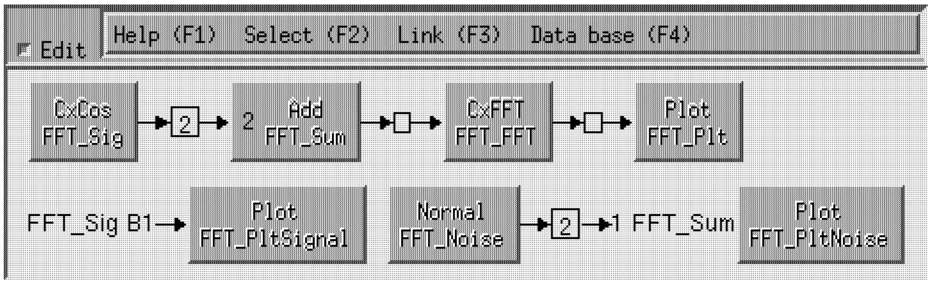
ObjectProDSP User's Reference

Paul P. Budnik Jr. Phd.
Internet: support@MTNMATH.COM

September 1994

'.dvi' file created September 16, 1994



Mountain
Math
Software
P. O. Box 2124, Saratoga, CA 95070
Fax or voice (408) 353-3989

Published by Mountain Math Software, P. O. Box 2124, Saratoga, CA 95070.

Copyright © 1994 by Mountain Math Software. All rights reserved.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled “GNU General Public License” and “Licensing” are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one, and provided the derived work is clearly identified as a derived work and not solely the creation of either the orignal authors or the authors of the derived work.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the sections entitled “GNU General Public License” and “Licensing”, and this permission notice, may be included in translations approved by Mountain Math Software instead of in the original English. Translations of the section entitled “GNU General Public License” must also be approved by the Free Software Foundation which owns the copyright to that text.

Licensing

ObjectProDSP^{TM} is licensed for free use and distribution under version 2 of the GNU General Public License. See Appendix D for the full text of this license. There is absolutely no warranty for ObjectProDSP under this license. ObjectProDSP is a trademark of Mountain Math Software.

You are free to use and distribute ObjectProDSP under the terms of version 2 of the GNU General Public License. Please note that *none* of the ObjectProDSP system is licensed for use under the GNU *Library* General Public License. The Gnu General Public License allows you to distribute executables or librarys linked with or created by ObjectProDSP *only* if you make *all* the *source* code used to create the librarys or executables (other than standard librarys that are part of a compiler or operating system) freely available. Please read the license in Appendix D for the full legal explanation of these conditions.

Mountain Math Software plans to offer, for a fee, a commercial version that will allow you to distribute executables generated with ObjectProDSP under standard commercial terms.

If you wish to extend ObjectProDSP you can distribute your code with ObjectProDSP under the terms of the GNU General Public License. If you include an appropriate copyright notice in your name for your upgrades then no one, including Mountain Math Software, will be able to distribute your code under any terms other than the GNU General Public License without your permission.

If you find ObjectProDSP useful in a commercial environment you are asked to consider purchasing a support contract. This is not shareware and you are under no obligation to do so but you will gain aceess to direct support from Mountain Math Software and you will make a contribution to the continued success of ObjectProDSP and thus to any of your endeavors that benefit from it.

If you are interrested in a custom port of ObjectProDSP to directly support your company's DSP development board or processor please contact us.

Mountain Math Software
P. O. Box 2124
Saratoga, CA 95070
Internet: support@mtnmath.com
Fax or voice (408) 353-3989

Documentation

- *ObjectProDSP Overview and Tutorial* This gives a general description of ObjectProDSP's purpose and function. It includes several tutorial examples. There are appendices on the DSP node and class library and Mountain Math Software.
- *ObjectProDSP User's Reference* This is the document you are reading. This describes the user interface and DSP++, a C++ based language for DSP. (You do not need to known DSP++ or C++ to use ObjectProDSP. DSP++ statements are generated for you when you graphically enter a network or execute menu data base commands.) This document includes a reference manual for the menu data base. Appendixes contain a synopsis of menu data base commands and a general index.
- *ObjectProDSP Library Reference* This gives a detailed description of ObjectProDSP interactive objects including DSP processing nodes.
- *ObjectProDSP Developer's Reference* This tells how to write DSP processing nodes and add them to ObjectProDSP. It describes ObjectPro++^{TM}, an extended C++ language for defining interactive objects for DSP or other applications. It explains how to modify the part of the menu data base that does not come from interactive object definitions in ObjectPro++. It describes how to update the ObjectProDSP manuals to include your new nodes and objects. Information about these objects is extracted from your definitions by ObjectPro++ and added to the manuals.

Contents

1 Using this manual

The ObjectProDSP user interface, signal processing language and menu data base are described in this manual. Section 2 describes the basics of starting the program, and gives an introduction to using it. Section 3 contains the release notes for this version. Section 4 describes how to graphically create and edit a network and view the output. Section 5 gives an overview of the menu data base. Section 6 describes the generic member functions. The information in the above sections is available as on-line help screens.

Section 7 describes the ObjectProDSP language for Digital Signal Processing. Language statements are created by interactively editing a network and though the menu database. You do not need to know the DSP++ language but you may find it of interest.

The second part of this manual is a printed version of the menu data base. This part of the manual is created automatically from the same source used to create the interactive data base.

Appendix A describes the environmental varialbes used by ObjectProDSP. Appendix B is an index of ObjectProDSP menu commands. It includes a brief synopsis of each command. Appendix D is version 2 of the GNU General Public License.

2 Using ObjectProDSP

In this section we describe how to start ObjectProDSP and give a brief overview of how to use it.

2.1 Getting started

Environmental variable `OPD_ROOT` must be set to the root directory where ObjectProDSP is installed.

To execute ObjectProDSP enter the unix command '`opd`'. This assumes that `opd` or a link to it is installed in your execution path.

ObjectProDSP automatically saves it state on exiting. The state includes any networks or other user defined objects. These are restored by reading the default state file when you next execute ObjectProDSP in the same directory.

You can override the loading of the default state and action files with command line options. (An action file allows you to record your actions during a session as opposed to a state file which only records the final state that resulted from those actions.) The default file to store the state is `dsppp.0`. Previous backups are kept in `dsppp.N` where the larger N is the older the backup is. The default is to save the four most recent backups. Up to 10 backups can be saved. The default action file is `.opdinit`.

The following command line parameters are optional.

-c do not read a default state or action file.

-r specify an initial state file (do not use the default).

-a specify an initial action file (do not use the default).

-i use the 16 bit integer simulator instead of the default single precision floating point version.

-e describes the environmental variables used by ObjectProDSP.

-d run `gdb` on the DSP process.

-g run `gdb` on the user interface process.

-T use the command line version of `gdb`.

2.2 Introduction to ObjectProDSP

ObjectProDSP is an object oriented tool for Digital Signal Processing (DSP) design analysis and implementation. You can graphically define a DSP network, provide test signals and view the output.

You can generate a stand alone C++ program that performs the functions of this network and link this program into your DSP application.

With the source code distribution you can define your own DSP nodes or other interactive objects.

The interface is designed to be self teaching. Access to most functions is through a menu data base. Instructions for each window and the most important functions in a window are available in pull down menus.

ObjectProDSP runs as two Unix processes: user interface and DSP execution. You can interrogate the menu data base, manipulate data plots and perform other operations in parallel with DSP execution.

2.2.1 It is easy to create and edit a DSP network

To get a quick idea of what you can do with ObjectProDSP select `help` (this is already selected when ObjectProDSP starts execution) `examples`, `fft` and `execute` by clicking the left mouse button over the push buttons in the menu data base. A window with a DSP network and three windows of plotted output will appear. Select the plot labeled `FFT_Plt_plot_of_FFT_FFT` and type `ctrl-b` (hold the `ctrl` key down and type `b`). A new window will appear with the same data on a decibel power scale.

To add a fourth plot of the input data to the FFT do the following. Select `objects` and `plot` from the menu data base. Then hold the shift key down and click the left mouse button with the cursor over the `Plot` button. An unlinked plot node will appear in the network window. Hold the shift key down while clicking the right mouse button over the node labeled `FFT_Sum` in the network window. The selected node will be highlighted and a message will appear in the information window saying: Linking output buffer on output channel 0 in node `FFT_Sum`. Hold the shift key down and click the left mouse button over the unlinked plot node in the network. The plot node will be linked into the network at the point you selected.

To execute the edited network select `database` and `network` from the menu bar in the network window. A separate window with commands for controlling the network will appear. Select `exec` and `Execute`. You will be prompted for the number of input samples to generate. Type RETURN to select the default. A new window will appear with the plot of the FFT input data and the other plots will be updated with the new data you generated.

2.2.2 The main window

ObjectProDSP starts with two windows. The `information` window displays help for most operations. It is frequently updated.

The main window contains four parts. At the top are pull down menus that control and document the window. Next is a section for entry of DSP++ language statements. ObjectProDSP signal processing operations are defined in C++ using classes and member functions. This internal DSP extension of C++ is called DSP++. Most often DSP++ statements are generated by menu selections or by graphically editing a network. You can enter statements directly by selecting the highlighted line at the bottom of this section and typing language statements. For example you can enter `double A = (Pi+3.5)/4;` Statements are scrolled up in this section as are messages related to execution of these statements. The history of all text in this window is available through the `create DSP++ history window` option under the `Data base` pull down menu.

The next section in the main window contains a record of up to the last three commands executed from the menu data base and messages related to command execution. The history of all text in this window is available through the `create command history window` option under the `Data base` pull down menu. You can enter commands directly in the highlighted line in this section. Commands will first be searched for in the visible menu data base buttons and then the full data base will be searched. If you enter a valid unique command the corresponding command will be executed. A command that selects a menu and is not on the visible data base buttons will cause the selected menu to appear in a separate window. For example if you type `CxFFT` the menu for that class will appear in a window.

To directly enter DSP++ statements or menu commands use the pull down menu `select` or click the left mouse in the text entry area. To return keyboard input to menu shortcut keys type the `escape` key.

The last section of the main window provide push button access to the menu data base. From this data base you can control most ObjectProDSP functions.

2.2.3 Windows management

It is easy to create many windows. There are some facilities for managing windows. You can delete some windows (type `ctrl-shift delete`), but not any windows containing network output. You can `raise` windows (make them appear above any overlapping windows) with options in the network window pull down menus and the menu data base. This can be helpful in locating the window output form a particular node. For example to raise a plot or listing node position the mouse over the plot node in the network window and type upper case `W` (shift-W).

2.3 Pull down menu codes

The most important commands for most windows are documented and available in the menu bars at the top of the window. There is a simple coding scheme for shortcut keystrokes and mouse shortcuts. Mouse shortcuts (if available) are in square brackets [ ]. Keyboard shortcuts are in round brackets `( )`. Keyboard shortcuts are denoted as a single character like `(p)` for lower case `p` or a word that designates the key like `(space)` for the space bar. A key can be modified with the shift or control keys. `(shift-space)` means hold the shift key down and press the space bar. For the most part you can enter shortcut keys whenever the window is selected and the cursor positioned within it. However there are some `dead spots` in some windows. If you find that shortcut keys are having no effect move the cursor around. Known dead spots are in blank space in the right part of network displays and in the transition between sections in the main menu. Short cut keys will not work in the main menu if you have selected keyboard entry for one of the two text areas. Type `escape` to return to short cut keys in the main window.

Mouse buttons are denoted as [`L`] (left), [`M`] (middle) and [`R`] (right). Most mouse actions occur when the button is released. Any that occurs when a button is pressed are preceded by the letter D. Mouse buttons are modified with either or both of the shift and control keys. [ctrl-L] means hold the control key down while pressing and releasing the left button. `shift` is generally used for commands related to editing networks and `ctrl` for commands to access the menu database.

