAMES USED IN THIS FILE ARE
;* CONSTRUCTED USING THIS INFORMATION!
;*
;* Whenever papco has to undertake some action for a plot_type, the field
;* plotInfo.panelkind is compared to the entries in the (COMMON ADDINS)
;* plot_types array. If a match is found, the required procedure name is
;* constructed and called If not, an error occurs.
;*
;*    These procedures are contained:
;*
;*    PRO addplottype_plot_type,plottypes_base,pb_plottype
;*        ->  adds the button for the new plot type to the main window.
;*
;*    PRO plottable_plot_type, plottable
;*        -> Sets a variable to indicate plottable without orbit.
;*
;*    PRO canget_plot_type, canget
;*        -> Sets a variable to indicate output to file is supported.
;*
;*    PRO descriptor_plot_type, plotInfo, descriptStr
;*        -> Constructs the string used to label plots in the panels window
;*
;*    PRO addpanel_plot_type
;*        -> Calls the panel editor for this plottype and adds the plot to the
;*            panel window.
;*
;*    PRO editor_plot_type, plotNumber, newPlotData
;*        -> This procedure calls the panel editor when called from
;*           papco_EditPlot (papco_paneledit.pro).
;*
;*    PRO  draw_plot_type, panelVector, currPlotInfo, get_data_call, IsRead
;*        -> This procedure performs the actual data read call and the plot
;*           call for this plottype
;*
;* MODIFICATION HISTORY:
;*     written November 1995, Reiner Friedel
;*     updated for OUTPUT functionality, June 1998, Reiner Friedel
;******************************************************************************
;******************************************************************************

;******************************************************************************
;* PROCEDURE:
;*      PRO addplottype_plot_type,plottypes_base,buttonGeometry,pb_plottype
;*
;* DESCRIPTION:
;*      This procedure adds a button for this plot_type to the plottypes_base.
;*      The UVALUE declared here is critical, since it is used to construct
;*      the procedure names of all procedures contained in this add-in
;*      package, so that they can be automatically called when needed.
;*      The button itself can contain access to a sub-menu for further
;*      plot_types.
;*
;* INPUTS:
;*      plottypes_base  the widget ID of the parent widget to contain the
;*                      plottype buttons.
;*
;* OUTPUTS:
;*      pb_plottype     the widget ID of the button created here.
;*      buttonGeometry  size of the button. Only used in simple
```

```
;*                      WIDGET_BUTTON to make them the same size as the
;*                      compound CW_PDMENU, pulldown for sub-types.
;*
;* KEYWORDS:
;*       none
;*
;* CALLING SEQUENCE:
;*      addplottype_plot_type,plottypes_base,buttonGeometry,pb_plottype
;******************************************************************************
pro addplottype_plot_type,plottypes_base,buttonGeometry,pdm_plot_type

; This example is for a simple button with a submenu.
; **NEW** from PAPCO 09 is the inclusion of the configure button as
; the first entry of the module submenu. Further entries into the
; submenu are easily added using
;
;        { CW_PDMENU_S, 0, ' plot_type submenu ' }, $
;
; The second entry in this structure defines the following:
;                       3   -   label for the button
;                       0   -   submenu entry
;                       2   -   last submenu entry
;
; The third entry in this structure is the name of the submenu entry.

  MenuDesc = [ $
        { CW_PDMENU_S, 3, strcenter('plot_type',10) }, $
        { CW_PDMENU_S, 0, ' configure plot_type ' }, $
        { CW_PDMENU_S, 2, ' plot plot_type      ' } ]

  pdm_plot_type = CW_PDMENU(plottypes_base, MenuDesc, $
                            UVALUE='pdm_plot_type')

; The name used for the submenu entries is user by PAPCO to construct
; the procedure/function calls needed for that part of the module.

; Given here is one submenu entry. IF YOU ADD FURTHER SUBMENU
; ENTRIES, THEN YOU WILL NEED TO WRITE ONE EACH THE
; plottable_plot_type, canget_plot_type, descriptor_plot_type,
; editor_plot_type AND draw_plot_type
; ROUTINES THAT FOLLOW FOR EACH SUBMENU ENTRY, MAKING SURE YOU ADHERE
; TO THE CORRECT NAMING CONVENTION.
;
; You will also have to add another case to the case statement in
; addpanel_plot_type.

END

;******************************************************************************
;* PROCEDURE:
;*      PRO addpanel_plot_type
;*
;* DESCRIPTION:
;*      Calls the panel editor for this plottype and adds the plot to the
;*      panel window.
;*      Using an instance of the papco_PLOTINFO structure called "defaults"
;*      the default settings of the panel editor can be controlled. The
;*      desired default values for the plotinfo tags need to be edited by
;*      the user. This plotinfo structure will be passed to the panel editor
;*      When it is first called for the plot_type.
;*      "panelEditor" must be set to the name of the panel editor routine.
;*
;* INPUTS:
;*      event           event value of papco main window. This is
;*                      only used if there are plot sub_types,
;*                      for example see papcoadd_crres_plots
;* OUTPUTS:
;*      none
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      addpanel_plot_type
;****************************************************************************
pro addpanel_plot_type,event

  common get_error, get_err_no, get_err_msg
```

```
; The case statement here needs an entry for every submenu type you
; added in  the first procedure in this file (addplottype_plot_type).
; For each submenu entry, you will need to specify the panelEditor to use.

; The FIRST time you use this module you will have to configure it,
; i.e. set the data paths etc. Once you have done this PAPCO will
; write a plot_type.config file into your papco_user_XX/papco_modules_defaults
; directory.
; So here we check if this plot_type.config file exists, and if not,
; we disallow the execution of any other option apart from "configure
; plot_type". In this way the user is forced to configure the module
; before using it the first time!

  r_module_config, strlowcase('plot_type'), config_data, msg
  if (get_err_no ne 0) and (event.value ne 1) then begin
    print,'%addpanel_plot_type: You need to configure the module first'
    papco_module_configure,'plot_type'
    return
  endif

  panelKind='plot_type'

  CASE event.value of
    1: BEGIN
        papco_module_configure, panelKind
    end
    2: BEGIN
        panelEditor=panelkind+'_panel'
        defaults=papco_getPlotInfoStruct() ;returns a basic plotinfo structure
        defaults.typeVector=[0,0,0,0]      ;set some tags to desired defaults
        defaults.channel=0
        papco_Add_Plot, panelEditor, panelKind, initialPlotInfo=defaults
    end
    else: begin
      print,'%addpanel_plot_type: Nothing defined for this submenu entry'
      return
    end
  endcase

END

;***************************************************************************
;* PROCEDURE:
;*      PRO plottable_plot_type, plottable
;*
;* DESCRIPTION:
;*      Sets a variable to indicate plottable with / without orbit.
;*      PAPCO provides "set time by orbit" feature for CRRES and POLAR
;*      so far. Included for compatibility with older CRRES modules
;*      which ONLY use orbit as input to their read/plot routines
;*      (plotting over orbit boundaries is not enabled for CRRES)
;*
;* NB!  You will need one of these routines for each submenu entry you
;*      add. The name of the routine MUST consist of "plottable_" plus
;*      the panel_kind you have set for your submenu entry in
;*      addpanel_plot_type.
;*
;* INPUTS:
;*      none
;*
;* OUTPUTS:
;*      plottable        a boolean 0 if plottable only by orbit
;*                                 1 if plottable by time also.
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;***************************************************************************
pro plottable_plot_type, plottable

   plottable=1     ; 1 -> plotable without orbit info, by time
                   ; 0 -> plotable only by orbit number not by time
END

;***************************************************************************
;* PROCEDURE:
;*      PRO canget_plot_type, canget
;*
```

```
;* DESCRIPTION:
;*      Sets a variable to indicate that plot_type support the OUTPUT
;*      functionality.
;*
;* NB!  You will need one of these routines for each submenu entry you
;*      add. The name of the routine MUST consist of "canget_" plus
;*      the panel_kind you have set for your submenu entry in
;*      addpanel_plot_type.
;*
;* INPUTS:
;*      none
;*
;* OUTPUTS:
;*      canget         a boolean 0 if this feature is not available
;*                               1 if this feature is available
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      canget_plot_type, canget
;***************************************************************************
pro canget_plot_type, canget

   canget=  1      ; 1 -> plot routine can return just data to file
                   ; 0 -> feature disabled
END

;***************************************************************************
;* PROCEDURE:
;*      PRO descriptor_plot_type, plotInfo, descriptStr
;*
;* DESCRIPTION:
;*      Constructs the string used to label plots in the panel window.
;*      The plotinfo structure passed in here is the panel's plotinfo
;*      as returned by the panel editor for this plot_type. You can use
;*      the information in that structure to construct some informative
;*      label for the panel which then appears in the Panel Compose list.
;*
;* NB!  You will need one of these routines for each submenu entry you
;*      add. The name of the routine MUST consist of "descriptor_" plus
;*      the panel_kind you have set for your submenu entry in
;*      addpanel_plot_type.
;*
;* INPUTS:
;*      plotInfo        structure containing all data for this plot.
;*                      (structure type: papco_PLOTINFO,
;*                                    see papco_variables.text)
;*
;* OUTPUTS:
;*      descriptStr     a string containing the label
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      descriptor_plot_type, plotInfo, descriptStr
;***************************************************************************
pro descriptor_plot_type, plotInfo, descriptStr

   if PlotInfo.typeVector(0) ne 0 then $
      descriptStr=descriptStr+'Lineplot Channel : ' + $
        strtrim(string(plotInfo.typevector(0)), 2) $
   else $
     descriptStr=descriptStr+'Spectrum plot'

END

;***************************************************************************
;* PROCEDURE:
;*      PRO editor_plot_type, plotNumber, newPlotData
;*
;* DESCRIPTION:
;*      This procedure calls the panel editor when called from
;*      papco_EditPlot (papco_paneledit.pro). Provided here to allow
;*      several plot_types to share a common panel editor. Normally this
;*      procedure does not need to be edited beyond the required name
;*      change for plot_type.
;*
```

```
;* NB!  You will need one of these routines for each submenu entry you
;*      add. The name of the routine MUST consist of "editor_" plus
;*      the panel_kind you have set for your submenu entry in
;*      addpanel_plot_type.
;*
;* INPUTS:
;*      plotNumber      anInteger. 0<=plotNumber<=widgetData.numberOfPlots
;*                      the index of the plot in widgetData.plotInfos.
;*
;* OUTPUTS:
;*      newPlotData     the new structure returned by the panel editor
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      editor_plot_type, plotNumber, newPlotData
;***************************************************************************
PRO editor_plot_type, plotNumber, newPlotData

  COMMON PLOT_COMPOSER, widgetData

  plotInfo=widgetData.plotInfos(plotNumber)
  newPlotData=plot_type_panel(plotInfo, $
                              ACTION='Update '+plotInfo.panelKind, $
                              GROUP=widgetData.main_base)