2.4 Moving though a help window

Help windows, pop up windows and the information window are the only ones that do not have a menu bar at the top to document their use. In addition to using the scroll bar you can use the following keys to move through the help and information windows: page up, page down, home (beginning of file), end, up arrow, down arrow. You can delete a help window by typing `ctrl-shift-delete`.

2.5 The menu data base

The two or three rows of buttons in the lower part of the main window provide access to the menu data base. Most functions are available in this data base. Objects you create interactively are added to it. Documentation is integrated at three levels. Moving the mouse over a button produces a brief description above the button. Pressing the left or right mouse button while the cursor is over a selection usually displays a more complete description of the selection in the information window. Some selections have an associated help file. Clicking the right mouse button will display this file in a new window.

The information window contents is frequently updated. Usually any information that you might want to reference later is also scrolled into the DSP++ command history. If you want to preserve something in the information window that has not been scrolled use the `append information window to DSP++ history option` under the `Data base` menu.

You can `peel off` lines of the help data base menu to appear in a separate window. Hold the control key down and select the menu with the left mouse button. If the selection is a menu then that menu will appear in a separate window otherwise the menu that contains this selection will appear. There are never more than three rows of buttons displayed in the main window. If you go deeper only the top level and the deepest two levels will be displayed. Database menus in a separate window display as many levels as you select up to the height of the screen.

A menu data base button is selected when you release the left mouse button (or type RETURN) with the cursor positioned inside the button. Buttons that execute commands are in boldface. Other buttons select new menus.

Commands can be entered directly in the lower highlighted area of the main window. Just click the mouse in this area and type the command. First the visible commands will be searched and then the entire data base. Menus selected in this way are `peeled off` into a separate window if the selection is not among the displayed buttons. Type the `escape` key to stop entering commands and return input to pull down menu shortcut keys.

2.6 Editing a DSP network

The menu database provides information about nodes and networks. A button in the database labeled with a node class name (select `objects` and then `dsp processing` to see the processing node classes) allows you to add an instance of this class to a network. If no network is being edited a new network will be created. You can select the default parameters of the node or you can be prompted for each parameter. The pull down menu `Edit and control` shows how to do this. The help files in the network editing window tell you how to edit an existing network. To create a network editing window hold the shift key down and press the left mouse button selecting any class (`other` is not a class) under the `dsp processing` menu. This will create a new network window with an instance of the selected object.

2.7 Aborting commands and deleting windows

You can abort commands which prompt you for input by typing the `escape` key. You can abort the current DSP command by selecting `abort` from the `state` menu. You can delete `help`, `information` and `menu` windows by selecting the window to be deleted and typing `delete` while holding the control and shift keys down.

2.8 Saving window and display images

You can save any window (except pop up windows) and the entire X-window display to a disk file. This file can be viewed using `xwud` or converted to postscript format for printing with `xpr`. The window save commands use the utility `$OPD_X_BIN/xwd`. This is assumed to be in directory `$OPD_X_BIN` (which defaults to `/usr/bin/X11` if `OPD_X_BIN` is not defined in the environment). If that program is not at that location no files will be generated.

To save the current window type `ctrl-shift-*`. To save the current window without first raising it above overlapping windows type `ctrl-shift-&`. These commands are accessible in the pull down menus in the main window under `Edit and control`. The first of these commands is in the pull down menus for the network and plot menus. Both commands work for all windows. The save window commands are among the few commands not accessible through the menu data base.

To save the entire display type `ctrl-shift-^` in the main window.

2.9 Help Structure

The help facility consists of the following:

1. Pull down menus at the top of most windows document the main commands in the window. These commands can be accessed using the pull down menus and with shortcut keys and mouse commands listed in the pull down menus.
2. A subtree in the menu data base serves as an online reference manual.
3. Confirmation and other messages are displayed in response to user commands. You can control how much of this information is displayed by changing the help level.
4. Help windows are available for many data base menu selections by pressing the right mouse button.
5. All user defined objects are added to the menu data base. You can list the objects you defined and their parameters.
6. Example DSP`++` programs and ObjectProDSP sessions are provided.

The multiple methods for providing information is intended to make it quick and convenient to obtain the information you need.

2.9.1 Pull down menus

Most windows have pull down menus at the top. To access a pull down menu click on it. The first menu in each window is labeled `Help` and provides documentation of the window. The others contain the main commands in the window. These commands are accessible through the pull down menus and though short cut keys and mouse buttons. Each menu item gives the short cut keys for itself. The coding for key and mouse short cuts is described in the `pull down menu codes` selection from the `help` pull down menu in the main window.

2.9.2 Help menu database tree

All selections in the menu data base have one line of help information displayed when the cursor is over the help button. Many display a short paragraph of information when the left mouse button is pressed. Some display a help window when the right mouse button is released.

Selecting help in the top level of the menu data base provides access to an online reference manual organized by topics.

2.9.3 Message levels

Selecting `help` and then `help levels` gives the menu of three available help levels. The default, `help all`, provides hints and confirmation messages that may be helpful to the new user. `help confirm` confirms most commands. `help none` only displays error messages.

Some messages are displayed redundantly in both the information window and scrolled in the main window. This allows related messages such as all those associated with a single statement to be collected. All messages are preserved in a history file that can be viewed by selecting `create DSP++ history window` from the `Data base` pull down menu.

2.9.4 State description

In building a DSP network you are defining objects in a global name space. This space starts with some predefined objects. These are primarily DSP processing nodes and facilities for generating and displaying data. The real number π is defined in the variable ‘Pi’.

These objects are organized in a hierarchical tree structure.

These description are accessible in the menu data base from the `objects` button.

3 Release notes and validation

This section contains the release notes and information about validation suites.

3.1 Release notes for ObjectProDSP version 0.1

This is the first public beta release of ObjectProDSP. Although this is a beta release and some problems are to be expected, considerable effort was put into validating this release to make it a useful and valuable tool. Earlier versions have been used by Mountain Math Software in consulting projects. Many useful ideas that came from those projects have been incorporated in this version.

.

3.1.1 System requiremnts

3.1.1.1 Memory ObjectProDSP runs well with 16 megabytes of physical memory and 30 megabytes of swap space. We have also run it with 20 megabytes of physical memory and no swap space but we needed to be careful about what else is running. We suggest 16 megabytes of physical memory and a 16 megabyte swap partition. The program cannot recover if it runs out of memory but you can automatically and frequently save the state.

3.1.1.2 Disk space Binary installation: 8 megabytes

Source float installation: 31 megabytes

Source float and int16 installation: 37 megabytes

There is only one source distribution that includes the documentation. The above sizes are total requirements after you have dearchived the distributions and built the executables.

The following are the additional disk space needed to build the documentation and run the validation tests.

Float only validation or test data creation: 34 megabytes

Float and int16 validation or test data creation: 38 megabytes

Documentation: 20 megabytes

This is the additional space to build `.dvi` but not postscript files from the source distribution. However it does include the postscript format files of the X-windows images. They are converted from the more compact `xwud` format when the manuals are created. These are needed to build the `.dvi` files and they require 15 of the 20 megabytes of space.

The documentation component of the distribution contains only the postscript files. These require an additional 20 megabytes if you decompress them again because of the X-windows images. You may be able to pipe them directly to your printer with `gzip` without creating the uncompressed files.

3.1.1.3 Other hardware ObjectProDSP can be used (awkwardly) with standard VGA resolution. The higher the resolution and the larger the monitor the better. A resolution of at least 1024 x 768 is recommended. A color monitor and display card is required. The InterViews `monochrome` mode is not supported and does not work. Although a high performance graphics card is desirable graphics performance should be reasonable with almost any card.

Hardware floating point is recommended.

3.1.1.4 Software requirements The binary distribution requires XFree86 2.0 and a version of Linux that supports XFree86 2.0. It make work with earlier versions of XFree86 but we have not tested this. If you want to create stand alone executables you also need gcc 2.5.8, libc 4.4.24 and the version of libg++ that comes with Slackware 1.2.0. Other versions may work but this has not been tested.

The binary release for version 0.1 is currently available only for Linux. It should be possible to port ObjectProDSP to any system on which InterViews version 3.1 is available.

.

3.1.1.5 Using gdb You can run either or both of the two ObjectProDSP processes with the GNU debugger. If large numbers of programs are not compiled with the `-O` option under gcc-2.5.8 the executable become large and difficult to work with using the debugger. This is due to the large number of inline functions defined inside class definitions. These are made into static functions in each module if `-O` is not used.

3.1.2 DSP Node library and target code efficiency

Although there are many nodes in the existing library this is one area that we hope to expand quickly. For example we do not yet have an IIR filter or a FIR filter for non integer resampling. The list of possibilities is almost endless. It is usually straightforward and simple to make an ObjectProDSP node from an existing C or C++ kernel algorithm. If you are aware of public domain or GPL licensed code that would make good candidates let us know. Everyone is encouraged to write their own nodes and release them under the GPL.

The easiest way to write a node is to use routines `ReadWord` and `WriteWord` for every input and output operation. This is simple but results in fairly inefficient code. Most of the exiting nodes are written in this way. It is not difficult operate on chunks of data in buffers and this significantly improves performance. Routines are available to determine the amount of consecutive input and output available and there are special classes to simplify using this information. We hope to convert most of the nodes to operate on chunks of data in the near future. The existing nodes will usually not be suitable for demanding real time applications or other applications where high performance is critical.

3.1.3 Supported targets

All nodes are translated into two C++ programs. One is suitable for interactive use and the other for dedicated applications. You can automatically generate a stand alone executable of any network you create interactively.

This facility has been used to generate code for the TI TMS320C30 processor. The C++ code that ObjectProDSP generates was translated to C with the front end of a C++ compiler and this in turn was compiled with TI's C compiler. The stand alone target code should be portable to any system with a C++ compiler. With a C++ to C translator the code should be usable on any processor with a C compiler.

Currently there is no custom support for target processors. You must make any modifications needed for a particular target to the generic C++ code generated. The support for the TMS320C30 used a proprietary board and driver software for a customer of Mountain Math Software.

3.1.4 Filter design

Currently there is no filter design software integrated into ObjectProDSP. There are public domain packages available and it is simple to edit the output from these and to use as coefficients for a filter. We would eventually like to integrate one of more of these programs.

To use a filter designed elsewhere define the coefficients for a filter with a statement like the following:

```
MachWord * coeff[] = {.1, .2, .8};
```

Then define a complex fir filter with the following statement:

```
CxFir filter( 1, 0, 0.0, 0, coeff);
```

The above can be typed in the DSP++ statement entry window or can be entered in a file and read with the `read state` option in the `state` menu.

Alternatively you can use the `create` option and when you are prompted for the filter coefficients you can enter the name of the array `coeff`. (To create a `CxFir` node select `objects`, `dsp processing` and `CxFir`. Then click the right mouse button with the cursor over the `CxFir` button.)

3.1.5 Bugs

There is a know bug in the pop up menus. If a new window obscures a pop up window while it is the process of appearing, subsequent behavior of the menu bar is erratic and, if the window is rebuilt, ObjectProDSP can crash. If this happens you should immediately `save and exit` and then read the state back in and continue working. This problem is rare and can be avoided by not using the pop up windows when new windows are being created by the DSP process at the start of execution of a network.

3.1.6 Deleted TargetSystem capability

Previous versions had a `TargetSystem` capability that was used extensively in a previous project. We have not had time to test and update this so we removed it. Code that references and supports `TargetSystem` is still present. We only removed the user interface `.usr` files. We plan to restore this capability in a future release. `TargetSystem` refers to combining several networks in a single Target module. The generation of Target stand alone code for a single network is extensively tested in the validation suites.

3.2 ObjectProDSP validation suites

There are several regression tests you can use to check the installation or check to see if any source changes you made create new problems.

Some of the validation suites also have a limited use as tutorials. Select `pause` under the `state` and `session` menus to set a delay between each command. With this delay you can play back action file validation suites as a tutorial. (One limitation of this is you do not see the pop up windows that prompt for user input.)

The output from node validation tests are written with the node `CompareDisk`. This tests the current output against known good results. If any errors are encountered a file giving the list of errors is generated. See `README` in `$OPD_ROOT/validate` for more information. Go to `$OPD_ROOT/build` and enter `make VALIDATE` to run the full suite of validation tests. This will take a while.

4 Creating and editing DSP networks

4.1 ObjectProDSP networks

A network consists of one or more data stream sources, DSP processes that transform input data streams to output data streams and data stream sinks. The simplest complete network is a signal node linked to a plot node as in the `basic` example. You can build DSP networks interactively, execute them and display the output. You can construct a stand alone version of a network in C`++` that you can execute or link with a target application. When a stand alone target is generated any plot nodes are replaced with nodes that write their output to disk.

Many DSP processes transform data in a uniform linear way. Whenever possible this information is used to compute sample rates and times for data streams. It is also used to construct deterministic scheduling schemes for the stand alone DSP networks.

The `network` menu under `objects` describes several structures related to network control. `CircBufDes` is the buffer descriptor class used to control buffer size and related parameters. `DataFlow` is the controller for a network. Most controller functions are also directly accessible through the `Network` object. `Network` is the class for defining a network. `TargetSytem` is a class for building a target system for stand alone execution using multiple networks that you can create interactively.

4.2 Graphically editing a network

You can create and edit a network using the keyboard or mouse. The `Edit` button in the upper left hand corner of the network window must be depressed to enable editing. To select editing press and release any mouse button with the cursor in the `Edit` button. Only one network can be edited at a time.

To create a new network disable editing on any existing network and create an instance of a node. When a network window is first displayed, editing will be enabled unless there is existing network with editing enabled. Instances of nodes can be created and added to a network from the menu database. Select `menu database` under the `Help` pull down menu for a description. Or try the example described in the `introduction` help window.

Once a node is in the network window you can connect it to other nodes in the same window. Nodes can be connected either to an output channel or a buffer for an output channel. There are fixed number of unique output channels (most commonly one) for each node. You can have many taps from a single output channel. Use the middle mouse button to select the next unused output channel. Use the right mouse button to add a tap to the buffer in the current output channel. (For all network editing operations with the mouse you must hold the shift key down.)

The buffers are represented by a small box with a number in it if there is more than one connection for the buffer. If you want to tap into a node for a channel that is already connected press the right mouse button in the box representing the buffer you want to connect to. If there is more than one input or output channel for a node the channel number will appear in front of (output channel) or after (input) channel the buffer for the connection.

A single node can be replaced with a different compatible node. To replace a node hold both the control and shift keys down and press and release the right mouse button on the node to be replaced. Then do the same thing to select the replacement node.

You can remove a node from a network. This will force any node in the network that is only connected through that node to also be removed from the network. Most removed nodes will be available for reuse but some will be deleted because they cannot be reconnected. You can also clear all nodes in a network with the `Network` and `DataFlow` member function `ClearNetwork`.

An existing node can be added to a network with member function `edit`.

4.3 Network display

The `Select` and `Link` menus allow you to select a node for, connect or disconnect it to other nodes, display nodes connected to it and perform related operations.

4.3.1 Select menu

As you add nodes to a network the window will grow to fixed limits and a horizontal scroll bar will then appear. If you want to change these limits, resize the window using the windows manger commands and then select `redraw to existing size` from this menu. The dimensions of you set will now be upper limits on how large the window can grow.

Ordinarily editing is selected by the check box in the upper left corner of the network window. It can also be enabled or disabled from this menu.

The remaining commands select a node by highlighting its name. You can initially select the upper left `HOME` or lower right `END` node. You can move the selection with the arrow keys. Display windows associated with a node (such as plots and listings) are displayed above overlapping windows with the `raise selected outputs` option.

A selected node is either one that is highlighted or one that has the mouse cursor positioned over it. Highlighted nodes are chosen first for commands entered by the keyboard. Nodes selected by the mouse are chosen first for mouse commands.

4.3.2 Link menu

The link menu allows you to highlight the connections to a node and perform editing operations on the node. You can show the connections to a node by selecting it and choosing to highlight the input connections, the output connections or all connections. Use the left, middle and right mouse buttons respectively for this. You can do this for a node drawn with a box and for the labels of a node that are not in a box.

4.4 DSP processing nodes

DSP processing nodes perform DSP operations on the data in their input channels and write the results to their output channels. There can be any number of input or output channels. For example there are nodes that do multiplexing (combine multiple channels into a single channel) and demultiplexing (the inverse). The number of input channels is displayed as an integer on the left of each node in a network. (If no integer is displayed there is a single input channel.) Similarly the number of output channels is displayed on the right. Each output channel of a node can have multiple buffers. The number of buffers is displayed in a small box to the right of the node. Each buffer is a copy of the same data stream. In contrast the different channels in a node ordinarily contain different data. For example in a node for demultiplexing each output channel will represent one of the demultiplexed data streams.

To access the DSP processing nodes select `objects` and `dsp processing`.

4.5 Plots

Plots in ObjectProDSP are windows on data streams. You can move the window though the data with the `page-up` and `page-down` commands and the scroll bar at the bottom of each plot.

Real data may be plotted as a linear time series. Complex data may be plotted as a linear time series or as an `eye` or X-Y plot, i. e. where the real value determines the X coordinate and the imaginary value determines the y coordinate. The linear plot node configures itself based on the type of input data it receives. If its input is complex it will plot the real and imaginary components in different colors.

Data streams have an element size parameter that determines the number of scalar values in each sample. This includes but is not limited to 1 for real and 2 for complex data. All element size components will be displayed on the same plot using with different colors for each component.

To access the plot nodes select `objects` and `plot`.

4.5.1 Plotting FFT (blocked) output

If the input to a plot node is from an FFT one frame from the FFT will be displayed in each plot. The X axis will be labeled in frequency based on the sample rate of the data stream and the FFT parameters. (A default sample rate of 1 hz for the first input channel is assumed if no rate is set.)

A block size parameter associated with each data stream determines the number of samples displayed in a plot. If the block size is 1 then the a strictly linear time series is assumed. The block size parameter is generated by the FFT node and read by the plot node.

4.5.2 Output after a network has been edited

If you execute a network, edit it and then execute it for additional iterations there may be discontinuities in the output at the point of the edit. If your editing adds new plotting nodes these will have a different time base then the plot nodes that were present before the edit. When you edit a network all the buffers are cleared and the nodes are reset. If you want output from an edited network with a consistent time base write the state out and then read it back in with options in the `state` menu. You must use the `read over state` option if you want to replace a network that has already been defined. Alternatively you can exit ObjectProDSP and then reexecute it. The state will be saved and read back in when you restart (unless you use the `-c` option or not using the default state file name).

4.5.3 Invalid numeric values

If the numbers sent to the plotting routine in the user interface process are not valid IEEE floating point values they will be converted to 0. A warning is displayed the first times this happen. A count of these is kept and can be displayed from `state` and `plot err` in the menu data base. This should not happen unless you a debugging a node that generates such values. There is only limited protection against this. Such values can cause the system to crash with a floating point exception.

4.5.4 Saving and reading plots

A plot can be saved as a file and a plot window image from the `View` pull down menu. This image window is saved with the X-windows utility `xwd`. It can be translated to a postscript file with the utility `xpr`. The postscript file can be included in printed documents.

A plot saved in a file can be read by ObjectProDSP and includes all the data in the original plot, not just the data displayed in the window. To read a plot file saved in this way select `setup` and `read plot` in the menu data base.

4.6 Plot detail

There are two ways to change the detail displayed in a plot. You can change the size of a window and you can show more or less of the data in the window. Use the standard X-windows controls to change the window size.

The pull down menus at the top of each plot window can be used to change the scale.

To change the data displayed in a plot with short cut use the arrow keys to indicate more or less detail in the either or both dimensions. You then select a region of the existing plot. If you are showing more detail this region will fill the plot window. If you are showing less detail the data displayed in the plot window will be compressed to fit in the region you select.

To select a region press the mouse button at one corner of the plot and drag the cursor to the other corner. When you let up on the mouse you will have selected a region and the window will be redrawn with the selected data. To return to the original scale type `home`. You can alternate between two scales by typing `shift-home` and `home`.

4.7 Listing plot coordinates

If you click the left mouse button at any location in a plot the coordinates of that position will be scrolled to the DSP++ history area in the main window. If you click the right mouse button close enough to a plotted point the coordinates of that point will be displayed and a box will appear around the point.

4.8 Views of plot data

The default view of all plots is linear. You can also view plots (except eye plots) on a power scale, a decibel scale or a power-decibel scale. The pull down menus can be used to select different views and they document the short cut keys to create views.

Data in a long stream for a new view is transformed up to the last point you display. Thus creating a new view and going to the last frame of that view may take some time.

You can also create one or more copy views of any plot. This can be useful if you want to simultaneously compare different frames of the same data stream.

You can save a plot window in a format that can be converted to postscript for inclusion in printed matter. You can save the entire plot data stream to a file that can be read later with the `read plot` option under `setup`.

4.9 Listing output from a DSP process

The `list` menu under `objects` gives access to nodes that display ascii formatted data.

The `HexList` and `Listing` nodes write data to text windows. As the process executes the data available for display is continually updated. You can scroll through this data at any time, even while execution is proceeding. You can make the windows larger or smaller. If you increase their height more lines will be displayed at once. If you decrease their width some part of each line may not be displayed.

The `HexList` node displays one or more channels in hexadecimal format and hard limits (with a warning) any input that will not fit in a 32 bit word. `HexList` displays an index of the total words output. If you do not want to display several channels in a single listing you may prefer to use the `Hex` option in the more flexible `Listing` node.

`Listing` lists its input in decimal or hexadecimal format. Under hexadecimal format values that would overflow 32 bits are hard limited with a warning. `Listing` groups elements from each sample in parenthesis. It gives a sample index for each line. If the data is blocked (for example FFT output) it gives a block index and an index within the block. Only the two least significant digits of these indices are displayed. Periodically there is a line that gives the complete index. by samples and blocks.

4.10 Moving through and saving listing output

A listing display is a window on a data stream. You can move the window through the data stream using the scroll bar or the keyboard with the shortcut keys described in the the pull down menu. You can save the entire data stream to an ascii file. Because this file contains sample indices it may not be suitable for exporting data to other processes. The `AsciiFile` node (under `objects`, `disk` and `ascii`) is designed for that purpose.

5 Menu database overview

5.1 Network window connection to menu database

Using the menu data base you can add nodes to a network change variables in a node and control a network. If you create an instance of a node from the menu database it will be added to the network being edited. If no network is being edited a new network will be created. Select `objects` in the main menu and a class of objects like `dsp processing`. Then hold the shift key down and push and release the left mouse button on a an object like `Add`. An instance of that object with default parameters will be added to the network. If you use the right mouse button you will be prompted for each of the parameters.

You can get a database menu for the network, buffer descriptor or controller from the `Database` pull down menu. You can can also get a menu for a selected node (or the class of that node). The menu for a node and the class of a node provide access to the documentation for the node including the values of parameters.

5.2 The `objects` menu

The `objects` menu describes the nodes or objects you can create interactively and all instances of those objects. The nodes are grouped under submenus. `dsp processing` selects nodes that read input channels and write output channels, `signal` select nodes that create output data streams with no input data streams and without reading disk files. `plot` selects the plotting nodes. `list` selects nodes that display numeric values. `disk` selects nodes that read or write disk files.