END

;***************************************************************************
;* PROCEDURE:
;*      PRO draw_plot_type, panelVector, currPlotInfo, orbitNo $
;*                          get_data_call, slice_type, IsRead
;* DESCRIPTION:
;*      This procedure performs the actual data read call and the plot call
;*      for this plottype. The user need to pass to his plot routine all the
;*      parameters needed, typically these parameters are stored in the
;*      currplotinfo structure or constructed from them. The currplotinfo
;*      structure tags normally used for passing information from the panel
;*      editor to the plot routine are:
;*          type_vector - a four element array for choosing the type of data
;*                        (eg. elec or ion, diff or integ., pitch angle,
;*                             or sector, channel ....)
;*          switch      - a three element vector for specific plot appearance
;*                        (eg. smoothing/averaging, interpolating ...)
;*          filename    - a filename
;*
;*      These tags have grown out of traditional usage and can be used for
;*      anything the user wants to use them for. There are plenty variables
;*      for most applications: Adding a new tag to the plotinfo structure
;*      is not user-definable but need to be done at the papco core level.
;*
;*      The panelVector MUST be passed to your plotroutine as it is used
;*      to set the panel position.
;*      The _EXTRA=extra_PlotInfo should be passed if you intend to make use
;*      of papco functionality such as overplotting, linestyle choice and
;*      color bar choice.
;*
;* NB!  You will need one of these routines for each submenu entry you
;*      add. The name of the routine MUST consist of "draw_" plus
;*      the panel_kind you have set for your submenu entry in
;*      addpanel_plot_type.
;*
;* INPUTS:
;*      panelVector    position of the plot (see papco_conventions.text, 6)
;*      currPlotInfo   structure containing all data for this plot.
;*                     (structure papco_PLOTINFO, see papco_variables.text)
;*
;* OUTPUTS:
;*      get_data_call  a string containing the data call. Used by other
;*                     routines that need to know how to get the data
;*                     in a panel, e.g. to draw spectra, distribution functions
;*      IsRead         a boolean which is set when the the data is read the
;*                     first time. Useful if a given data-read puts data
;*                     into commons which can be plotted in many ways.
;*                     When set, data does not have to be read again.
;   slice_type:   Selects which slices are supported and for which slice
;                 routines have been written.
;                 This is an integer logical with 16 switches. Currently
;                 there are 5 papco slices. You may extend this by
```

```
;                 further USER slices up to a total of 16 slices.
;                     0 = disable all slices
;                     1 = papco Zoom                 (2^0)
;                     2 = papco View Slice           (2^1)
;                     4 = papco Access level Zero    (2^2)
;                     8 = papco Write Paneldata      (2^3)
;                    16 = papco Hair Cursor          (2^4)
;                    32 = USER slice A or not used   (2^5)
;                    64 = USER slice B or not used   (2^6)
;                    "     "     "  "  "  "     "      "
;                 32768 = USER slice Z or not used  (2^15)
;
;   You can support more than one type of slice by arithmetically adding the
;   options, eg slice_type=1+4  support Zoom and Level Zero
;*
;* KEYWORDS:
;*      OUTPUT        if set then plot routine should output data to file,
;*                    as well as to screen.
;*
;* CALLING SEQUENCE:
;*      draw_plot_type, panelVector, currPlotInfo, orbitNo, $
;*                      get_data_call, slice_type, IsRead, NODATA=NODATA
;***************************************************************************
pro draw_plot_type, panelVector, currPlotInfo, orbitNo, $
                    get_data_call, slice_type, IsRead, $
                    OUTPUT=OUTPUT

  COMMON extra, extra_PlotInfo  ; passes in papco-set keywords structure

  get_data_call="r_plot_type"   ; data read call used by slices

; set parameter for plot routine. In this example, channel 1 -10 chooses the
; channel number for a line plot, channel=0 chooses a spectrum color plot.
  channel=currPlotInfo.typeVector(0)

; as papco dynamically allocates slice values for the user defined
; slices, one needs to be able to find out this assignment! Use the
; following function to find out the slice values for any slices you
; might have added for your module, so that you know which values to enable!

  slice_assigned_value=papco_slice_value('USER Defined Slice A')

; set up which mouse functions to allow
  slice_type=1+2+4+8+16

  IF NOT IsRead THEN BEGIN      ; check if data has already been read
    r_plot_type                ; read data
    ; check the read for success.
    IF papco_Draw_CheckResult(currPlotInfo.panelKind, $
                              panelVector, orbitNr, $
                              currPlotInfo.typeVector, $
                              currPlotInfo.channel)       THEN IsRead=1 $
    ELSE IsRead=0
  ENDIF

  IF IsRead THEN BEGIN
    if channel ne 0 then $
      p_plot_type_line, panelVector, channel, OUTPUT=OUTPUT, $
      _EXTRA=extra_PlotInfo $
    else $
      p_plot_type_spec, panelVector, channel, OUTPUT=OUTPUT, $
      _EXTRA=extra_PlotInfo
  ENDIF

; Note: The structure extra_PlotInfo passed to the plot routine contains the
;       following tags:
;                  overPlot               - overplot switch
;                  plots_AtThisPosition   - no. of overplots here
;                  subtable               - index of papco colortable
;                  PSYM                   - linestyle keyword for plot
;                  COLOR                  - linecolor keyword for plot
;
;       They are set by papco and your plot routine can use them if you
;       want to support that functionality.

END
```

B.1.3 plot_type_slice.pro

This file contains all the slice templates for `plot_type`.

```
;***************************************************************************
;* file plot_type_slice.pro
;*
;* this file contains the routines needed to enable the extra mouse features
;* provided for in papco.
;*
;* Currently possible:
;*
;*    mouse zooms                         no routine here, in papco main
;*    mouse vertical slice                routine here
;*    mouse views slice (polar)           specialty slice - no routine here
;*    mouse views ephemeris               under development (POLAR only)
;*    mouse accesses level zero           routine here
;*    mouse writes paneldata              routine here
;*    mouse horizontal slice              routine here
;*
;* Here the procedure names must again adhere to a strict naming convention so
;* that they can be automatically called when needed.
;*
;* The slice types supported by these routines must be ENABLED by the user, so
;* that papco can call them. This is done by setting the value of
;* slice_type in draw_plot_type.pro (contained in papcoadd_plot_type.pro).
;* slice_type is defined in the following way:
;*
;*   slice_type:   Selects which slices are supported and for which slice
;*                 routines have been written.
;*                 This is an integer logical with 16 switches. Currently
;*                 there are 5 papco slices. You may extend this by
;*                 further USER slices up to a total of 16 slices.
;*                     0 = disable all slices
;*                     1 = papco Zoom                 (2^0)
;*                     2 = papco View Slice           (2^1)
;*                     4 = papco Access level Zero    (2^2)
;*                     8 = papco Write Paneldata      (2^3)
;*                    16 = papco Hair Cursor          (2^4)
;*                    32 = papco horizontal Slice     (2^5)
;*                    64 = USER slice A or not used   (2^6)
;*                   128 = USER slice B or not used   (2^7)
;*                    "     "     "   "  "  "    "       "
;*                 32768 = USER slice Z or not used  (2^15)
;*
;*   You can support more than one type of slice by arithmetically adding the
;*   options, eg slice_type=1+4  support Zoom and Level Zero
;*
;* The "papco Zoom" function uses the mouse to define an area of a panel.
;* Dragging the mouse with the left button pressed marks out a shaded rectangle
;* in the panel. The time range and y-range of the rectangle are read out in a
;* window and the user has the option of adopting these new ranges. The papco
;* main common block for time (common mjdt, mjdt_start, mjdt_end) and y-range
;* (common yscale, yscl) are updated accordingly. Provided within PAPCO.
;*
;* The "papco Views Slice" function uses the time returned by papco and the
;* get_data_call specified for the plot_type to read the data and to draw a
;* slice at the clicked position. This is commonly used in spectra color plots
;* to draw a spectrum at that time, or a pitch angle distribution etc, or to
;* call up any other plot at that time as specified by the user.
;* -> user defined interface routine needed, template included here
;*
;* The "papco Accesses level Zero" uses the time passed and calls up another
;* widget application to read and work with level zero data. This application
;* must be entirely independent of papco and is only called from papco via
;* a mouse click in the panel required for the time required.
;* -> user defined interface routine needed, template included here
;*
;* The "papco Write Paneldata" function allows the user to read off data from
;* panels using the mouse and to write that data to a file. This is a papco
;* supplied utility slice which by default only writes the time and y-value of
;* the mouse-click to file. The user can however interface his own routine to
;* add auxiliary data to the string written to file.
;* The output file is written in ASCII and has the same format as the .event
;* files used by the event times utility module. The files can thus be used
;* in the events time module to mark events on panels.
;* -> user defined interface routine possible, template included here
;*
```

```
;* The "papco Hair Cursor" function draws a large hair cursor and outputs
;* x,y,and z corrdinates onto the plot.  Provided within PAPCO.
;*
;* If your new data contains several sub-data types, one slice and/or levelzero
;* routine has to be provided for each type, since in general they would do
;* different things. See
;* $papco_modules\crres_meb\papco_interface\crres_slices.pro
;* for such an example.
;*
;*      PRO DrawSlice_plot_type, panelNr, seconds, yValue
;*        -> Draw a slice for plot_type
;*
;*      PRO levelzero_plot_type, seconds, yValue
;*        -> This procedure calls the widget for levelzero data access
;*           for this plottype.
;*
;*      PRO WriteSlice_plot_type, panelNr, seconds, yValue
;*        -> This procedure calls the widget for writing panelinfo data to file
;*
;* MODIFICATION HISTORY:
;*      December 1995, written by R.Friedel at Goddard
;*      modified may 1997, Reiner Friedel
;******************************************************************************

;******************************************************************************
;* PROCEDURE:
;*      PRO DrawSlice_plot_type, panelNr, seconds, yValue
;*
;* DESCRIPTION:
;*      Draw a slice for plot_type
;*      calling a separate window for the plot.
;*      slice_type in draw_plot_type must be set to +2 to enable this feature.
;*
;* INPUTS:
;*      panelNr     an integer, the number of the panel for which the slice
;*                  is to be drawn
;*      seconds     time of mouse click, a float
;*      yvalue      y-value of mouse click, a float
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      called by papco_DrawSLice (in $papco_PATH/papco_cursor.pro)
;*
;* MODIFICATION HISTORY:
;*      December 1995, written by R.Friedel at Goddard
;******************************************************************************
PRO DrawSlice_plot_type, panelNr, seconds, yValue

   COMMON PAPCO_DRAWSLICE, sliceWidgetData
   COMMON PLOT_COMPOSER, widgetData

; add in here any other common blocks you would need to make your plot data
; available. The common block included here is a dummy for illustration.
; papco uses the the getdatacall you set in papco_add_plot_type to read the
; data you need for the slice. As long as you stay in one panel with the
; mouse, the data is not read again. When you switch panels, the new data is
; read.

   common plot_type_data, input_header, input_data
   common mjdt, mjdt_start, mjdt_end    ;common time limit in mjdt

   time=input_data.time           ; dummy, remove once your own is here

   WIDGET_CONTROL,/HOURGLASS
   thisPlot=widgetData.plotsDrawn(panelNr)

; use the same typevector, switch etc as is used in the call to your plot
; routine. thisPlot contains the full papco structure of plot information for
; the panel in which the mouse was clicked in.

   thisPlot=widgetData.plotsDrawn(panelNr)
   type=thisPlot.typeVector

; set the panel window for the slice to full window

   !p.position=[0.1,0.1,0.9,0.9]
```

```
; select the index for the time chosen out of the time array

   DiffTimes=ABS(time - seconds)        ; this sets timeIndex to the index with
   result=MIN(DiffTimes, timeIndex)     ; t90(index) is closest to 'seconds'

; these two info strings are user-definable output strings to label the slice
; window. To calculate the time use the start time in mjdt and the seconds
; passed in. If your plot uses the papco philosophy then the time axis is in
; running seconds since the start of the start day.
;
; However, you of course can use your own time format, then the information
; in seconds is whatever your time format is. papco provides a function
; to return a verbose time for ANY time format.
; papco always knows the start end time of a plot from the time common
; block, and the start/end value of the x_axis shouldalways be stored in
; papco by the user's plot routine:;
;
;   common coordinateSystems, plotted_x, plotted_y
;
;   plotted_x = !x
;   plotted_y = !y     -   after the user has set up his axis.
;
; If this is done properly then the call
;
;   time_str=papco_cursor_time(seconds,panelNr)
;
; should return the correct time.

   time_str=papco_cursor_time(seconds,panelNr)
   time_mjdt=papco_cursor_time(seconds,panelNr,/mjdt)

   info1='You got to the default slice window'
   info2=string(time_str, yValue, $
        format="('time: ',A20,'   Y-value: ',f9.2)")

; Replace these labels and include your plot!

; user defined routine for plotting the slice included here. An example is in
; $papco_MODULES/crres_MEB/papco_interface/crres_slices.pro
; You typically need to include the commons from the read routine, a way
; of finding the index of the data you want to plot using the time "seconds"
; plus a simple plot statement to plot the data.
; Two strings, info1 and info2, can be used to annotate the slice window
; by providing additional information, ie ephemeris data, etc.

; these calls put your info strings in the provided labels of the slice widget

   WIDGET_CONTROL, sliceWidgetData.lbl_slice1, SET_VALUE=info1
   WIDGET_CONTROL, sliceWidgetData.lbl_slice2, SET_VALUE=info2

END

;******************************************************************************
;* PROCEDURE:
;*      PRO levelzero_plot_type, seconds, yvalue
;*
;* DESCRIPTION:
;*      This procedure calls the widget for level zero data access
;*      for this plottype.
;*      slice_type in draw_plot_type must be set to +4 to enable this feature.
;*
;* INPUTS:
;*      seconds           T90 time at which mouse is clicked
;*      yvalue            y-value at which mouse is clicked
;*
;* OUTPUTS:
;*      none
;*
;* KEYWORDS:
;*      none
;*
;* CALLING SEQUENCE:
;*      levelzero_plot_type, seconds, yvalue
;*
;* MODIFICATION HISTORY:
;*     written November 1995, Reiner Friedel at Goddard
;******************************************************************************
pro levelzero_plot_type, seconds, yvalue
```

```
; The actual procedure is very dependent on what the level zero widget
; application needs. An example is in $papco_MODULES/addpapco_swe_plots.pro.
; Provided as parameters to be passed to the new application, are the values
; of the mouse pointer when clicked.
;
; The main widget remains active and can be switched to (e.g. to get a new
; time from the graph) by using a "Parent" button in the new application
; which then executes the line
;            current_appl=xregistered('papco_main')
; to switch back to the papco main calling widget.

; call the levelzero widget.
;   plot_type_levelzero, seconds, yvalue

; the following line is printed on your xterm as a check to see if papco
; correctly called this routine. If there is a problem make sure your naming
; convention is O.K. : This routine MUST be called levelzero_plot_type,
; where plot_type is the name of the panel you have set in papcoadd_plot_type.

    print, 'got to the levelzero interface  ',seconds, yvalue

END


;******************************************************************************
;* PROCEDURE:
;*      PRO WriteSlice_plot_type, panelNr, seconds, yValue
;*
;* DESCRIPTION:
;*      Draw a Write Paneldata slice for plot_type
;*      calling a seperate window for the widget
;*      slice_type in draw_ceppad must be set to +32 to enable this feature.
;*
;* INPUTS:
;*      panelNr     an integer, the number of the panel for which the slice
;*                  is to be drawn
;*      seconds     time of mouse click, a float
;*      yvalue      y-value of mouse click, a float
;*
;* KEYWORDS:
;*      READDATA    if set the data needs to be re-read from file
;*
;* CALLING SEQUENCE:
;*      called by papco_DrawSLice (in $papco_PATH/papco_cursor.pro)
;*
;* MODIFICATION HISTORY:
;*      may 1997, written by R.Friedel
;******************************************************************************
PRO WriteSlice_plot_type, panelNr, seconds, yValue,READDATA=READDATA

   COMMON PAPCO_WRITE_PANELDATA, sliceWidgetData
   COMMON PLOT_COMPOSER, widgetData

; THIS TEMPLATE ACTUALLY DOES NOTHING, ONLY THE DEFAULT WRITE SLICE
; FUNCTIONALITY WILL WORK (WRITE TIME AND Y-VALUE OF MOUSECLICK TO FILE). THE
; EXAMPLE HERE IS TO SHOW HOW ONE COULD ADD ADDITIONAL INFORMATION.

; add in here any other common blocks you would need to make your additional
; data available (ie ephemeris).

; papco uses the the getdatacall you set in papco_add_plot_type to read the
; plotdata For your panel. As long as you stay in one panel with the
; mouse, the data does not need to be read again (READDATA=0). When you switch
; panels, new data needs to be read (READDATA=0). It's up to the user to read
; in the data he needs here.

;   common plot_type_data, input_header, input_data
;   common plot_type_ephem, plot_type_ephem
;   common mjdt, mjdt_start, mjdt_end    ;common time limits in mjdt

   WIDGET_CONTROL,/HOURGLASS
   thisPlot=widgetData.plotsDrawn(panelNr)

; use the same typevector, switch etc as is used in the call to your plot
; routine. thisPlot contains the full papco structure of plot information for
; the panel in which the mouse was clicked in.

   thisPlot=widgetData.plotsDrawn(panelNr)
   type=thisPlot.typeVector
```

```
   getdatacall=thisPlot.get_data_call

;read the data if needed

;   if READDATA then begin
;      callgood=execute(getdatacall)
;      r_plot_type_ephem
;   endif

; select the index for the time chosen out of the time array

;   DiffTimes=ABSplot_type(.time - seconds)
                                       ; this sets timeIndex to the index
;   result=MIN(DiffTimes, tidx)        ; closest to 'seconds'

; these strings are user-definable output strings to label the slice window.
; The "write Panelinfo" window papco supplies has 6 lines of user-definable
; text.

   info1='User label 1'
   info2='User data  1'
   info3='User label 2'
   info4='User data 2'
   info5='User label 3'
   info6='User data 3'

; Time and y-value of mouse click are automatically put into labels.
; The calls below put your info strings in the provided user labels (max 6).
;
; NB NB NB:
; However, only info2, info4, info6 are written to file in the form
; info2+info4+info6.
;
; The idea is that info1 will have the labels for the data in info2, info3 the
; labels for the data in info4, and info5 the labels for the data in info6.
; Time and y-value of mouse click are written first, then info2+info4+info6
; followed by a user defined comment from the Write Paneldata window.

   WIDGET_CONTROL, sliceWidgetData.lbl_slice1, SET_VALUE=info1
   WIDGET_CONTROL, sliceWidgetData.lbl_slice2, SET_VALUE=info2
   WIDGET_CONTROL, sliceWidgetData.lbl_slice3, SET_VALUE=info3
   WIDGET_CONTROL, sliceWidgetData.lbl_slice4, SET_VALUE=info4
   WIDGET_CONTROL, sliceWidgetData.lbl_slice5, SET_VALUE=info5
   WIDGET_CONTROL, sliceWidgetData.lbl_slice6, SET_VALUE=info6

END
```

newpage

B.1.4 plot_type_panel.help

This text file is the help file for the panel editor widget for `plot_type`.

```
******************************************************************************
  plot_type
******************************************************************************

   Use this window to add new plot_type panels to the Panel Composer or to
   edit existing plot_type panels.

   %%% ADD DESCRIPTION OF DATA HERE %%%


DATA AVAILABILITY
------------------------------------------------------------------------------

   %%% ADD HOW MUCH DATA IS AVAILABLE AT THIS SITE (TIME COVERAGE)  %%%
   %%% ADD A SOURCE WHERE THIS DATA CAN BE OBTAINED FOR OTHER TIMES %%%


FORM OF PLOTS
------------------------------------------------------------------------------

   %%% ADD A DESCRIPTION OF THE TYPE OF PLOTS AVAILABLE IN THIS MODULE %%%
```

```
PAPCO SUPPORT
-------------------------------------------------------------------------------

   %%% ADD A DESCRIPTION OF HOW MUCH PAPCO FUNCTIONALITY YOUR PLOTS %%%
   %%% SUPPORT. IN PARTICULAR:                                      %%%
   %%%                  Y-SCALING SUPPORT                           %%%
   %%%                  SLICE SUPPORT                               %%%
   %%%                  ZOOMING SUPPORT                             %%%
   %%%                  NO DATA FOUND SUPPORT                       %%%
   %%%                  PLOT BY ORBIT AVAILABLE?                    %%%
   %%%                  OVERPLOTPOSSIBLE?                           %%%
   %%%                  PLOT OVER FILE BOUNDARIES POSSIBLE?         %%%


===============================================================================
DESCRIPTION OF PANEL EDITOR FUNCTIONS
===============================================================================

HEIGHT OF THE PLOT
-------------------------------------------------------------------------------
   Use this slider to modify the plot's height. Effectively, this slider is the
   third element of the panel-vector which is displayed in the plot list-box in
   PAPCO's main window. You may as well change the plot's height by using the
   buttons to the right of the plot-description list-box in PAPCO's
   main-window.

-------------------------------------------------------------------------------

   %%% ADD HERE DESCRIPTIONS USER DEFINED WIDGETS IN PANEL EDITOR %%%

-------------------------------------------------------------------------------


SYMBOL / COLOR
-------------------------------------------------------------------------------
   Choose a line style and color for your line plots

   THIS FUNCTION NEEDS TO BE SUPPORTED BY THE PLOT ROUTINE TO WORK.
   IT IS ONLY MEANINGFUL FOR LINE PLOTS.

   If SYMBOL is clicked use the slider below to set your symbol type for each
   data point. Default (0) is no symbols (connect data with line). The
   graphics window to the right shows an example of the symbol chosen.
   Positive SYMBOL numbers are for symbols not connected with lines.
   Negative SYMBOL numbers are for symbols connected with lines.

   If COLOR is clicked a color for your lines and symbols can be chosen.


COLORBAR
-------------------------------------------------------------------------------
   Choose a PAPCO color subtable.

   THIS FUNCTION NEEDS TO BE SUPPORTED BY THE PLOT ROUTINE TO WORK.
   IT IS ONLY MEANINGFUL FOR COLOR PLOTS.

   PAPCO provides the user with four internal color tables. These can be used
   in several plots at the same time as PAPCO subdivides the IDL color-table
   into four. The buttons choose the color sub-table to be used, the names
   given are those for the color tables loaded at startup. The first one
   is the default PAPCO color table (GSFC rainbow) and cannot be changed.
   The other three can be user defined.

   The colorbars indicate the current status of the loaded color sub-tables.


YSCALING
-------------------------------------------------------------------------------
   Choose automatic or manual Y-scaling.

   THIS FUNCTION NEEDS TO BE SUPPORTED BY THE PLOT ROUTINE TO WORK.

   For Use automatic y-scaling, the common block vector yscl has it's first
   element set to zero, which tells the plot routine to use it's own
   yscaling - which is either hard-wired or employs some auto-scaling
   routines (autoscale, ylim).

   For Use manual y-scaling the entry fields below are activated: enter your
```

```
   own y-range values. These are stored in the 2nd and 3rd elements of
   the common block vector yscl.

   Once a plot has been done, the automatic values chosen are displayed in the
   entry field to guide the user if he wants to change them.