The `network` menu describes objects related to network definition and control. These include the network itself `Network`, the buffers assigned to a network `CircBufDes` and the controller for a network `DataFlow`.

Finally the `variables` menu lists defined variables and arrays.

5.3 Saving the state

State refers to the state of any DSP networks you are editing. The state is saved when you select `save and exit`. You can have the state saved periodically and you can save it at any point from the `state` menu. Multiple backup copies of the state are maintained. See the state menu for details.

You can also create action files as a record of almost everything you do with the mouse or keyboard. State files only preserve the final state of networks you create. Action files preserve the history of most actions in a session. Because of the way action files are tied to the detailed structure of menus they may not be portable across versions of ObjectProDSP. You should use state files and not action files to back up your work. They are also not completely reliable because of timing considerations. They are intended mainly to support regression tests of the user interface and to allow you to easily customize the initial windows.

Select `state`, `session` and `rec` to record your actions. You can only correctly play back an action file if ObjectProDSP is in the same state it was when the action file was created. Thus you should ordinarily start recording at the beginning of a session. Whenever you record or play back actions the session menu is selected to make sure the menu data base buttons are in the correct state. Action files were designed for regression testing but they can also be used for automated demonstrations and as an additional level of backup.

5.4 Menu data base commands

The menu data base is a central point for controlling ObjectProDSP. It is a hierarchical menu tree with integrated documentation.

The top level `help` menu provides online manuals, example DSP networks and control over the amount of help information displayed.

Most of the tree is under the `objects` menu. This contains all the classes of objects you can manipulate interactively and all instances of those objects. For example a DSP processing network is an object as is each of the nodes in the network. All of the manipulations of a network are through member functions. Many of these functions are called when graphically editing a network and most easily used in that way. However they are all implemented as member functions and documented as such in the menu data base.

The state menu allows you to save the DSP network state and to control how often the state is saved automatically and how many copies of the state are preserved. It allows you to `abort` the DSP process and to exist without saving the state. It includes a `session` menu to record and play back sessions.

The `objects` menus provides access to the objects you manipulate to create DSP networks. These are organized as categories of objects such as DSP processing and signal generators. Under each category there are the classes of objects that implement a function such as FFT. These are implemented and documented as C++ classes. (It is not necessary to understand C++ to use ObjectProDSP. You can do everything by graphically editing networks and using menus.) Under each class the instances of objects of that class are available. You use the classes to create instances of the objects and then you use member functions of the objects to manipulate them.

5.5 The `setup` menu

The `setup` menu allows you to read and ObjectProDSP state file. If you choose `read state` and the file defines a variable or object that already exists an error message will be displayed and the file read will stop. If you choose `read over state` then previously defined objects referenced in the file will be overwritten.

From `setup` you can read an ObjectProDSP plot file and display the contents in a window. Finally you can access a `debug` menu for options that are helpful if you are debugging your own processing nodes.

5.6 Debugging menu

The `debug` menu controls options that are helpful in debugging your own processing nodes. Trace outputs the name of each node just before it is executed to the log file `dsp.messages`. Trace is helpful if your node crashes the program. It allows you to trace which node is causing the problem. `heap ck` does a simple heap consistency check just before and just after each node is executing by allocating and freeing many memory segments. This frequently catches problems related to an inconsistent heap and makes it easier to trace the source of the problem. This option can significantly increase execution time.

Finally you can control how often some overflows are reported with the `over lin` option. There are two different mechanisms for reporting overflow. The more complex one is controlled by this option and only works if the node makes special provisions for it. Most conversions check for overflow and issue a warning if an overflow occurs. The total number of warnings reported for each network execution is limited to four. All warnings will be written to `dsp.messages` if trace is enabled.

5.7 Examples

The `examples` menu under `help` contains several examples. You can get a brief description of each example, view the DSP++ code for it and execute it. If you use the `execute` option and you have used any of the variables in the example you will get an error messages. With `execute over` previously defined variables with the same names will be overwritten.

The DSP++ code for each example (generated by save state) is in directory `$OPD_ROOT/examp`. By adding a file with suffix `.xml` to this directory you can add more examples to this menu. You need to add a help file with the same base name and suffix `.hlp` to directory `$OPD_ROOT/help` for each example you add.

5.8 Variables

You can interactively define scalar variables and arrays by entering declarations and assignment statements in C syntax in the DSP++ statement entry area in the main window. These variables can be used in arithmetic expressions entered in this area and as parameter values that you are interactively prompted for. You cannot enter expression when prompted for a parameter value. You must either enter a numerical value or the name of a variable.

All variables you defined and their values can be listed from the `variables` menu under `objects`. This menu is organized by the variable type. `MachWord` is the type of the simulator you are using. In the standard simulators it is a 16 bit integer a and single precision floating point value. `AccMachWord` is a double precision version of the same value that is typically used to represent the increased accuracy of a processor's accumulator.

5.9 Recording and playing back sessions

You can record and play back complete sessions. Select `state` and `session` from the menu data base or use the `Actions` pull down menu. You cannot record or play back X-windows operations such as resizing a window. You also cannot record scrolling through windows or changing the scaling on plots. However every menu data base command, or pull down menu command no matter how it is entered can be recorded.

Recording is not completely reliable because of timing considerations. In particular if you enter a command that prompts for inputs and those prompts originate from the DSP process you must not do anything else until the prompt appears. Action files are mainly useful for regression test validation and for customizing the initial appearance of ObjectProDSP. They are not a reliable way to back up your work. Use state files for that purpose.

There are examples of recorded sessions in the directory `$OPD_ROOT/validate`.

6 Member functions

6.1 Node member functions

Each node can have member functions specific to that node. In addition there are generic member functions served by all nodes, all signal nodes and all display nodes.

`Raise` raises a network window containing the node, i. e. makes the window visible over any overlapping windows. It also raises a plot or listing generated by the node.

`SetSampleRate` sets the output sample rate of a node and adjust the sample rate for all nodes in the network.

`DisplayInputTiming` and `DisplayOutputTiming` displays the timing for the input and output channels of a node.

`Edit` adds the node to a network being edited or creates a new network if none is being edited.

Other member functions are related to editing a network and are not normally used directly.

6.2 Make stand alone target executable

ObectPro++ translates its source into two `C++` programs. One is suitable for interactive use and the other for stand alone applications. You can automatically generate a stand alone executable of any network you create interactively with the `MakeTarget` member function of a network.

This option generates `C++` code and a `Makefile` to create a stand alone executable for the network using the target `C++` code for each of the nodes in the network. When a target is created all plotting nodes are replaced with `OutputNode` objects. The data that was plotted in the interactive version of the network will be written to a disk file. This data can be read with `InputNode` and plotted or processed in other ways.

6.3 Buffer descriptors

Buffer descriptors determine how data is buffered between nodes. Currently only circular buffers are supported. Usually you do not need to worry about these. They are created and assigned to a network automatically and the default values are generally sufficient. You need to use these objects to change the default buffer size or to modify the control and buffering in the stand alone version of a network. In interactive execution all buffers are of a fixed size and this must be large enough to prevent blocking. In the stand alone network buffers are made as small as possible consistent with other constraints.

There is a potential trade off between buffer size and efficiency. You can specify a desired size for the buffer and whether the buffer should be enlarged (increased performance) or shrunk (minimal memory requirements) if the desired size will waste memory.

There are two scheduling algorithms available. With fixed scheduling a static table determines the number of executions for each node. With some networks fixed scheduling is impossible or would produce an excessively long table. You can select if you want to attempt to used fixed scheduling. If it is not possible to used fixed scheduling then dynamic scheduling will be used. Dynamic scheduling is slightly less efficient because it requires computing the number of times a node can execute every time it is invoked. However this is done with an efficient algorithm that does not require a divide.

To assign a new buffer to an existing network select `network menu` from the pull down menu for `Data base` in the network window. Then select `exec` and `SetBufferDescriptor` (you may need to select `other` several times to get to `SetBufferDescriptor`) in the menu window for the network. You can also assign a buffer descriptor to an existing network be marking that network as being edited and creating a `CircBufDesc` object instance.

7 The DSP++ language

Development of Digital Signal Processing (DSP) applications is a labor intensive time consuming process. Each implementation is a custom product that makes little use of previous work and has little that can be reused in future projects. This is not unlike the state of programming in the early fifties when all programs were custom products written in assembly language.

Things are changing slowly. C compilers are becoming available for many DSP processors. While these generally do not provide the efficiency required for the kernel DSP algorithms they do allow prototyping in C and limiting assembly code to the kernel routines.

It is generally the case that any DSP system implementation is preceded by the construction of high level model. This is required because of the complexity of DSP algorithms and the many issues and tradeoffs involved in their implementation. Thus every system is implemented twice. The first is a functional implementation in a development environment that supports analysis and debugging of the algorithms. The second is a custom assembly language implementation on DSP processors.

DSP`++` is a language and a methodology to dramatically improve the efficiency of DSP system implementation. It uses the object oriented methodology of C`++` and the universality of C to provide a practical approach to the problems of DSP system implementation.

The DSP`++` language itself is a restricted subset of C`++`. Classes for signal processing objects form the core of the DSP`++` language. These include signal processing nodes, signal generation and input nodes and signal output or display nodes. There are additional objects for defining DSP network, controlling DSP networks and controlling buffer allocation.

7.1 ObjectProDSP language overview

DSP`++` is a set of C`++` classes for defining and manipulating DSP networks. The operators for defining a network are `+` and >>. The `+` operator adds thread to an existing network. For example the statement `Net + Signal` adds a signal node `Signal` to a network `Net`. To connect a node to a node already linked in a network enter `Net >> Node_A`. These two statements can be written as a single statement: `Net + Signal >> Node_A`.

Technically the operators `+` and >> are member functions of the DSP++ class `Network`. You write statements using these member functions as shown above but they are defined as functions `operator+(SignalStr& signal_node)` and `operator>>(Node& node)`. All the operations in DSP++ are implemented as member functions and documented as such in the menu data base. C++ member functions have been extended in ObjectPro++ to create interactive member functions. If you interactively execute a member function you be prompted for the parameter values. Creating instances of an object is a special case of the member function. The function to create an instance is called a constructor. When you create an object interactively that object is added to the menu data base.

7.2 The DSP++ interactive interpreter

The interpreter works similarly to the language SmallTalk. Instead of a C main program you defines and manipulates objects. The objects are DSP processing nodes that are built into networks. The networks are then executed and the output displayed. The simulation results may be graphical or numeric. They can be viewed interactively and saved to a disk file.

Although you can enter statements directly it is easier and quicker to have most statements written for you. As you graphically editing a DSP network DSP++ statements are generated to implement the actions you request. You can create nodes by a single mouse click (when the default parameters are adequate) and connect nodes in the network with two mouse clicks.

7.3 DSP networks

DSP++ allows you to define an arbitrary network of DSP processing nodes. This network can include feedback loops and resampling. Input can be from either signal generation nodes or disk files. Output can be to plot windows, text windows, disk files or any combination of these. The same data output can be sent to multiple destinations.

Analysis routines determine the completeness and consistency of the network and insure that all feedback loops have enough delay to prevent deadlock. A timing analysis routine determines the timing relationship of all data streams. The data is displayed with timing labels that reflect the effect of DSP processing. For example, the correct time of the output sample of a symmetric FIR filter is the time of the midpoint of the filter impulse response. DSP`++` understands and accounts for such timing relationships.

Networks can be controlled interactively. You may specify how many input samples are created and display the output generated. He may then vary some of the parameters and execute the network again, to see the effect of these changes. Networks can be edited interactively. Display nodes may be added at any point in the network.