   The lin/log buttons set the 4th element in the common block vector yscl
   and can be used directly to set the ylog keyword to the IDL plot routine.


ZSCALING
-------------------------------------------------------------------------------
   Choose automatic or manual Z-scaling.

   THIS FUNCTION NEEDS TO BE SUPPORTED BY THE PLOT ROUTINE TO WORK.
   IT IS ONLY MEANINGFUL FOR COLOR OR 3-D PLOTS.

   For Use automatic z-scaling, the common block vector zscl has it's first
   element set to zero, which tells the plot routine to use it's own
   yscaling - which is either hard-wired or employs some auto-scaling
   routines (autoscale, ylim).

   For Use manual z-scaling the entry fields below are activated: enter your
   own y-range values. These are stored in the 2nd and 3rd elements of
   the common block vector zscl.

   Once a plot has been done, the automatic values chosen are displayed in the
   entry field to guide the user if he wants to change them.

   The lin/log buttons set the 4th element in the common block vector zscl
   and can be used directly to set the zlog keyword to the IDL plot routine.

HELP
-------------------------------------------------------------------------------

   Displays this help widget.

THE EXIT BUTTONS
-------------------------------------------------------------------------------

   CANCEL           closes the window and does nothing.

   ADD and CONTINUE closes the current window, adds the plot to the composer,
                    and re-enters the panel editor where it left off and
                    allows you to add a further panel with possibly another
                    choice of parameters.
   ADD and CLOSE    adds the plot to the composer and closes the window.


   ADD and CONTINUE and ADD and CLOSE are replaced by
   UPDATE plot_type when the editor is called to edit an existing panel.
```

B.1.5 init_module.pro

This file is the initialization batch file for the module `plot_type`.

```
print, '  --> for plot type plots -------------------------------------------'
print, '     --> read and plot routines'
.run r_plot_type
.run p_plot_type_line
.run p_plot_type_spec
print, '     --> papco support routines'
.run plot_type_panel
.run papcoadd_plot_type
.run plot_type_slice
```

B.2 Plot & Read templates

These routines are user specific and do the actual data reading and data plotting. The template provided here serves as a working guide, but there are NO RESTRICTIONS on how you decide to read or plot your data yourself.

PAPCO provides some plot routines for line plots and color spectrograms. Most modules written so far use these routines, but again, there is NO RESTRICTION on the way you want to do things. However, to make panels look good together, some of the conventions used by these plot routines make sense!

B.2.1 r_plot_type.pro

This file is the read routine for `plot_type`.

```
;***************************************************************************
;*  NAME:        r_plot_type
;*
;*  DESCRIPTION: Routine to read plot_type files. Adheres to
;*               papco conventions: Error common block
;*                                  Uses MJDT time
;*                                  papco read philosophy
;*
;*  INPUT:       none (uses common block time from papco)
;*
;*  OUTPUT:      none (places data into common blocks)
;*
;*  KEYWORDS:    VERBOSE  -  enables print comments, save/restore comments
;*
;*  DATE:        June 1996
;*
;*  AUTOR:       R. Friedel
;*
;*  CHANGES:     February 1997. Used MJDT time now, and is a fully
;*               running example module on it's own
;*               October 1997. Rewritten to be more general and fully
;*               adhering to papco philosophy. Added remore Get feature, and
;*               the example data file is no longer part of the
;*               module - it needs to be gotten remotely first!
;***************************************************************************
pro r_plot_type, VERBOSE=VERBOSE

  common plot_type_data, input_header, input_data
  common mjdt, mjdt_start, mjdt_end
  common get_error, get_err_no, get_err_msg


; common get_error is used by papco to check if the data read was successful.
; it is up to the user to set this correctly and to make sure that the read-
; routine does not crash on an I/O error - otherwise papco will crash at that
; point, too. papco draws an empty panel when get_err_no in not equal to zero
; and prints get_err_msg in that panel for user information. papco will try
; to invoce the plot routine for this data if get_err_no = 0 , otherwise an
; emplty panel with the error message get_err_msg is plotted.

  get_err_no=0
  get_err_msg='plot_type read succesfully'

; check for env. variable pointing to data directory
  data_dir='PLOT_TYPE_DATA'
  path=papco_getenv(data_dir)
  if strlen(path) eq 0 then begin
    get_err_no=1
    get_err_msg='Environmental variable '+data_dir+' not set'
    print,get_err_msg
    return
  endif

; check for existence of data directory
  cc=papco_finddir(path)
  if cc eq 0 then begin
    get_err_no=2
    get_err_msg='Data directory '+path+' not found'
    print,get_err_msg
```

```
    return
  endif

; check VERBOSE keyword and set
   if keyword_set(VERBOSE) then verbose=VERBOSE else verbose=0

; clear data arrays for reuse
   input_header=0  &  input_data=0

; loop through from start to end time concatonating data files into one
; array. Here we assume that the data is stored in files per day, and
; that the time info in each file is seconds of day. papco standard is
; to hand over data to the plot routines with the time running in
; seconds from the number of seconds of the startday.

   time=mjdt_start  &  no_files_read=0  &  day_count=0.0

   while mjdt_end.mjd ge time.mjd do begin

     ; assemble the data file from the time range requested
     convert_t90_to_date, mjdt_start, year, day_oy, hour, minute, second,$
                         month, cmonth, day_om, /mjdt
     datestr=string(year-1900,month,day_om,format='(i2.2,i2.2,i2.2)')
     fln='plot_type_'+datestr

     ; read one day of data
     r_plot_type_onefile, path+fln, /VERBOSE

     ; check for "OPENR: Error opening file" in error string - and then get
     ; the file remotely!
     ; comment this section out if you do not want this feature.
     res=strpos(!err_string,'OPENR: Error opening file')
     if res ne -1 then begin
       c_url=papco_getenv('PLOT_TYPE_FTP_ARCHIVE')
       c_file=fln+'.idl'
       c_download_dir=path
       papco_get_remote_data, c_url, c_file, c_download_dir
       ;check if file "arrived" and then read it
       if get_err_no eq 0 then $
         r_plot_type_onefile, path+fln, /VERBOSE $
       else begin
         get_err_msg='% r_plot_type: No data locally, remote get failed!C  ' $
           + get_err_msg
         return
       endelse
     endif

     ;if a file was succesfully read, concatonate!
     if get_err_no eq 0 then begin
       if no_files_read eq 0 then begin   ;start the array buffer
         input_data.time=input_data.time+86400.0*day_count
         input_data_buf=temporary(input_data)
       endif else begin                   ;otherwise concatonate arrays
         input_data.time=input_data.time+86400*day_count
         input_data_buf=[temporary(input_data_buf),temporary(input_data)]
       endelse
       input_data=0                     ;clear array for reuse
       no_files_read=no_files_read+1
     endif
     day_count=day_count+1
     time.mjd=time.mjd+1                 ; next day
   endwhile

; check if any files were read
   if (no_files_read ne 0) and (get_err_no ne 1) then begin
     get_err_no=0
     get_err_msg='% r_plot_type: '+string(no_files_read,format="(i2)")+ $
       ' files read succesfully'
     ;clear arrays and load buffer into output array
     input_data=0  &  input_data=input_data_buf  &  input_data_buf=0
   endif else begin
     get_err_no=1
     get_err_msg='% r_plot_type: no files found for timerange'
     return
   endelse

; restrict data to actual time range requested
   ut1=mjdt_start.t
   ut2=long(mjdt_end.mjd-mjdt_start.mjd)*86400l+mjdt_end.t
```

```
   index=where((input_data.Time ge ut1) and (input_data.Time le ut2),count)
   if count ne 0 then begin
      input_data=input_data(index)
   endif

end

;******************************************************************************
;*  NAME:        r_plot_type_onefile,fln
;*
;*  DESCRIPTION: reads one file of type plot_file. Assumes data has been
;*               stored in IDL saveset using structures input_header and
;*               input_data
;*
;*  INPUT:       fln  - string of filename
;*
;*  OUTPUT:      none (places data into common blocks)
;*
;*  DATE:        October 1997
;*
;*  AUTOR:       R. Friedel
;******************************************************************************
pro r_plot_type_onefile, fln, VERBOSE=VERBOSE

   common plot_type_data, input_header, input_data
   common get_error, get_err_no, get_err_msg

   get_err_no=0
   get_err_msg='plot_type onefile read succesfully'

; clear data arrays for reuse
   input_header=0  &  input_data=0

   on_ioerror, read_original

   restore,fln+'.idl',/verbose
   dealloc_lun,/quiet    ;frees logical units used by save/restore
   print,'% r_plot_type_oneday: idl saveset read succesfully'
   return

   read_original:
   ; if the .idl save file could not be found, then try to read the original
   r_plot_type_onefile_original, fln, VERBOSE=VERBOSE

end

;******************************************************************************
;*  NAME:        r_plot_type_onefile_original,fln
;*
;*  DESCRIPTION: reads one file of type plot_file_original. Under the
;*               papco read philosophy, files should be read the first
;*               time in whatever format they might have - ascii,
;*               fortran binary, pascal binary, etc. The files might also
;*               be in some compressed format, so they need to be uncompressed
;*               first.
;*
;*               You might also have to change time formats.
;*               A lot of files are stored per day, and have seconds of
;*               that day as the time. This is what is assumed here and
;*               is papco standard. You need to convert the time
;*               to papco standard if you intend to use the plotroutines
;*               supplied with papco (use the routines in
;*               $papco_BASE/papco_Lib/time_convert.pro).
;*               If you use your own plot routines this step is not needed.
;*
;*               The first read might be slow, esp if the format is ascii.
;*               Use the IDL "save" command to make a binary copy of the
;*               file which can then be read quickly using "restore". The
;*               next time only the IDL saveset is read, which is much
;*               faster.
;*
;*               In order to be able to concatonate the arrays easily,
;*               it makes sense to have the data in an array of time-ordered
;*               structures, which makes adding together days of data
;*               trivial using the IDL concatonation operator "[]".
;*
;*  INPUT:       fln  - string of filename
;*
;*  OUTPUT:      none (places data into common blocks)
```

```
;*
;*  DATE:        October 1997
;*
;*  AUTOR:       R. Friedel
;******************************************************************************
pro r_plot_type_onefile_original, fln, VERBOSE=VERBOSE

   common plot_type_data, input_header, input_data
   common get_error, get_err_no, get_err_msg

   get_err_no=0
   get_err_msg='plot_type_onefile_orignal read succesfully'

; clear data arrays for reuse
   input_header=0  &  input_data=0

;------------------------------------------------------------------------------
;
; read the original data file - THIS NEEDS TO BE REWRITTEN BU USER!!!!

; define the header and data structures. Edit to suit your data!
   max_no_rec=1000              ;the maximum number time records in your data
   ext='.txt'                   ;file extention of your dat files
   input_header={plot_type_header,tag1:0,tag2:0}
   dummy={plot_type_data, time:0.0d, val1:0.0, val2:0.0}
   input_data=replicate(dummy, max_no_rec)

   on_ioerror, out
   openr,unit,fln+ext,/get_lun
   while not eof(unit) do begin
     ;read in data and store in input_data
     readu,unit,dummy
     input_data(i)=dummy
   endwhile

; end of user written section
;------------------------------------------------------------------------------


; save read data with correct time. Rename data to *.idl
  save,filename=fln+'.idl',input_header,input_data,/verbose
  dealloc_lun,/quiet              ;frees logical units used by save/restore
  return

  out:
  ; if there was an error, report it. Use the IDL error variables. Error
  ; 171 is for "file not fond" which is used to invoke the remote get.
  get_err_no=!ERROR
  get_err_msg='r_plot_type_onefile:!C '+!ERR_STRING

end
```

B.2.2 p_plot_type_line.pro

This file is the plot routine for `plot_type` line plots.

```
;***************************************************************************
;* file p_plot_type_line.pro
;*
;* this file is a template for a plot routine which can be used by papco.
;* it includes all the periphery needed, the user simply has to add in
;* the actual plot statement for his/her kind of data.
;*
;* This is a working example for a pre-defined example data array.
;***************************************************************************
;***************************************************************************
;* PROCEDURE:
;*      p_plot_type_line, panel, type, $
;*           OVERPLOT=OVERPLOT,$
;*           PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,$
;*           SUBTABLE=SUBTABLE, $
;*           _EXTRA=extra_par
;*
;* DESCRIPTION:
;* This procedure plots data of type data_type for inclusion in papco.
;*      Data must have been read and placed into common blocks before this
;*      routine is called. Behavior is controlled by the vector panel
;* which controls positioning of the plot (see papco_conventions.text
;*      in $papco). In general, the following plot conventions are used:
;*              - if the panel is at the bottom plot time axis plus any
;*                additional axis (ie ephemeris) needed.
;*              - if panel is at the top also plot a title
;*              - a descriptor for panels is plotted to the right of each
;*                panel, rotated by 90 deg if there is a papco_colorbar.
;*              - panels all have common time
;*              - y-scaling (or z-scaling for color plots) is either
;*              - automatic or manualy set using common yscale or zscale
;*
;* INPUTS:
;*      panel   three element integer vector. Controls positioning of the plot
;*              (see papco_conventions.text in $papco)
;*      type    optional parameter for options. Normally is a four element
;*              integer vector. This is normally equivalent to the field
;*              typeVector of the papco structure papco_PLOTINFO which is set
;*              by the panel editor. Additional fields that can be used if
;*              needed are papco_PLOTINFO.switch (three element vector) and
;*              papco_PLOTINFO.channel (integer). If your plot routine needs
;*              more input fields than these you have to extend the definition
;*              of the structure papco_PLOTINFO
;*              (procedure papco_getPlotInfoStruct in $papco/papco.pro). Try to
;*              avoid this!
;*
;* OUTPUTS:
;*      none
;*
;* KEYWORDS:
;*      OUTPUT  if set call the papco routine that outputs plot data
;*              to a file.
;*    OVERPLOT  if you support this you can alter the appearance of your
;*              plot if it is an overplot to an existing panel. Standard is
;*              to plot data, no axis, using the y-scaling of the first plot
;*              and to shift the labeling on the left down one line. Useful
;*              to plot model data together with real data, ie mag. field.
;*              see p_crres_mag for an example.
;*    PLOTS_ATTHISPOSITION  used to control the left side labeling for more
;*              than one overplot at the same position.
;*      _EXTRA  the plot structure containing some keyword for calls to
;*              plot. Used to customize color and line style.
;*
;* CALLING SEQUENCE:
;*       p_plot_type_line, panel, type, $
;*           OVERPLOT=OVERPLOT, PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,$
;*           _EXTRA=extra_par
;*
;* MODIFICATION HISTORY:
;*     written December 1995, Reiner Friedel, at Goddard
;*     added OUTPUT support, July 1998, , Reiner Friedel
;***************************************************************************
pro p_plot_type_line, panel, channel, OUTPUT=OUTPUT, $
                      OVERPLOT=OVERPLOT, $
```

```
                     PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION, $
                     SUBTABLE=SUBTABLE, $
                     _EXTRA=extra_par

; NOTE: I use a little trick in passing in the keywords to to this
; routine. The calling routine's call is simply:
;        p_plot_type_line, panelVector, currPlotInfo.typeVector(0), $
;                          _EXTRA=extra_PlotInfo
; extra_PlotInfo contains the tags OVERPLOT, PLOTS_ATTHISPOSITION,
; SUBTABLE and then some tags which are meant to be passed to the plot
; call. By putting in the explicit keywords in the procedure definition they
; get split out of the structure extra_PlotInfo that was passed in, and only
; those tags which are meant for the plot routine remain in extra_par.

; Comments here should include an explanation of the plot options used
; in type, switch. Replace type in the call with whatever variables
; you need, or add more

; The following common blocks are needed:

   common mjdt, mjdt_start, mjdt_end    ;common time limit in mjdt
   common yscale, yscl ;man/auto yscaling
   common shift_label, down             ;common for x-axis label shifting
   common coordinateSystems, plotted_x, plotted_y
                                        ;info on coords used by mouse

; One of the common blocks should contain the plot data as returned by the
; data read procedure r_plot_type.

   common plot_type_data, input_header, input_data

   panelset,panel ;sets the panel position viewport

; option to automatically calculate ytick numbers in order to avoid to
; many ticks if panels get too narrow. Maximum is set to 10. You may
; leave this out and allow plot to use the built-in defaults.

   ytickno=50*(!p.position(3)-!p.position(1))
   if ytickno gt 10 then ytickno=10
   ytickno=fix(ytickno) > 1

; if yaxis is logarithmic, allow automatic yticks.
   if yscl(panel(0),3) eq 1 then ytickno=0

; since there is a common time axis plots look better if they all use the same
; number of xticks. In papco, this number is 6 by default. This is set by papco
; using the IDL graphics system variable !X.TICKS.
; You can override this for your own plot by setting the variable
; below. However, for plotting panels together, it makes sense to stick to the
; default. If you do set this variable, it needs to be included in the
; extra_plotPar_common definition below.

;   xtickno=6

; NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!!
; Avoid setting the global IDL graphics system variables in your plot routine,
; as they may affect other modules' plot routines. The IDL graphics system
; variables are set by papco to default values that guarantee some uniformity
; of plot appearance and should not be tampered with - unless you don't
; care if your plot "looks good" when combined with the other modules.
; NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!!

; the plot-calls normally use the arrays time for the time axis and yarr (ymat
; for color plots) for the y or z axis.
; depending on your plot options, different data may be selected and different
; axis-labeling may be needed. This is normally done using a case statement.
; In this example the data array consist of flux values in ten channels
; at different energies, typical of particle spectrometer data. Here I use the
; channel information passed in the plotinfo structure from the panel editor
; to select the channel to be drawn.

; set the begin end end times. papco convention is to use running seconds
; starting with the no of seconds since the beginning of the start day.
   time=input_data.time
   xut1=mjdt_start.t
   xut2=long(mjdt_end.mjd-mjdt_start.mjd)*86400l+mjdt_end.t

   case channel-1 of
     0:begin
```

```
          yray=input_data(*).ymat(channel-1)
          utitle='channel 1'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      1:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 2'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      2:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 3'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      3:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 4'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      4:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 5'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      5:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 6'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      6:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 7'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      7:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 8'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      8:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 9'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
      9:begin
          yray=input_data(*).ymat(channel-1)
          utitle='channel 10'
          uytitle='flux'
          ylim3,time,yray,yst,yen
        end
    endcase

; restrict data to actual time range requested - this makes for faster zooming

    index=where((time ge xut1) and (time le xut2),count)
    if count ne 0 then begin
       yray=yray(index)
       time=time(index)
    endif

; the variables used above have the following meaning:
;  uytitle : the y-axis label
; utitle  : the right-of plot label defining the data
;
; the procedure YLIM3 is optional and sets the ylimit automatically for the
; data leaving a 10% gap at the top and bottom limits.
; this routine is very simple and does not handle spikes or bad data, you may
; need to supply your own procedure for your own data.
; (in $papco/papco_Lib/ylim.pro)
```

```
; yscl of common block yscale is an array of dimensions (*,4) and contains info
; on the yscaling to be used. The first dimension identifies the panel, and the
; next dimension has three entries:
;         yscl(*,0)  is  0 for "use automatic scaling", 1 for "use manual"
;         yscl(*,1)  is then ymin (or zmin)
;         yscl(*,2)  is then ymax (or zmax)
;         yscl(*,3)  is 0 for "linear" and 1 for "log"  scale
; when scaling is set to automatic, the automatic y-limits are loaded in.

   if (yscl(panel(0),0) eq 1) then begin
      yst=yscl(panel(0),1)
      yen=yscl(panel(0),2)
   endif else begin
      yscl(panel(0),1)=yst
      yscl(panel(0),2)=yen
   endelse

; set up extra plot keywords: yrange, xrange, zrange  are always needed.
; xtickformat is used to set the format of the x-axis ticks, by
; default  no ticks ar plotted.
; For the rest you can use any of the kywords allowed with IDL's cplot
; routine. But beware that some keywords have default values set by
; PAPCO:
;
;    !P.CHARSIZE  = widgetData.default_charsize
;    !P.CHARTHICK = widgetData.default_charthick
;    !P.FONT      = -1
;    !P.NOERASE   = 1
;    !P.TICKLEN   = widgetData.default_TICKLEN
;    !X.TICKS     = widgetData.default_XTICKS
;    !X.STYLE     = widgetData.default_XSTYLE
;    !Y.STYLE     = 1
;    !X.MINOR     = widgetData.default_XMINOR
;
; Overriding these values by setting keywords is of course possible
; and allowed but discouraged, since
; then your plot might no longer have the "common" look used in
; PAPCO. The widgetData structure is PAPCO's main internal data structure.

   extra_plotPar={xrange:[xut1,xut2], yrange:[yst,yen],  $
                ylog:yscl(panel(0),3), $
                yticks:ytickno, ytitle:uytitle, xtickformat:'noticks'}

; check if the passed-in structure extra_par was set. If not, create
; it and set color to default black; which for the color table used by
; papco is 1. If extra_par was set, see if the tag color exists. If it
; does, leave it unchanged, if not, add it and set color to default
; black.

  if n_elements(extra_par) EQ 0 then extra_par={color:1} else begin
    names=tag_names(extra_par)
    res=where(names eq 'COLOR',count)
    if count eq 0 then extra_par=create_struct({color:1}, extra_par)
  endelse

; add keyword structure set here with the one passed in

   extra_plotPar=create_struct(extra_plotPar, extra_par)

; change the xtickformat if the panel is the bottom one and needs to
; plot a time axis

  if panel(0) eq 0 then extra_plotPar.xtickformat='MJDT_DATE_4'

; You might have to add your own xtickformat for time axis
; labeling. If you stick to the papco convention of always having your plot
; time array having consecutive seconds from midnight of the start day, then
; you can use the pre-defined xtickformat functions defined in ticks.pro in
; papco_Lib/axis_label.
; You can at this point also define a custom ytickformat, or leave it out for
; IDL default plot behaviour.