7.3.1 C++ language extensions for networks

The operators `+` and `>>` used in defining DSP networks is an example of an internal C`++` language extension. One specifies the addition of a new thread to a network as `Net + Signal` where Net is a network and Signal is signal source node. To connect a new node to a node already in a network one writes `Net >> Node_A`. One can write the whole thing as a single statement: `Net + Signal >> Node_A`.

The language extension capabilities of C`++` allow the definition of a high level DSP language while maintaining conformance to the industry standard programming language C`++`.

7.3.2 Modeling DSP processor arithmetic

The arithmetic in the ObjectProDSP processing nodes is done on two defined classes: `MachWord` and `AccMachWord`. These classes correspond to the basic word size (`MachWord`) and the accumulator word size (`AccMachWord`) of target systems. Doing this allows us take our existing library of DSP processing nodes and model the arithmetic of virtually any target DSP processor. There is no need to modify the processing node code. We merely need to define the arithmetic operators in these two classes to correspond to the arithmetic in the target DSP processor.

Currently ObjectProDSP has two arithmetic models. The first uses all single precision floating point arithmetic. The second model assumes a 16 bit word size and a 32 bit accumulator. It is possible to do both 32 bit integer and 32 bit floating point operations in the floating point simulator.

7.4 A language for DSP?

The world is not in need of more programming languages. It can benefit from extending languages to more effectively deal with applications. C++ and object oriented languages in general recognize this. They are designed to be the bottom level of a hierarchy of object libraries for different application areas.

The traditional method of extending languages is via subroutine libraries. This method is effective and useful. It works well for low level routines such as those in the standard library provided with most C compilers. It is also useful for providing high level functions that are used frequently such as Digital Signal Processing kernel operations.

Subroutine libraries have limitations:

- Subroutine calls with more than a few parameters are awkward.
- It is difficult to construct libraries that are both general and easy to use.
- Libraries do not support use of natural mathematical notation such as arithmetic operators in matrix and vector expressions.

Good engineering practice aims at building complex systems in a modular layered way. No tools will insure that a project is implemented in this manner. Good tools can make it easier and less labor intensive to do so. Subroutine libraries is an important but limited step in this direction.

DSP++ will support the creation of DSP programs that can be ported to a variety of architectures. The idea is to allow the expression of a DSP algorithm at as high a level of abstraction as possible. The algorithm is designed, developed, debugged and verified at that level. It can then be directly ported to any target architecture that supports a standard C compiler.

ObjectProDSP uses both C++ facilities such as class hierarchies and a language preprocessor to allow a single program to be suitable for both interactive execution and use in a dedicated DSP processor.

Use of DSP++ provides critical advantages to the creator of DSP products.

1. Time to market will improve significantly: The algorithm will be expressed and validated at a high level and then rapidly ported to the target architecture.
2. Porting of existing products to new or more cost effective hardware architectures will be a relatively easy task.
3. The same application can be supported on a variety of architectures.
4. The overall cost of software development and maintenance will be reduced.

7.5 Data flow model

Virtually all computer processes fit a data flow model at least to the extent of having input and output. In the absence of output, there is no point to the program. In the absence of input, the program only needs to be run once. DSP processes are frequently, but not always, blessed with regular and predictable data flow. We need to fully exploit this predictability without limiting DSP++ to only support regular data flow.

7.5.1 Data input/output model

A general mechanism for modeling such data flow and using the model to schedule process execution is described in [7]. We incorporate some of these ides in our data flow model. We support processes that have one or more input and one or more output nodes and consume and emit data as follows. Input channel i for node j needs to have available in its input buffer $B_i^j + XC_i^j$ samples to output on channel k XD_k^j output samples. X is a parameter that can be set to any integer value.

There are many examples that fit this model. A FIR filter of length L with output resampling of r must have $L - r + rk$ inputs to generate k outputs. An FFT of size F with an overlap factor of v requires $F - vF + vFk$ inputs to generate Fk outputs. Many DSP processes have a fixed relationship between input and output samples that fit this model.

7.5.2 Processing nodes

Node definitions must satisfy generic and process specific requirements. For example a FIR filter will contain a pointer to the filter coefficients and the filter length. It will also contain generic data flow model parameters that relate input data consumption to output data generation. To support both generic and specific requirements we use C++ derived classes. We define a generic base class for all processing nodes that fit the data flow model and then construct derived classes for specific functions such as FIR filters.

8 ObjectProDSP example networks

There are several examples to illustrate the ObjectProDSP language and tool. It is easy to add custom examples to your installation and have these available in the menus. This section describes the examples menu and the basic examples. Your installation may have additional examples.

8.1 Examples menu

The `Help` selection in the main menu and then the `examples` option lead to the examples menu. From this menu you can select any of the examples described in the following sections. You can then select a description of the example, execute it or read its DSP`++` code.

8.2 Basic example

The basic example generates and displays 1024 samples from a complex sine wave. It shows how to declare a signal generator, a plot and a network. It shows how to define a simple network topology and execute the network.

This is the ObjectProDSP program for this example.

```
// This basic example displays a complex sine wave

// Declare the signal generator:
CxCos BasicSig ;

// Declare the plot:
Plot BasicPlt ;

// Declare the network:
Network Basic ;

// Define the network topology:
```

```
Basic + BasicSig >> BasicPlt ;

// Display the network in graphical form
Basic.GraphDisplay();

// Execute the network generating 1024 samples:
Basic.Execute(1024);
```

8.3 FFT example

This example adds a complex sine wave to Gaussian noise. This sum is processed by an FFT. The FFT output and both of the signal sources are plotted. This example shows how to:

1. Sum two data streams to generate a single output.
2. Link a display node into an existing network.
3. Use the Gaussian noise generator.
4. Display multiple plots.
5. Use he FFT.

This is the ObjectProDSP program for this example.

```
// Generate a complex .08 hz sine signal
// assuming a 1 hz sample rate
// The initial phase is 0 and the amplitude is 32.
CxCos FFT_Sig( 2.*Pi*.08,0,32);

// Define a 512 (2^9) point fft node
CxFFT FFT_FFT(9);

// Define 3 plot nodes for the output, noise and signal
Plot FFT_Plt;
Plot FFT_PltNoise ;
```

```
Plot FFT_PltSignal ;

// Define a gaussian random number generator with
// standard deviation of 32, and mean of 0.
// Generate pairs of sample for complex data.
Normal FFT_Noise(32.,0,2);

// Define a summation node for 2 complex channels.
// The overall gain of this node is 1.
Add FFT_Sum(2,2,1.);

// Define a network
Network FFT_Net;

// Define the network topology
FFT_Net + FFT_Sig >> FFT_Sum >> FFT_FFT >> FFT_Plt ;
FFT_Net + FFT_Noise >> FFT_Sum ;
FFT_Net.Link(FFT_Noise)>>FFT_PltNoise ;
FFT_Net.Link(FFT_Sig)>>FFT_PltSignal ;

// Display the network in graphical form
FFT_Net.GraphDisplay();

// Execute the network generating 4096 samples
FFT_Net.Execute(4096);
```

8.4 Three stage fir filter example

This is the ObjectProDSP program for this example.

```
CxCos Cos02hz( 2 * Pi * .02, 0.0, 1.024e+03);
CxCos CosPt2hz( 2 * Pi * .2, 0.0, 1.024e+03);
Network Fir_net;
CxFFT FFT_1024(10, 0.0, 5.e-01, 0);
```

```
CxFFT FFT_256(8, 0.0, 5.e-01, 0);
static MachWord Stage_one[] = {
  -1.8691079691052437e-02,  -3.1646952033042908e-02,
   1.3136906921863556e-01,   4.188159704208374e-01
};

CxFir Fir_one(2, 0, 0.0, 0, Stage_one);
static MachWord Stage_two[] = {
   5.0137522630393505e-03,   8.9365877211093903e-03,
  -3.1649746000766754e-02,  -5.4023709148168564e-02,
   1.3631737232208252e-01,   4.3563303351402283e-01
};

CxFir Fir_two(2, 0, 0.0, 0, Stage_two);
static MachWord Stage_three[] = {
   8.3521055057644844e-04,   2.0192209631204605e-03,
   8.7504170369356871e-04,  -1.5354243805631995e-03,
  -1.9233264029026031e-03,   1.4449445297941566e-03,
   3.3509572967886925e-03,  -4.9146544188261032e-04,
  -4.9332436174154282e-03,  -1.4811638975515962e-03,
   6.0873683542013168e-03,   4.6224407851696014e-03,
  -6.2061776407063007e-03,  -8.7587479501962662e-03,
   4.5680347830057144e-03,   1.3436957262456417e-02,
  -4.8366468399763107e-04,  -1.7854765057563782e-02,
  -6.6574704833328724e-03,   2.0863097161054611e-02,
   1.7380831763148308e-02,  -2.0901164039969444e-02,
  -3.2361775636672974e-02,   1.565072312951088e-02,
   5.3351804614067078e-02,  -3.5667233169078827e-04,
  -8.7312132120132446e-02,  -4.2652428150177002e-02,
   1.8306387960910797e-01,   4.0873810648918152e-01
};

CxFir Fir_three(2, 0, 0.0, 0, Stage_three);
Normal Fir_noise( "Fir_noise", 1.024e+03, 0.0, 2);
Add Fir_sum( "Fir_sum", 3, 2, 1.);
Plot Plot_cos02hz;
Plot Plot_cosPt2hz;
```

```
Plot Plot_fft_1024;
Plot Plot_fir_sum;
Plot Plot_fft_256;
CircBufDes Fir_net_buf(4096) ;


// Network topology for 'Fir_net'
Fir_net + Fir_noise >> Fir_sum >>Fir_one >> Fir_two >> Fir_three >>
FFT_256 >> Plot_fft_256 ;
Fir_net.Link(Fir_sum,0) >> FFT_1024 >>  Plot_fft_1024 ;
Fir_net.Link(Fir_sum,0) >> Plot_fir_sum ;
Fir_net + Cos02hz >> Fir_sum.LinkIn(1) ;
Fir_net.Link(Cos02hz,0) >> Plot_cos02hz ;
Fir_net + CosPt2hz >> Fir_sum.LinkIn(2) ;
Fir_net.Link(CosPt2hz,0) >> Plot_cosPt2hz ;

Fir_net.AssignBuffers(Fir_net_buf);
Fir_net.GraphDisplay();
Fir_net.Execute(4096);
```

8.5 Feedback example

This example shows a feedback loop that produces overflows. It includes a four sample delay so the feedback loop will not deadlock.

This is the ObjectProDSP program for this example.

```
// Declare signal and output nodes
CxCos SigG(1.2566370614399999e-01, 0.0, 1.024e+03);
Listing FeedbackListing;

// Declare the DSP nodes
SampleDelay Delay_4(4);
Add CxAddNode_2(2, 2, 1.);
Gain Gain4X(4.);
```

```
// Declare the network
Network Feedback ;

// Network topology for ‘Feedback’
Feedback + SigG >> CxAddNode_2 >> Gain4X >> FeedbackListing;
Feedback.Link(Gain4X,0) >> Delay_4 >> CxAddNode_2.LinkIn(1) ;

// Display the network in graphical form
Feedback.GraphDisplay();

// Execute the network
Feedback.Execute(256);
```

8.6 Fir filter response example

This example creates a graph of the frequency response of a fir filter operating on a complex data stream. The plot is generate by passing an impulse through the filter and taking a complex FFT of this output. The resulting complex plot shows the amplitude and phase response of the filter. Position the cursor in this plot and type ∧`B` to see a decibel power plot or ∧`P` to see a power plot of the same data.

This is the ObjectProDSP program for this example.

```
MachWord Fir_resp_cxfir_data[] = {
   1.0001855116570368e-04,   3.7074790452606976e-04,
   4.4659440754912794e-04,  -5.3696951363235712e-04,
  -2.6749277021735907e-03,  -3.5268759820610285e-03,
   8.2571519305929542e-04,   1.0149464011192322e-02,
   1.508740521967411e-02,    2.4800843093544245e-03,
  -2.7687691152095795e-02,  -4.934985563158989e-02,
  -2.2059476003050804e-02,   7.307908684015274e-02,
   2.0401032269001007e-01,   2.992863655090332e-01 } ;
Network Fir_resp_net ;
CxFFT Fir_resp_fft(7, 0.0, .5, 0);
```

```
CxFir Fir_resp_cxfir(1, 0, 0.0, 0, Fir_resp_cxfir_data);
CxImp Fir_resp_cximp(128, 0.0, 1000, 0.0, 0);
Plot Fir_resp_fft_plot ;
Plot Fir_resp_cxfir_plot;

// Network topology for 'Fir_resp_net'
Fir_resp_net + Fir_resp_cximp >> Fir_resp_cxfir >>
Fir_resp_fft >> Fir_resp_fft_plot ;
Fir_resp_net.Link(Fir_resp_cxfir,0) >> Fir_resp_cxfir_plot ;

Fir_resp_net.GraphDisplay();

Fir_resp_net.Execute(2048);
```

8.7 Adding new examples

To add an example to the ObjectProDSP menus you create a help file and place it in directory `$PPD_ROOT/examp` and a DSP++ language file and add it to `$PPD_ROOT/help`. The example files must have suffix `.xml`. The corresponding help file must have the same base name and end in suffix `.hlp`. `.hlp` are created automatically from `.roff` files in `$PPD_ROOT/doc/roff`. See *ObjectProDSP Developer's Reference* for instructions for creating help files. The examples appear in the menus in alphabetic order with the root of the file name as the option to select a given example.

ObjectProDSP MENUS REFERENCE

9 Using the menus reference manual

The following sections list the ObjectProDSP menus. They are structured to match the hierarchy in the ObjectProDSP interactive menus with two exceptions. The Main menu described in the next section has all its child menus at the same top level as itself. Menus that fall under the menu of all ObjectProDSP classes in Section 12 on page 53 are also at the top level.

Every interactive class in ObjectProDSP has an identical hierarchy of menus to create, destroy and manipulate objects. This hierarchy is shown twice for nodes `Add` in Section 13.1 on page 56 and node `InputNode` in Section 17.2.2 on page 115. It is not shown for other nodes to avoid needless repetition of the same information. All other ObjectProDSP menus are fully described. For additional details about the parameters and documentation of any ObjectProDSP object see the *ObjectProDSP Library Reference*[3]. The information in this section is available on-line from within ObjectProDSP.

10 ObjectProDSP menu data base

This is the main ObjectProDSP menu. All other menus and commands are reachable from this menu.

The commands in this menu are:

- `help`: Main help menu. This menu option invokes the menu defined in Section 11 on page 50.
- `objects`: Display and describe existing objects. This menu option invokes the menu defined in Section 12 on page 53.
- `setup`: Read state and plot files, debugging. This menu option invokes the menu defined in Section 20 on page 148.
- `state`: Program state menu. This menu option invokes the menu defined in Section 21 on page 150.

- `save and exit`: Save program state and exit. This menu option is a command.

  `save and exit` exits to the operating system. If you have not saved the state of objects since they were last changed the state will be saved before exiting. Use the `state` menu if you wish to exit without saving. If the DSP process is hung you may need to do this twice to get information about its state.

11 Structured access to on-line documentation

The `help` menu contains information organized by topics. It covers the ObjectProDSP language and describes the use of this program. It has information for the new user and is an on-line reference manual. You can control the amount of automatic help information from this menu.

The commands in this menu are:

- `introduction`: Basic information for the new user. This menu option is a command.

  The `introduction` help screen provides a general description of the ObjectProDSP system. It describes on the purpose, capabilities and general structure of this tool.

- `copying`: THERE IS NO WARRANTY and information about copying. This menu option is a command.

  Copyright 1994 Mountain Math Software, all rights reserved. This software is licensed for free use and distribution under version 2 of the GNU General Public License. You can use and distribute this software free of charge provided you do so in accord with the provisions of this license. The `copying` help screen contains the text of this license, a warranty notice (THERE IS NO WARRANTY) and information about support and other licensing arrangements.

- `manual`: Online ObjectProDSP manual. This menu option invokes the menu defined in Section 11.1 on page 51.
- `examples`: Display and execute examples. See Section 8 on page 39 for a description of the examples menu and the basic set of examples. There may be additional examples available in your installation.
- `help levels`: Control the display of help information. This menu option invokes the menu defined in Section 11.2 on page 53.

11.1 Online manual

The `manual` menu is an online manual. It describes how to graphically create and edit DSP networks, the DSP++ language, and other capabilities of this tool.

The commands in this menu are:

- `about help`: Describe the help facilities. This menu option is a command.

  The `about help` help screen describes the level of help available, how to access help information and how to control the information displayed automatically.
- `DSP nets`: Graphically creating and editing networks. This menu option is a command.

  The `DSP nets` help screen describe how to graphically create and edit DSP networks. It describes how to execute the networks and create a stand alone program that implements the network you built.
- `DSP++`: ObjectProDSP language. This menu option invokes the menu defined in Section 11.1.1 on page 52.
- `commands`: Data base commands, record and playback a session. This menu option is a command.

  The `commands` help screen describes commands accessible through the menu data base and describes how to record and playback action files of commands and other user input.

- `release`: Release notes for this version. This menu option is a command.

  The `release` help screen displays the release notes for this version of ObjectProDSP. These include known limitations and describe things we would like to change and improve.

11.1.1 ObjectProDSP language

The `DSP++` menu describes describes the main elements in the DSP++ language. It also contains information on validation suites.

The commands in this menu are:

- `network help`: ObjectProDSP networks. This menu option is a command.

  Ths `network help` screen describes ObjectProDSP processing networks. Processing is done in a network of nodes or objects. The initial node or nodes are signal sources. The terminal node or nodes output to a window or disk file.

- `object help`: DSP processing objects. This menu option is a command.

  The `object help` screen describes the ObjectProDSP processing objects.

- `plot help`: Plotting objects. This menu option is a command.

  The `plot help` screen describes objects that plot their input.

- `syntax`: ObjectProDSP language syntax. This menu option is a command.

  The `syntax` help screen gives an introduction ObjectProDSP language structure. You do not need to know this language but you may find it helpful especially if you are already familiar with C++. Often operations that you enter graphically are translated to DSP++ language statements which are scrolled in the main window.

- `validation`: ObjectProDSP validation suites. This menu option is a command.

ObjectProDSP has facilities for automated validation.

11.2 Control information display

From the `help levels` menu you can change the information displayed automatically. This includes descriptions such as this one of menu entries and the confirmation messages from most actions.

The commands in this menu are:

- `help none`: Display no help information. This menu option is a command.

  `help none` disables display of help information except for error messages.

- `help confirm`: Display action confirmation messages only. This menu option is a command.

  `help confirm` displays confirmation messages for most user actions. It disables the automatic display of the menu item descriptions such as this is. These descriptions can still be selected manually using mouse button three.

- `help all`: Display all help messages. This menu option is a command.

  `help all` enables all help displays including confirmation of user actions and descriptions of each menu selection.

12 Object classes

The `objects` menu provides tree structured access to the definitions and descriptions of objects. It allows objects to be created and destroyed.

The commands in this menu are:

- `dsp processing`: DSP processing objects. This menu option invokes the menu defined in Section 13 on page 54.

- `signal`: Signal generation objects. This menu option invokes the menu defined in Section 14 on page 87.
- `plot`: Plotting objects. This menu option invokes the menu defined in Section 15 on page 99.
- `list`: Listing objects. This menu option invokes the menu defined in Section 16 on page 102.
- `disk`: Read and write disk files. This menu option invokes the menu defined in Section 17 on page 105.
- `network`: Network and system objects. This menu option invokes the menu defined in Section 18 on page 125.
- `variables`: Simple variables. This menu option invokes the menu defined in Section 19 on page 147.

13 DSP processing objects

These objects do signal processing such as filtering, demodulation and resampling. The `dsp processing` menu displays object classes and object instances including those you have defined interactively.

The commands in this menu are:

- `Add`: `Add` sums two or more input channels. This menu option invokes the menu defined in Section 13.1 on page 56.
- `Block`: Converts an input stream to a new blocking and sample size. This menu option invokes the menu defined in Section 13.2 on page 60.
- `CxFFT`: `CxFFT` computes the complex FFT of a single input channel. This menu option invokes the menu defined in Section 13.3 on page 61.
- `CxFir`: `CxFir` is a complex symmetric (even or odd) fir filter. This menu option invokes the menu defined in Section 13.4 on page 63.

- `Demod`: `DemodFreq` is a complex modulation/demodulation function. This menu option invokes the menu defined in Section 13.5 on page 64.
- `Demux`: Demultiplexes 1 input channel to `Channels` output channels. This menu option invokes the menu defined in Section 13.6 on page 66.
- `FindStartTail`: discard initial input data within bounds. This menu option invokes the menu defined in Section 13.7 on page 67.
- `Gain`: Gain provides a linear gain. This menu option invokes the menu defined in Section 13.8 on page 68.
- `GainPad`: `GainPad` provides a linear gain. This menu option invokes the menu defined in Section 13.9 on page 70.
- `Integrate`: `Integrate` sums consecutive input vector. This menu option invokes the menu defined in Section 13.10 on page 71.
- `Interpolate`: sample rate conversion with linear interpolation. This menu option invokes the menu defined in Section 13.11 on page 73.
- `MaskWord`: applies a mask to a binary data stream. This menu option invokes the menu defined in Section 13.12 on page 74.
- `Mux`: Multiplexes `Channels` inputs into 1 output channel. This menu option invokes the menu defined in Section 13.13 on page 75.
- `PackWord`: packs multiple input words to a single output word. This menu option invokes the menu defined in Section 13.14 on page 76.
- `Power`: Power computes and scales the power in each sample. This menu option invokes the menu defined in Section 13.15 on page 77.
- `RealFir`: `RealFir` is a real symmetric (even or odd) fir filter. This menu option invokes the menu defined in Section 13.16 on page 79.
- `RepackStream`: repack bit streams to different physical word sizes. This menu option invokes the menu defined in Section 13.17 on page 80.
- `SampleDelay`: delays the output by a selected number of samples. This menu option invokes the menu defined in Section 13.18 on page 81.

- `ToInteger`: converts MachWord data stream to integer. This menu option invokes the menu defined in Section 13.19 on page 82.
- `ToMach`: converts binary data stream to `MachWord`. This menu option invokes the menu defined in Section 13.20 on page 83.
- `Truncate`: Limit the dynamic range and significant bits in a stream. This menu option invokes the menu defined in Section 13.21 on page 84.
- `UnpackWord`: unpack a single input word to multiple output words. This menu option invokes the menu defined in Section 13.22 on page 86.

13.1 Options for `Add`

The commands in this menu are:

- `help`: Explain the use of `Add`. This menu option is a command.
- `param`: Parameters of `Add`. This menu option invokes the menu defined in Section 13.1.1 on page 57.
- `variables`: Changeable variables of `Add`. This menu option invokes the menu defined in Section 13.1.2 on page 57.
- `members`: Members of `Add`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Add`. This menu option invokes the menu defined in Section 13.1.3 on page 58.
- `create default`: Create instance of `Add` with default parameters. This menu option is a command.
- `create`: Create instance of `Add`. This menu option is a command.