; "down" sets the axis labeling level.

  down=0

; To keep things general, plotting of the data is split up into plotting the
; axis with no data, and then plotting the data with no axis. This is done to
; enable the feature of overplots, which just plot data into an existing axis.
```

```
; plot the axis in color=1, and only plot axis if if you're NOT over-plotting!
; down sets the axis labeling level (adds carriage returns [!C] to labels).

   down=0
   color1_plotpar=extra_plotPar
   color1_plotpar.color=1

   if not keyword_set(overplot) then $
      plot,time, _extra=color1_plotPar, /NODATA

; now do the actual data plot, data only, no axis

   plot,time,yray, _extra=extra_plotPar, xstyle=5, ystyle=5

; store the coordinate information into common block for slices

   plotted_x = !x
   plotted_y = !y

; plot extra x-axis labels (eg. ephemeris) and a x-axis label if required.
; plot in color=1. Example here adds an axis with date information.
; The routine x_side_label puts x-axis label to the right of the axis.
; If you stick to the papco convention of always having your plot
; time array having consecutive seconds from midnight of the start day, then
; you can use the pre-defined xtickformat functions defined in ticks.pro in
; papco_Lib/axis_label.
; Or use any other function you define yourself.
; down sets the axis labeling level

  if (panel(0) eq 0) and not keyword_set(OVERPLOT) then begin
     extra_plotPar.xtickformat='MJDT_DATE_3'
     extra_plotPar.color=1
     down=1
     plot,time,yray, _extra=extra_plotPar, /NODATA
     x_side_label,panel,'time (UT)!CDate'
  endif

; plot info right of the panel. If you want overplots to have their own labels
; you need to add position-control for this label using the keyword
; PLOTS_ATTHISPOSITION

  if not keyword_set(overplot) then begin
    right_side_label,panel,utitle,_extra=extra_Par
  endif else begin
    for i=0, PLOTS_ATTHISPOSITION-1 do utitle='!C'+utitle
    right_side_label,panel,utitle,_extra=extra_Par
  endelse

; check if data needs to be output to file. Use the optional keywords
; to add more information to the data file
;      DESCRIPTION   a string or string array describing the data
;      TIME_LABEL    a string decribing the x-axis format
;      CHANNELS      an (ch,2) array giving the channel start/end
;      Y_LABEL       a string with the y-axis label
;      Z_LABEL       a string with the z-axis label
;      RS_LABEL      a string with the righ-side label of the plot

  if KEYWORD_SET(OUTPUT) then begin
    print,'% p_plot_type_line: Writing plot data out to file'
    description='plot_type template sample data - single energy channel line'
    time_label='Seconds since start of day'
    ;channels=yarr
    y_label=uytitle
    ;z_label=uztit
    rs_label=utitle
    papco_write_data, time, yray, $
                      DESCRIPTION=description, TIME_LABEL=time_label, $
                      Y_LABEL=y_label, $
                      RS_LABEL=RS_LABEL
  endif


end
```

B.2.3 p_plot_type_spec.pro

This file is plot routine for `plot_type` spectra plots.

```
;******************************************************************************
;* file p_plot_type_spec.pro
;*
;* This file is a template for a plot routine which can be used by PAPCO.
;* It includes all the periphery needed. This routine makes use of a
;* color spectrogram routine that is supplied with PAPCO:
;* papco_XX/papco_lib/papco_plot_colorspec.pro. This routine should
;* meet all the requirements for the display of 3-D data (tyhe third
;* dimension being color). This routine also adheres to the PAPCO color scheme.
;* You of course can use your own color plot rotuine if desired!
;*
;* This is a working example for a pre-defined example data array.
;******************************************************************************
;******************************************************************************
;* PROCEDURE:
;*      p_plot_type_spec, panel, type, OUTPUT=OUTPUT, $
;*           OVERPLOT=OVERPLOT,$
;*           PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,$
;*           SUBTABLE=SUBTABLE, $
;*           _EXTRA=extra_par
;*
;* DESCRIPTION:
;* This procedure plots data of type data_type for inclusion in papco.
;*      Data must have been read and placed into common blocks before this
;*      routine is called. In general, the following plot conventions are used:
;*       - if the panel is at the bottom plot time axis plus any
;*               additional axis (ie ephemeris) needed.
;*             - a descriptor for panels is plotted to the right of each
;*               panel, rotated by 90 deg if there is a papco_colorbar
;*             - panels all have common time
;*             - y-scaling (and/or z-scaling for color plots) is either
;*             - automatic or manually set using common yscale or zscale
;*
;* INPUTS:
;* panel three element integer vector. Controls positioning of the plot
;*             (see papco_conventions.text in $papco)
;*      type    optional parameter for options. Normally is a four element
;*             integer vector. This is normally equivalent to the field
;*             typeVector of the papco structure PAPCD_PLOTINFO which is set
;*             by the panel editor. Additional fields that can be used if
;*             needed are PAPCO_PLOTINFO.switch (three element vector) and
;*             PAPCO_PLOTINFO.channel (integer). If your plot routine needs
;*             more input fields than these you have to make use of
;*             an extra common block.
;*
;* OUTPUTS:
;*    none
;*
;* KEYWORDS:
;*      OUTPUT  if set call the papco routine that outputs plot data
;*             to a file.
;*    OVERPLOT  if you support this you can alter the appearance of your
;*             plot if it is an overplot to an exisiting panel. Standard is
;*             to plot data, no axis, using the y-scaling of the first plot
;*             and to shift the labeling on the left down one line. Useful
;*             to plot model data together with real data, ie mag. field
;*             (see p_crres_mag for an example).
;* PLOTS_ATTHISPOSITION  used to control the left side labeling for more
;*             than one overplot at the same position.
;*      _EXTRA  the plot structure containing some keyword for calls to
;*             plot. Used to customize color and line style.
;*
;* CALLING SEQUENCE:
;*       p_plot_type, panel, type, $
;*           OVERPLOT=OVERPLOT, PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,$
;*           _EXTRA=extra_par
;*
;* MODIFICATION HISTORY:
;*     written april 1996, Reiner Friedel
;*     added OUTPUT support, July 1998, , Reiner Friedel
;******************************************************************************
pro p_plot_type_spec, panel, type, OUTPUT=OUTPUT, $
                      OVERPLOT=OVERPLOT, $
                      PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,$
```

```
                    SUBTABLE=SUBTABLE, $
                    _EXTRA=extra_par

; It doesn't make much sense to use a color plot as an overplot, as it will
; hide what's below. Line styles are also not useful. So the keywords
; OVERPLOT=OVERPLOT, PLOTS_ATTHISPOSITION=PLOTS_ATTHISPOSITION,_EXTRA=extra_par
; are not used at all here, but are in the call for consistency...

; NOTE: I use a little trick in passing in the keywords to to this
; routine. The calling routine's call is simply:
;         p_plot_type_line, panelVector, currPlotInfo.typeVector(0), $
;                           _EXTRA=extra_PlotInfo
; extra_PlotInfo contains the tags OVERPLOT, PLOTS_ATTHISPOSITION,
; SUBTABLE and then some tags which are meant to be passed to the plot
; call. By putting in the explicit keywords in the procedure definition they
; get split out of the structure extra_PlotInfo that was passed in, and only
; those tags which are meant for the plot routine remain in extra_par.

; Comments here should include an explanation of the plot options used
; in type and/or switch. Replace type in the call with whatever variables
; you need, or add more.

; The following common blocks are needed for PAPCO:

   common mjdt, mjdt_start, mjdt_end    ;common time limit in mjdt
   common yscale, yscl ;man/auto yscaling
   common zscale, zscl ;man/auto zscaling
   common shift_label, down             ;common for x-axis label shifting
   common coordinateSystems, plotted_x, plotted_y
                                        ;info on coords used by mouse

; One of the common blocks should contain the plot data as returned by the
; data read procedure r_plot_type.

   common plot_type_data, input_header, input_data

   papco_Set_SubTable, SUBTABLE         ;sets the papco color table index
   panelset,panel ;sets the panel position viewport

; option to automatically calculate ytick numbers in order to avoid to
; many ticks if panels get too narrow. Maximum is set to 10. You may
; leave this out and allow plot to use the built-in defaults.

   ytickno=50*(!p.position(3)-!p.position(1))
   if ytickno gt 10 then ytickno=10
   ytickno=fix(ytickno) > 1

; since there is a common time axis plots look better if they all use the same
; number of xticks. In papco, this number is 6 by default. This is set by papco
; using the IDL graphics system variable !X.TICKS.
; You can overide this for your own plot by setting the varbale
; below. However, for plotting panels together, it makes sense to stick to the
; default. If you do set this variable, it needs to be included in the
; extra_plotPar_common definition below.

;   xtickno=6

; NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!!
; Avoid setting the global IDL graphics system variables in your plot routine,
; as they may affect other modules' plot routines. The IDL graphics system
; variables are set by papco to default values that gurantee some uniformity
; of plot appearance and should not be tampered with - unless you don't
; care if your plot "looks good" when combined with the other modules.
; NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!! NB!!!

; for the color plot routine, you need to set up arrays containing the upper
; and lower limits of each channel. There must be as many entries as
; there are channels in the data array!

   n_channels=10
   yarr=fltarr(n_channels,2)

   yarr(*,0)=[37.,54.,69., 85.,113.,147.,193.,254.,335.,447.]
   yarr(*,1)=[54.,69.,85.,113.,147.,193.,254.,335.,447.,602.]

; the plot-calls normally use the arrays time for the time axis and yarr (zmat
; for color plots) for the y or z axis.
; depending on your plot options different data may be selected and different
; axis-labeling may be needed. This is normally done using a case statement.
```

```
; in this example we have a single test array, so no case statement is needed.

; set the begin end end times. PAPCO convention is to use running secconds
; starting with the no of seconds since the beginning of the start
; day.

   time=input_data.time
   xut1=mjdt_start.t
   xut2=long(mjdt_end.mjd-mjdt_start.mjd)*86400l+mjdt_end.t

   zmat=input_data.ymat
   zmat=transpose(zmat)
   uztit='flux'
   uytitle='Energy (keV)'
   utitle='Test Spectra!Cdata'

   yst=yarr(0,0)
   yen=yarr(n_channels-1,1)

; restrict data to actual time range requested - this makes for faster zooming

   index=where((time ge xut1) and (time le xut2),count)
   if count ne 0 then begin
      zmat=zmat(index,*)
      time=time(index)
   endif

; the variables used above have the following meaning:
;       uztit   : the z-axis (papco_colorbar) label
;  uytitle : the y-axis label
; utitle  : the right-of plot label defining the data

; the procedure papco_autorange (in papco_XX/papco_lib/autoscaling is
; optional and sets the zlimits automatically for the data, attempting
; to use the full dynamic range of the color table. you may need to
; supply your own procedure for your own data. Nodata sets the nodata
; flag used for the dataset.