13.1.1 Menu of Parameters for `Add` Object

The commands in this menu are:

- `Channels`: Parameter `Channels` of `Add` object. This menu option is a command.

  `Channels` specifies the number of input channels to be added together in a single output channel.

- `ElementSize`: Parameter `ElementSize` of `Add` object. This menu option is a command.

  `ElementSize` specifies the sample size for each channel. The most common use of `ElementSize` is to set it to two for a complex data stream. All input data streams must have the same sample size. If set to 0 `ElementSize` is set automatically based on its value for the first input node.

- `Scale`: Parameter `Scale` of `Add` object. This menu option is a command.

  `Scale` is a scale factor applied to each channel before it is added to the output channel. The channel addition will clip any data that would otherwise create an overflow and generate a help message. Only one help messages is generated, for every 400 input samples regardless of the number of overflows that may occur.

13.1.2 Menu of Variables for Add Object

The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `Add` object. This menu option is a command.

  `Scale` is a scale factor applied to each channel before it is added to the output channel. The channel addition will clip any data that would otherwise create an overflow and generate a help message. Only one help messages is generated, for every 400 input samples regardless of the number of overflows that may occur.

13.1.3 Select an Instance of `Add` to describe or delete

The commands in this menu are:

- *instance of this class*: Select this instance of `Add`. This menu option invokes the menu defined in Section 13.1.3.1 on page 58.

13.1.3.1 Describe or delete an instance of object `Add`

The commands in this menu are:

- `desc`: Describe this instance of `Add`. This menu option is a command.
- `param`: Describe parameters of this `Add`. This menu option invokes the menu defined in Section 13.1.3.1.1 on page 58.
- `exec`: Select a member of `Add` to execute. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `exec`. See , Section 22.22 on page 163 for additional base class member functions in this menu.
- `variables`: Describe variables of this `Add`. This menu option invokes the menu defined in Section 13.1.3.1.2 on page 59.
- `set`: Set variable values of this `Add`. This menu option invokes the menu defined in Section 13.1.3.1.3 on page 59.
- `delete`: Delete this `Add`. This menu option is a command.

13.1.3.1.1 Menu of Parameters for `Add` Object Instance

The commands in this menu are:

- `Channels`: Parameter `Channels` of `Add` object instance. This menu option is a command.

  `Channels` specifies the number of input channels to be added together in a single output channel.

- `ElementSize`: Parameter `ElementSize` of `Add` object instance. This menu option is a command.

  `ElementSize` specifies the sample size for each channel. The most common use of `ElementSize` is to set it to two for a complex data stream. All input data streams must have the same sample size. If set to 0 `ElementSize` is set automatically based on its value for the first input node.

- `Scale`: Parameter `Scale` of `Add` object instance. This menu option is a command.

  `Scale` is a scale factor applied to each channel before it is added to the output channel. The channel addition will clip any data that would otherwise create an overflow and generate a help message. Only one help messages is generated, for every 400 input samples regardless of the number of overflows that may occur.

13.1.3.1.2 Menu of Variables for Add Object Instance The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `Add` object instance. This menu option is a command.

  `Scale` is a scale factor applied to each channel before it is added to the output channel. The channel addition will clip any data that would otherwise create an overflow and generate a help message. Only one help messages is generated, for every 400 input samples regardless of the number of overflows that may occur.

13.1.3.1.3 Menu of Variables for Add Object Instance The commands in this menu are:

- `set Scale`: Change value of variable `Scale` for an instance of `Add`. This menu option is a command.

  `Scale` is a scale factor applied to each channel before it is added to the output channel. The channel addition will clip any data that would otherwise create an overflow and generate a help message. Only one help messages is generated, for every 400 input samples regardless of the number of overflows that may occur.

13.2 Options for `Block`

The commands in this menu are:

- `help`: Explain the use of `Block`. This menu option is a command.
- `param`: Parameters of `Block`. This menu option invokes the menu defined in Section 13.2.1 on page 60.
- `members`: Members of `Block`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Block`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Block` with default parameters. This menu option is a command.
- `create`: Create instance of `Block`. This menu option is a command.

13.2.1 Menu of Parameters for `Block` Object

The commands in this menu are:

- `ElementSize`: Parameter `ElementSize` of `Block` object. This menu option is a command.

  `ElementSize` is the number of words in each output sample (1 for real, 2 for complex or larger for other purposes).

- `BlockSize`: Parameter `BlockSize` of `Block` object. This menu option is a command.

  `BlockSize` is the number of samples in each output block. If set to 1 the output is not blocked.

- `OutputArithmetic`: Parameter `OutputArithmetic` of `Block` object. This menu option is a command.

  `Block` can read data from any input arithmetic type. `OutputArithmetic` selects the output arithmetic type. On a 32 bit simulator `Block` can write output as either 32 bit floating point or 32 bit fixed point. Choose 0 to write output in the default type of the simulator, 1 for 32 bit integers and 2 for 32 bit floating point.

13.3 Options for CxFFT

The commands in this menu are:

- `help`: Explain the use of `CxFFT`. This menu option is a command.
- `param`: Parameters of `CxFFT`. This menu option invokes the menu defined in Section 13.3.1 on page 62.
- `variables`: Changeable variables of `CxFFT`. This menu option invokes the menu defined in Section 13.3.2 on page 62.
- `members`: Members of `CxFFT`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `CxFFT`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `CxFFT` with default parameters. This menu option is a command.
- `create`: Create instance of `CxFFT`. This menu option is a command.

13.3.1 Menu of Parameters for `CxFFT` Object

The commands in this menu are:

- `LogSize`: Parameter `LogSize` of `CxFFT` object. This menu option is a command.

  `LogSize` is the log base 2 of the FFT size.

- `Overlap`: Parameter `Overlap` of `CxFFT` object. This menu option is a command.

  `Overlap` specifies the fractional overlap of successive FFTs. For example `Overlap` = .5 and `LogSize` = 64 would result in each FFT being 32 samples beyond the previous FFT. `Overlap` = 1.0 is interpreted to mean that successive FFTs having their inputs separated by a single sample.

- `CenterFrequency`: Parameter `CenterFrequency` of `CxFFT` object. This menu option is a command.

  The true center frequency of the FFT is determined by the data entering it. `CenterFrequency` allows you to rotate the output bins to conform to this. The default value of .5 corresponds to a signal with center frequency of 0 hz in the input sample stream. This value should set to the relative position of the center frequency of the input sample stream.

- `InverseFlag`: Parameter `InverseFlag` of `CxFFT` object. This menu option is a command.

  `InverseFlag` set to 1 selects an inverse FFT.

13.3.2 Menu of Variables for CxFFT Object

The commands in this menu are:

- `CenterFrequency`: Changeable variable `CenterFrequency` of `CxFFT` object. This menu option is a command.

  The true center frequency of the FFT is determined by the data entering it. `CenterFrequency` allows you to rotate the output bins to conform to this. The default value of .5 corresponds to a signal with center frequency of 0 hz in the input sample stream. This value should set to the relative position of the center frequency of the input sample stream.

13.4 Options for `CxFir`

The commands in this menu are:

- `help`: Explain the use of `CxFir`. This menu option is a command.
- `param`: Parameters of `CxFir`. This menu option invokes the menu defined in Section 13.4.1 on page 63.
- `members`: Members of `CxFir`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `CxFir`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `CxFir` with default parameters. This menu option is a command.
- `create`: Create instance of `CxFir`. This menu option is a command.

13.4.1 Menu of Parameters for `CxFir` Object

The commands in this menu are:

- `Resample`: Parameter `Resample` of `CxFir` object. This menu option is a command.

  `Resample` specifies the filter resampling factor. This is the ratio of the input sampling rate to the output sampling rate.

- `ZeroPad`: Parameter `ZeroPad` of `CxFir` object. This menu option is a command.

  `ZeroPad` specifies the number of 0's that are added after each input sample. If Zero padding is done the filter must be designed as an interpolation filter.

- `DemodFreq`: Parameter `DemodFreq` of `CxFir` object. This menu option is a command.

  `DemodFreq` is the demodulation frequency in radians per sample. Input sample N is multiplied by e∧(-i × 2 × Pi × `DemodFreq` × N) before the filtering operation. If `DemodFreq` is 0 then the demodulation step is skipped.

- `Odd`: Parameter `Odd` of `CxFir` object. This menu option is a command.

  `Odd` determines if the filter is odd (`Odd` =1) or even (`Odd` =0).

- `Coeff`: Parameter `Coeff` of `CxFir` object. This menu option is a command.

  `Coeff` is the list of filter coefficients. The filter is symmetric and only half (or half plus one for odd length filters) are specified. The first in the list is the first coefficient of the filter. The middle coefficient is at the end of the list. The default values for the coefficients define a low pass FIR filter with a pass band of .125 times the sample rate and transition band of .375 times the sample rate. The pass band is extremely flat and the stop band is down over 100 db. This is an overdesigned filter for most practical applications but it provides a good test case. The performance will be degraded by 16 bit integer arithmetic.

13.5 Options for `Demod`

The commands in this menu are:

- `help`: Explain the use of `Demod`. This menu option is a command.
- `param`: Parameters of `Demod`. This menu option invokes the menu defined in Section 13.5.1 on page 65.

- `variables`: Changeable variables of `Demod`. This menu option invokes the menu defined in Section 13.5.2 on page 65.
- `members`: Members of `Demod`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Demod`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Demod` with default parameters. This menu option is a command.
- `create`: Create instance of `Demod`. This menu option is a command.

13.5.1 Menu of Parameters for `Demod` Object

The commands in this menu are:

- `DataType`: Parameter `DataType` of `Demod` object. This menu option is a command.

  `DataType` selects complex input and complex output (0), complex input and real output (1), real input and complex output (2) or real input and real output (3).

- `DemodFreq`: Parameter `DemodFreq` of `Demod` object. This menu option is a command.

  `DemodFreq` is the demodulation frequency in radians/sample. Input sample N is multiplied by e∧(-i × N × 2 × Pi × `DemodFreq`)

13.5.2 Menu of Variables for Demod Object

The commands in this menu are:

- `DemodFreq`: Changeable variable `DemodFreq` of `Demod` object. This menu option is a command.

  `DemodFreq` is the demodulation frequency in radians/sample. Input sample N is multiplied by e∧(-i × N × 2 × Pi × `DemodFreq`)

13.6 Options for `Demux`

The commands in this menu are:

- `help`: Explain the use of `Demux`. This menu option is a command.
- `param`: Parameters of `Demux`. This menu option invokes the menu defined in Section 13.6.1 on page 66.
- `members`: Members of `Demux`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Demux`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Demux` with default parameters. This menu option is a command.
- `create`: Create instance of `Demux`. This menu option is a command.

13.6.1 Menu of Parameters for `Demux` Object

The commands in this menu are:

- `Channels`: Parameter `Channels` of `Demux` object. This menu option is a command.

  `Demux` demultiplexes a single input channel into `Channels` output channels.

- `InputSampleSize`: Parameter `InputSampleSize` of `Demux` object. This menu option is a command.

  The input channel has samples of `InputSampleSize` words.

- `InputElementSize`: Parameter `InputElementSize` of `Demux` object. This menu option is a command.

  The parameter does not affect the demultiplexing loop. The input channel must have this value for `InputElementSize`. If you are demultiplexing an element into its component parts (such as demultiplexing complex data to real and imaginary streams) `Channels` and `InputElementSize` must have the same value.

- `OutputElementSize`: Parameter `OutputElementSize` of `Demux` object. This menu option is a command.

  The parameter does not affect the demultiplexing loop. All output channels have this value for `OutputElementSize`.