; yscl or zscale is an array of dimensions (*,4) and contains info
; on the scaling to be used. The first dimension identifies the panel, and the
; next dimension has three entries:
;         yscl(*,0)  is  0 for "use automatic scaling", 1 for "use manual"
;         yscl(*,1)  is then ymin (or zmin)
;         yscl(*,2)  is then ymax (or zmax)
;         yscl(*,3)  is 0 for lin, 1 for log axis
; when scaling is set to automatic, the automatic y limits are loaded in.
; for color plots, the z-limits are used for the color (z-axis) scaling

   if (zscl(panel(0),0) eq 1) then begin
     zmin=zscl(panel(0),1)
     zmax=zscl(panel(0),2)
   endif else begin
     papco_autorange,zmat,zmin,zmax, $
       log=zscl(panel(0),3), exclude=0, nodata=1e-20
     zscl(panel(0),1)=zmin
     zscl(panel(0),2)=zmax
   endelse

   if (yscl(panel(0),0) eq 1) then begin
     yst=yscl(panel(0),1)
     yen=yscl(panel(0),2)
   endif else begin
     yscl(panel(0),1)=yst
     yscl(panel(0),2)=yen
   endelse

; set up extra plot keywords: yrange, xrange, zrange  are always needed.
; xtickformat is used to set the format of the x-axis ticks, by
; default  no ticks ar plotted.
; For the rest you can use any of the kywords allowed with IDL's cplot
; routine. But beware that some keywords have default values set by
; PAPCO:
;
;    !P.CHARSIZE  = widgetData.default_charsize
;    !P.CHARTHICK = widgetData.default_charthick
;    !P.FONT      = -1
;    !P.NOERASE   = 1
;    !P.TICKLEN   = widgetData.default_TICKLEN
;    !X.TICKS     = widgetData.default_XTICKS
```

```
;    !X.STYLE     = widgetData.default_XSTYLE
;    !Y.STYLE     = 1
;    !X.MINOR     = widgetData.default_XMINOR
;
; Overriding these values by setting keywords is of course possible
; and allowed but discouraged, since
; then your plot might no longer have the "common" look used in
; PAPCO. The widgetData structure is PAPCO's main internal data structure.

   extra_plotPar={xrange:[xut1,xut2], yrange:[yst,yen], zrange:[zmin,zmax], $
                ylog:yscl(panel(0),3), zlog:zscl(panel(0),3), $
                ytitle:uytitle, xtickformat:'noticks'}

; check if the passed-in structure extra_par was set. If not, create
; it and set color to default black; which for the color table used by
; papco is 1. If extra_par was set, see if the tag color exists. If it
; does, leave it unchanged, if not, add it and set color to default
; black.

  if n_elements(extra_par) EQ 0 then extra_par={color:1} else begin
    names=tag_names(extra_par)
    res=where(names eq 'COLOR',count)
    if count eq 0 then extra_par=create_struct({color:1}, extra_par)
  endelse

; add keyword structure set locally and the one passed in

  extra_plotPar=create_struct(extra_plotPar, extra_par)

; change the xtickformat if the panel is the bottom one and needs to
; plot a time axis

  if panel(0) eq 0 then extra_plotPar.xtickformat='MJDT_DATE_4'

; You might have to add your own xtickformat for time axis
; labeling. If you stick to the papco convention of always having your plot
; time array having consecutive seconds from midnight of the start day, then
; you can use the pre-defined xtickformat functions defined in ticks.pro in
; papco_Lib/axis_label.
; You can at this point also define a custom ytickformat, or leave it out for
; IDL default plot behaviour.

; "down" sets the axis labeling level.

  down=0

; do color plot and color bar plot. nodata specifies the "nodata" flag
; in your data. Here we do not plot axis and data seperate, as
; papco_plot_colorspec cannot be used as an overplot!

  ;check the routine papco_plot_colorspec in papco_XX\papco_lib\color_plot
  ;for all the possible options of papco_plot_colorspec.
  ;Hint: you might want to use the resx, resy keywords, which set the
  ;resolution to plot add. Without these, the routine will try to use
  ;device resolution, which for postscript may result in HUGE files!
  ;resx, resy are in data coordinates. If the axis is logarithmic,
  ;resx, resy need to be too.
  resx=20
  resy=0.05
  papco_plot_colorspec,zmat,time,yarr, nodata=nodata, resx=resx, resy=resy, $
    _extra=extra_plotPar
  papco_colorbar,zmin,zmax,0,0,uztit

; store the coordinate information into common block

  plotted_x = !x
  plotted_y = !y

; plot extra x-axis labels (eg. ephemeris) and a x-axis label if required.
; plot in color=1. Example here adds an axis with date information.
; If you stick to the papco convention of always having your plot
; time array having consecutive seconds from midnight of the start day, then
; you can use the pre-defined xtickformat functions defined in ticks.pro in
; papco_Lib/axis_label - Or use any other function you define yourself.
; The routine x_side_label puts x-axis label to the right of the axis.

  if (panel(0) eq 0) and not keyword_set(OVERPLOT) then begin
    extra_plotPar.xtickformat='MJDT_DATE_3'
    extra_plotPar.color=1
```

```
    down=1
    plot,time,time, _extra=extra_plotPar, /NODATA
    x_side_label,panel,'time (UT)!CDate'
  endif

; Since here papco_colorbars are plotted, labels are rotated by 90
; degrees and then plotted. No allowance is made for overplots, as
; color plots cannot be an overplot.

  right_side_label,panel,utitle,/rot90

; check if data needs to be output to file. Use the optional keywords
; to add more information to the data file
;      DESCRIPTION   a string or string array describing the data
;      TIME_LABEL    a string decribing the x-axis format
;      CHANNELS      an (ch,2) array giving the channel start/end
;      Y_LABEL       a string with the y-axis label
;      Z_LABEL       a string with the z-axis label
;      RS_LABEL      a string with the righ-side label of the plot
;      NODATA        contains the no data flag value.

  if KEYWORD_SET(OUTPUT) then begin
    print,'% p_plot_type_spec: Writing plot data out to file'
    description='plot_type template sample data - Energy spectra, 10 channels'
    time_label='Seconds since start of day'
    channels=yarr
    y_label=uytitle
    z_label=uztit
    rs_label=utitle
    nodata=0
    papco_write_data, time, zmat, $
                      DESCRIPTION=description, TIME_LABEL=time_label, $
                      CHANNELS=channels, Y_LABEL=y_label, Z_LABEL=z_label, $
                      RS_LABEL=rs_label, NODATA=nodata
  endif

end
```

B.2.4 defaults.config

This is a simple text file used to configure the module. This is done through the use of environmental variables, which point to the data directories or remote sites for use with wGet, or can be used to set other user-defined options in the module (such as enable/disable remote data get).

Each line in this file has the name of a global variable and it's definition. For the `plot_type` template the the variables needed are the one contaning the data path and the one containing the URL string to be used with the remote get data facility. The default settings that come with a module might not be correct for your installation, so you will need to use the config function of the `plot_type` panel editor to change it (replace `papco_XX` by the correct path to your PAPCO site).

```
PLOT_TYPE_DATA /some_host/some_directory//plot_type
PLOT_TYPE_FTP_ARCHIVE ftp://papco.lanl.gov/papco/data/plot_type/
```

wGet URL

The URL string used here needs a little explanation. The URL has the general format of

`ftp://user:password@host//data_dir`

Where

`user:password` are the account name and password for the site. If your URL contains exactly the string "user:password" here, then PAPCO will prompt you for the user and passowrd name. The idea is not to EVER put an password into a file - if you want to protect your site, make at least sure thast you never ship a module which has pasword information explicitly in it!

When `user:password` are omitted (us in this example), wGet will attempt to use anonymous ftp.

`host` is the internet address, in this case `papco.lanl.gov`.

`data_dir` is the full path to the directory containing the data.

For further information refer to the wGet manual.

You don't have wGet?

wGet is a GNU freeware utility for UNIX freeley available on the web and should also be available in some form for Win95.

If you cannot use wGet on your site then you'll have to do without the remote get data facility of PAPCO. You can, however, still access the files needed outside of PAPCO using normal ftp. For the `plot_type` module simply ftp to

`papco.lanl.gov`, login as user anonymous

`cd pub/papco/data/plot_type`

`bin`

`get plot_type_970111.idl`

B.3 mission_createorbitinfo.pro

Template for adding your own satellite mission to the PAPCO "read by orbit" feature. This file resides in papco_XX/papco_lib.

```
;******************************************************************************
;* PROCEDURE:   mission_createOrbitInfo
;*
;* DESCRIPTION: for all MISSION-ephemeris-files in the directory MISSION_EPH,
;*         orbit-infos are created and written to the file
;*              $MISSION_EPH/orbits.info
;*
;* INPUTS:      none
;*
;* OUTPUT: none
;*
;* CALLING SEQUENCE: mission_createOrbitInfo
;*
;* MODIFICATION HISTORY: December 1997, written by R. Friedel
;******************************************************************************
PRO mission_createOrbitInfo

   common mission_ephem, mission_ephem

; search for the MISSION-data-files. Use this code to get a list of all the
; ephemeris files available for MISSION. Replace with correct file template
; for your files in findfile below.

   print, 'Compiling list of all MISSION ephemeris files...'
   mission_path=papco_getenv('MISSION_EPH')
   allMISSIONFiles=findfile(mission_path+'MISSION_EPH_*.DAT')
   print, '...found', N_ELEMENTS(allMISSIONFiles), ' ephemeris-files'

; Depending on your MISSION ephemeris files, loop through them to find orbit
; start and end times. One way of doing this is to construct an array of
; radial distance. Max. of radius are then apogee times, Min perigee times.


;     Enter user code at this point
;     to construct an array of orbit start and end times
;     Assume there are ndat orbits, then continue:


; create a MISSION_ORBIT_INFO-structure and create an array of those

   tmp=papco_getOrbitInfoStruct()
   orbitInfo=replicate(tmp, ndat)

      for i=0l,ndat do begin
          ;
          ; format your orbit start /end times such that
          ; i      = orbit number
          ; day_oy = is the day of year of start of orbit
          ; year   = is the year of start of orbit
          ; xut1   = is the no of sec from the start day of the start of orbit
          ; xut2   = is the no of sec from the start day of the end of orbit
          ;
          ; Its up to you to define orbits from apogee to apogee or
          ; from perigee to perigee
          ;
          ; save the data in the orbitInfo-structure
          ;
          orbitInfo(i).number=i
          orbitInfo(i).day=day_oy
          orbitInfo(i).year=year
          orbitInfo(i).xut1=ap_time
          orbitInfo(i).xut2=ap_time + 63120l
          orbitInfo(i).xdata=0 ;dummy, don't use, here for old compatibility
          orbitInfo(i).ydata=0 ;dummy, don't use, here for old compatibility
      endfor

; write the orbits.info-file

   result=papco_writeOrbitInfo('MISSION', orbitInfo)
   IF result EQ 'OK' THEN $
      print, 'OrbitInfo successfully created.' $
   ELSE $
```

```
      print, result
   RETURN

END; (mission_createOrbitInfo)
```

Appendix C

Other useful PaPCo routines

This chapter includes a brief description of some of the routines which are distributed with PaPCo and which might be useful to the module developer.

C.1 Color plot routines

You can use ANY color plot routine you want in your module. PaPCo does not force you to use it's functionality or to adhere to its conventions. However, if you want to have your module "look good" when used in combination with other modules (which is the whole purpose of PaPCo!) then using some of the PaPCo conventions makes sense. For color plot routines, these conventions mainly concern the use of color tables, in order to support the PaPCo color scheme (5.4).

To make life easier PaPCo supplies a set of color plot routines as part of its library in `papco_XX/papco_lib/color plot`. These routines have evolved over time, and all the ones ever used are still available. The module template `plot_type` always uses the latest set of these routines.

These routines come from many sources and are provided here as a service - but use at own risk. The current "standard" set of color plot routines are

`papco_plot_colorspec.pro`
and `papco_colorbar.pro`

These routines came from the POLAR HYDRA module and are written and maintained by PaPCo Jeremy Faden[1] at University of Iowa.

C.1.1 papco_plot_colorspec.pro

This routine is fairly sophisticated and tested by now, and should serve well. It supports the PaPCo color scheme and makes an attempt at creating small postscript files. It also support either fixed or variable bin size in both x and y direction, and allows full control over the actual plot resolution.

For documentation on the color plot routine refer to the header in the file

`papco_XX/papco_lib/color plot/papco_plot_colorspec.pro`.

C.1.2 papco_colorbar.pro

This routine is used to place a color bar at the right-hand side of the panel. It supports the PaPCo color scheme and uses "nice" labeling.

For documentation on the colorbar routine refer to the header in the file

[1]"mailto:jbf@space-theory.physics.uiowa.edu"

papco_XX/papco_lib/color plot/papco_colorbar.pro.

C.2 PaPCo get remote data routine

This routine in PaPCo core uses the GNU freeware "wget" to obtain files from remote sites. It is up to the user to include this feature in his read routine. It has the following header:

```
PRO papco_get_remote_data, url, filename, download_dir
```

You need to supply the url (internet address), the filename to get (including path) and the local destination directory for the file. Transfers assume anonymous ftp, but you can of course rewrite this routine to allow for other sites that require user/password information.

It makes sense to use an environmental variable for the URL needed for you module, and to configure the module with this in its `defaults.config` file (see Section B.2.4 for more details).

This routine is self documenting. It provides some widgets to inform the user that files need to be got remotely, and provides some download status and an estimate of the time it will take to transfer.

See the omni module for an example of the implementation of this feature.

NB. This feature is not designed to obtain huge amounts of data over the net, as it will only get the data for the plot you're busy with. It has been made interactive on purpose, so you can't run a batch that will get a whole lot of remote data. If you need a large data set contact the curator of that dataset and arrange to have it sent in a more convenient way, or ftp the data to your site manually outside of PaPCo.

C.3 PaPCo output routine

One of the caveats of ISTP support for PaPCo was that PaPCo be able to produce output in digital form for use outside of PaPCo .

We have written a new PaPCo utility which can take the data arrays that are plotted and dump them out to file in a standard manner. This is done by providing an output routine which can be called at the point the data is plotted and which will write the data to file.

The user calls the `papco_output` routine and hands over the time and either one or two dimensional data arrays. A range of keywords can be set to add more information to the output product. For more information look at the header of the file `papco_XX/papco_output.pro`.

To use this feature press the "Output" action button in the PaPCo main widget - this will execute a normal draw loop and output all the data for the current PaPCo product into a single file (only for those modules in the product that support this feature). This data file can be written in either text, IDL saveset or CDF format (CDF format is still being implemented). The file name for this file is constructed using the current product name (set to "none" if no product name is defined) and the time range of the plot. The extension of the file indicates the file format, ".ascii" or ".idl" or ".cdf". The file is placed into the default output location set by the print widget.

As PaPCo is a distributed software project it is up to each module to support this feature in their draw routines. This feature can be transparently added to existing draw routines by including the keyword `\OUTPUT` to the `draw_plot_type` PaPCo interface routine and to the user's plot routine.

This feature is enabled by adding a new PaPCo interface routine (`canget_plot_type`) to "tell" PaPCo that the module supports this feature, into the modules `papcoadd_plot_type` file. All this interface routine does is set a variable "canget" to 1 to enable or to 0 to disable this feature. If this routine does not exist, this feature is disabled by default.

Finally all the user has to do is to add the lines

```
 if KEYWORD_SET(OUTPUT) then begin
   print,'% p_plot_type_spec: Writing plot data out to file'
```

```
    description='plot_type template sample data - Energy spectra, 10 channels'
    time_label='Seconds since start of day'
    channels=yarr
    y_label=uytitle
    z_label=uztit
    rs_label=utitle
    nodata=0
    papco_write_data, time, zmat, $
                      DESCRIPTION=description, TIME_LABEL=time_label, $
                      CHANNELS=channels, $
                      Y_LABEL=y_label, Z_LABEL=z_label, $
                      RS_LABEL=rs_label, NODATA=nodata
  endif
```

at the end of his/her plot routine. The above example is taken from the current module template `plot_type`, which implements this feature.

To find out more about how to implement this feature look at the module template `plot_type` and the documentation included within the code.

Appendix D

Files needed by PaPCo

D.1 Core PaPCo

In file papco_files_needed.text we provide a short description of the main files which are needed by PaPCo. The files are roughly divided into four categories: PaPCo "Core", which is the main system and should not be altered by the user. PaPCo Lib, a set of utility files that could be recycled by other applications too. PaPCo modules, user written code that actually does plots, and PaPCo doc, for documentation.

```
THIS FILE:
   papco_XX/papco_doc/papco_files_needed.text

   Describes the files of PAPCO.

MODIFICATION HISTORY:
   April 1996,     written by R.Friedel
   June  1996,     modified by R.Friedel
   March 1997,     modified by R.Friedel
   September 1997, modified by R.Friedel
   December 1997,  modified by R.Friedel
   August 1998,    modified by R.Friedel

------------------------------------------------------------------------------
compiling PAPCO (files in papco_user_XX/papco)
------------------------------------------------------------------------------

papco_add_init.pro       which complies all the required modules to be
                         added into PAPCO.
                         This file is rewritten each time PAPCO is started.


------------------------------------------------------------------------------
compiling PAPCO (files in papco_XX/papco)
------------------------------------------------------------------------------
papco_init.pro           initialize PAPCO and compile all needed programs
                         from IDL, type @papco_init - which also calls

find_modules.pro         searches for modules in !path and writes
                         papco_add_init.pro in user's papco_user_XX directory.


------------------------------------------------------------------------------
basic utilities needed by PAPCO (files in papco_XX/papco_lib)
------------------------------------------------------------------------------
convert_time.pro         utilities for converting time into strings. Supports
                         MJDT and old  T90-formats.

papco_util.pro           contains some utilities...

messagebox.pro           simple widget for providing message windows

entrybox.pro             simple widget for entry of variable values

panelset.pro             sets the viewport for plot based on the panel vector
```

```
set_device.pro          sets the device for ps, Z, X, etc

string_functions.pro    a collection of useful string manipulation routines

plotout.pro             sends output files to printers / gif files


--------------------------------------------------------------------------------
utilities needed by the plot-routines (files in papco_XX/papco_lib)
--------------------------------------------------------------------------------
cnvrttim.pro  \
daycnvrt.pro   >        old time-conversion routines
cnvrttim.pro  /
timcnvrt.pro /

/axis_label             subdirectory containing tick-marks routines

/color_plot             subdirectory containing color plot routines
 papco_plot_colorspec.pro  new standard papco color plot routine
 papco_colorbar            new standard papco colorbar routine
 plot_clr.pro              color spectrum plot routine by using the IDL
                             image functions
 plot_clr_old.pro          the old color spectrum plot routine using poly-fill.
                             Suitable for small plots with many gaps.
 colorbar.pro              draws a color bar for color plots
 to_display_size.pro       resizes plot arrays to display resolution

rf_plot.pro             a wrapper routine on plot to draw line plots with gaps

x_side_label.pro        annotates plot x-axis to the right of the x-axis

right_side_label.pro    annotates plot next to panel top right


--------------------------------------------------------------------------------
other windows for PAPCO
--------------------------------------------------------------------------------

crres_createorbitinfo   makes the orbitinfo file for CRRES

polar_createorbitinfo   makes the orbitinfo file for POLAR


--------------------------------------------------------------------------------
PAPCO main - program
--------------------------------------------------------------------------------

papco_forward.pro        this file *must* be executed (@papco_forward), before
                         the rest of PAPCO is run. Contained hare are some
                         forward_function-commands which declare PAPCO-
                         functions.  This file is run implicitly when executing
                         @papco_init. You can start it from IDL by typing
                         @papco_forward

papco_printer.pro        the widgets for selecting an output-medium

papco_defaults.pro       contains the widget for setting some defaults

papco_cursor.pro         implements functions for zooming or slicing in
                         PAPCO-plots

papco_edit_type.pro      change the plot-type of a panel

papco_draw.pro           draw the selected plots

papco_settime.pro        functions for choosing the time-range

papco_files.pro          functions for file-related PAPCO-operations

papco_plotedit.pro       functions for editing, adding, deleting panel-plots
                         to PAPCO

papco_editslots.pro      changing the panel editor list

papco_draw_window.pro    contains the draw window widget and event handler

papco_color.pro          contains the routines for managing the internal color
                         sub_table structure
```

```
papco_configure.pro      contains the startup PAPCO configure widget routines

papco_module_configure.pro  contains the routines for configuring modules

papco_get_remote_data.pro   contains routines based on wget to get remote
                            data via ftp, plus utility windows

papco_orbitinfo.pro         contains routines to read orbit info files and
                            enable the read by orbit feature

papco_opsys.pro             contains all reoutines needed to make PAPCO
                            operating system independent.

papco_output.pro            output procedure to write data to file.

papco.pro                   This file contains the main parts of PAPCO, the
                            Panelplot-Composer.


             *******************************************
             **    help - files needed by PAPCO       **
             *******************************************

papco_cursor_rect.help
papco_edit_type.help
papco_main.help
papco_printer.help
papco_slice.help


             *******************************************
             **    other files needed by PAPCO        **
             *******************************************

crres_orbits.info        this is a file created by the procedure
                         crres_createOrbitInfo
                         It's contained in $CRRES_FILES/orbits.info
                         and allows the used to choose time per CRRES orbit

polar_orbits.info        this is a file created by the procedure
                         polar_createOrbitInfo
                         It's contained in $POLAREPHMPATH/orbits.info
                         and allows the used to choose time per POLAR orbit

step.info                contains info about observers and regions on the
                         STEP-CDs. For information on this, see the file
                         step_read_observers.pro
                         It's contained in the step_6.4 module under plot_read

papco.tbl                The default color tables used by PAPCO

papco_printer_config.dat this file contains the printer configuration. All
                         available printers and their print commands have
                         to be entered in this file and separated by a comma.
                         Lines starting with a colon will be ignored.
                         You are only allowed to enter one printer and
                         its print command per line.
                         E.g., for a printer with the name Sprotte and the
                         print command 'lp -d sprotte' the line in the
                         configuration file would be:
                         Sprotte,lp -d sprotte
                         If it does not exist, PAPCo is configured without
                         printers and you can only print to file.

papco_defaults.dat       contains all the default settings that enable PAPCO to
                         remember its last state. Many of these are
                         configurable using the defaults widget.
                         If it does not exist, PAPCO starts with built in
                         defaults and writes this files upon exiting.
```

D.2 The directory structure of PAPCO

In file papco_directoires.text we give an overview of the PAPCO directory structure.

```
THIS FILE:
```

```
   $papco_XX/papco_doc/papco_directories.text

   Describes the directory structure of PAPCO.

MODIFICATION HISTORY:
   April 1996,    written  by R.Friedel
   June  1996,    modified by R.Friedel
   March 1997,    modified by R.Friedel
   December 1997, modified by R.Friedel

All of PAPCO should be placed in one directory called papco_XX

The PAPCO package is organized into four basic sub-directories. These are:


/papco           Contains all routines for the widget interface. These routines
                 are called the PAPCO "core" and should not be altered by the
                 user.
                 There is one sub-directory:

                 /help          contains help files for PAPCO core


/papco_lib       Contains utility programs which are PAPCO non-specific
                 and could be recycled by other applications


/papco_modules   Contains all the add-in modules written by users. All modules
                 have the same directory structure, and each module is kept in
                 its own directory. For example, a module called YOUR_MODULE
                 has the following structure:

                 /YOUR_MODULE   contains all that is needed for
                                a module. Each one of these directories
                                need to contain the following two
                                sub-directories:

                                /papco_interface  for all interface routines
                                                  and module help files
                                /plot_read        for the plot and read
                                                  routines

                 One of the modules included here is the user's development
                 module. It's called "plot_type" and is a fully functional
                 module that works with an included single data file for
                 date Jan 11 1997.


/papco_doc       contains all the documentation for PAPCO. There is one
                 sub-directory:

                 /comments_for  contains the output from analyzing the PAPCO
                                IDL file tree using the program commentsFor.pro
```