13.7 Options for `FindStartTail`

The commands in this menu are:

- `help`: Explain the use of `FindStartTail`. This menu option is a command.
- `param`: Parameters of `FindStartTail`. This menu option invokes the menu defined in Section 13.7.1 on page 68.
- `members`: Members of `FindStartTail`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `FindStartTail`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `FindStartTail` with default parameters. This menu option is a command.

- `create`: Create instance of `FindStartTail`. This menu option is a command.

13.7.1 Menu of Parameters for `FindStartTail` Object

The commands in this menu are:

- `LowerBound`: Parameter `LowerBound` of `FindStartTail` object. This menu option is a command.

  A sample containing an initial element > `LowerBound` will be ignored. Once one element of a sample has passed both bound tests all later samples will be passed to the next node. If the first element in a sample passes the tests then that sample will be passed.

- `UpperBound`: Parameter `UpperBound` of `FindStartTail` object. This menu option is a command.

  An initial element < `UpperBound` will be ignored. Once one element of a sample has passed both bound tests it and all later samples will be passed to the next node. If the first element in a sample passes the tests then that sample will be passed.

- `Flags`: Parameter `Flags` of `FindStartTail` object. This menu option is a command.

  If bit 2 is set then the first bounds test is ignored. If bit 3 is set the second test is ignored. If both bits 2 and 3 are set you can skip a fixed number of samples by setting `Skip`.

- `Skip`: Parameter `Skip` of `FindStartTail` object. This menu option is a command.

  The first `Skip` samples are read but not written.

13.8 Options for `Gain`

The commands in this menu are:

- `help`: Explain the use of `Gain`. This menu option is a command.
- `param`: Parameters of `Gain`. This menu option invokes the menu defined in Section 13.8.1 on page 69.
- `variables`: Changeable variables of `Gain`. This menu option invokes the menu defined in Section 13.8.2 on page 69.
- `members`: Members of `Gain`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Gain`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Gain` with default parameters. This menu option is a command.
- `create`: Create instance of `Gain`. This menu option is a command.

13.8.1 Menu of Parameters for `Gain` Object

The commands in this menu are:

- `Scale`: Parameter `Scale` of `Gain` object. This menu option is a command.

  `Scale` specifies the ratio of input amplitude to output amplitude. Integer overflows are prevented by clipping. The first time clipping occurs a help message is generated. A new help message is generated after every 400 clippings.

13.8.2 Menu of Variables for Gain Object

The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `Gain` object. This menu option is a command.

  `Scale` specifies the ratio of input amplitude to output amplitude. Integer overflows are prevented by clipping. The first time clipping occurs a help message is generated. A new help message is generated after every 400 clippings.

13.9 Options for `GainPad`

The commands in this menu are:

- `help`: Explain the use of `GainPad`. This menu option is a command.
- `param`: Parameters of `GainPad`. This menu option invokes the menu defined in Section 13.9.1 on page 70.
- `variables`: Changeable variables of `GainPad`. This menu option invokes the menu defined in Section 13.9.2 on page 71.
- `members`: Members of `GainPad`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `GainPad`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `GainPad` with default parameters. This menu option is a command.
- `create`: Create instance of `GainPad`. This menu option is a command.

13.9.1 Menu of Parameters for `GainPad` Object

The commands in this menu are:

- `Scale`: Parameter `Scale` of `GainPad` object. This menu option is a command.

  `Scale` specifies the ratio of input amplitude to output amplitude. Integer overflows are prevented by clipping. The first time clipping occurs a help message is generated. A new help message is generated after every 400 clippings.

- `ElementSize`: Parameter `ElementSize` of `GainPad` object. This menu option is a command.

  `ElementSize` specifies the sample size. The most common use of `ElementSize` is to set it to two for a complex data stream or 1 for real data. Set it to 0 to convert real to complex data by padding the imaginary part with 0.

- `NullOutputSample`: Parameter `NullOutputSample` of `GainPad` object. This menu option is a command.

  If `NullOutputSample` is non zero then all samples after `NullOutputSample` will be zero. If `NullOutputSample` is 0 then the input is written to the output continuously.

13.9.2 Menu of Variables for GainPad Object

The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `GainPad` object. This menu option is a command.

  `Scale` specifies the ratio of input amplitude to output amplitude. Integer overflows are prevented by clipping. The first time clipping occurs a help message is generated. A new help message is generated after every 400 clippings.

13.10 Options for `Integrate`

The commands in this menu are:

- `help`: Explain the use of `Integrate`. This menu option is a command.
- `param`: Parameters of `Integrate`. This menu option invokes the menu defined in Section 13.10.1 on page 72.
- `variables`: Changeable variables of `Integrate`. This menu option invokes the menu defined in Section 13.10.2 on page 73.
- `members`: Members of `Integrate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Integrate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Integrate` with default parameters. This menu option is a command.
- `create`: Create instance of `Integrate`. This menu option is a command.

13.10.1 Menu of Parameters for `Integrate` Object

The commands in this menu are:

- `IntegrationSize`: Parameter `IntegrationSize` of `Integrate` object. This menu option is a command.

  `IntegrationSize` is the number of samples to sum.
- `OutputStep`: Parameter `OutputStep` of `Integrate` object. This menu option is a command.

  An output is generated for every `OutputStep` inputs. The number of buffers that must be maintained is `IntegrationSize / OutputStep`.
- `Scale`: Parameter `Scale` of `Integrate` object. This menu option is a command.

  `Scale` multiplies each output sample. The intermediate arithmetic is done in double floating point point. `Scale` is applied to the final output point before it is converted to MachWord.

13.10.2 Menu of Variables for Integrate Object

The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `Integrate` object. This menu option is a command.

  `Scale` multiplies each output sample. The intermediate arithmetic is done in double floating point point. `Scale` is applied to the final output point before it is converted to MachWord.

13.11 Options for `Interpolate`

The commands in this menu are:

- `help`: Explain the use of `Interpolate`. This menu option is a command.
- `param`: Parameters of `Interpolate`. This menu option invokes the menu defined in Section 13.11.1 on page 73.
- `members`: Members of `Interpolate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Interpolate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Interpolate` with default parameters. This menu option is a command.
- `create`: Create instance of `Interpolate`. This menu option is a command.

13.11.1 Menu of Parameters for `Interpolate` Object

The commands in this menu are:

- `DeltaIn`: Parameter `DeltaIn` of `Interpolate` object. This menu option is a command.

  `DeltaIn` is the number of input samples for `DeltaOut` output samples.

- `DeltaOut`: Parameter `DeltaOut` of `Interpolate` object. This menu option is a command.

  `DeltaOut` is the number of output samples generated from `DeltaIn` input samples.

13.12 Options for `MaskWord`

The commands in this menu are:

- `help`: Explain the use of `MaskWord`. This menu option is a command.
- `param`: Parameters of `MaskWord`. This menu option invokes the menu defined in Section 13.12.1 on page 74.
- `members`: Members of `MaskWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `MaskWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `MaskWord` with default parameters. This menu option is a command.
- `create`: Create instance of `MaskWord`. This menu option is a command.

13.12.1 Menu of Parameters for `MaskWord` Object

The commands in this menu are:

- `Mask`: Parameter `Mask` of `MaskWord` object. This menu option is a command.

  `Mask` will be applied to each input sample to create an output sample.

13.13 Options for `Mux`

The commands in this menu are:

- `help`: Explain the use of `Mux`. This menu option is a command.
- `param`: Parameters of `Mux`. This menu option invokes the menu defined in Section 13.13.1 on page 75.
- `members`: Members of `Mux`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Mux`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Mux` with default parameters. This menu option is a command.
- `create`: Create instance of `Mux`. This menu option is a command.

13.13.1 Menu of Parameters for `Mux` Object

The commands in this menu are:

- `Channels`: Parameter `Channels` of `Mux` object. This menu option is a command.

  `Mux` combines `Channels` input channels into a single output channel.

- `InputSampleSize`: Parameter `InputSampleSize` of `Mux` object. This menu option is a command.

  For each input channel it is assumed that a single sample is made up of `InputSampleSize` words.

- `OutputSampleSize`: Parameter `OutputSampleSize` of `Mux` object. This menu option is a command.

  The number of words in an output channel sample is `OutputSampleSize`.

- `MinimumChunk`: Parameter `MinimumChunk` of `Mux` object. This menu option is a command.

  `MinimumChunk` can be set to a minimum number of input samples to be processed. This allows for greater efficiency but limits the degree to which data can be flushed.

13.14 Options for `PackWord`

The commands in this menu are:

- `help`: Explain the use of `PackWord`. This menu option is a command.
- `param`: Parameters of `PackWord`. This menu option invokes the menu defined in Section 13.14.1 on page 77.
- `members`: Members of `PackWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `PackWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `PackWord` with default parameters. This menu option is a command.
- `create`: Create instance of `PackWord`. This menu option is a command.

13.14.1 Menu of Parameters for `PackWord` Object

The commands in this menu are:

- `InputWordSize`: Parameter `InputWordSize` of `PackWord` object. This menu option is a command.

  `InputWordSize` is the number of bits in each input word to pack into the output word.
- `InputsPerOutput`: Parameter `InputsPerOutput` of `PackWord` object. This menu option is a command.

  `InputsPerOutput` is the number of input words combined to form a single output word.

13.15 Options for `Power`

The commands in this menu are:

- `help`: Explain the use of `Power`. This menu option is a command.
- `param`: Parameters of `Power`. This menu option invokes the menu defined in Section 13.15.1 on page 78.
- `variables`: Changeable variables of `Power`. This menu option invokes the menu defined in Section 13.15.2 on page 78.
- `members`: Members of `Power`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Power`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Power` with default parameters. This menu option is a command.
- `create`: Create instance of `Power`. This menu option is a command.

13.15.1 Menu of Parameters for `Power` Object

The commands in this menu are:

- `Amplitude`: Parameter `Amplitude` of `Power` object. This menu option is a command.

  If `Amplitude` is set to 1, the output will be the square root of the power and not the power.

- `Scale`: Parameter `Scale` of `Power` object. This menu option is a command.

  `Scale` is a linear scale factor applied before summing the squared elements in a sample. For integer arithmetic, `Scale` should be set to prevent overflows. Note the squaring operation (in the integer arithmetic model) is done in double precision integer arithmetic. Thus if one is taking the amplitude as the final output (`Amplitude` =1), overflows can only occur from the summation step. If integer overflows do occur the output signal is clipped. The first time clipping occurs, a help message is generated. A new help message is generated after every 400 clippings.

13.15.2 Menu of Variables for Power Object

The commands in this menu are:

- `Scale`: Changeable variable `Scale` of `Power` object. This menu option is a command.

  `Scale` is a linear scale factor applied before summing the squared elements in a sample. For integer arithmetic, `Scale` should be set to prevent overflows. Note the squaring operation (in the integer arithmetic model) is done in double precision integer arithmetic. Thus if one is taking the amplitude as the final output (`Amplitude` =1), overflows can only occur from the summation step. If integer overflows do occur the output signal is clipped. The first time clipping occurs, a help message is generated. A new help message is generated after every 400 clippings.

13.16 Options for `RealFir`

The commands in this menu are:

- `help`: Explain the use of `RealFir`. This menu option is a command.
- `param`: Parameters of `RealFir`. This menu option invokes the menu defined in Section 13.16.1 on page 79.
- `members`: Members of `RealFir`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `RealFir`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `RealFir` with default parameters. This menu option is a command.
- `create`: Create instance of `RealFir`. This menu option is a command.

13.16.1 Menu of Parameters for `RealFir` Object

The commands in this menu are:

- `Resample`: Parameter `Resample` of `RealFir` object. This menu option is a command.

  `Resample` specifies the filter resampling factor. This is the ratio of the input sampling rate to the output sampling rate.

- `ZeroPad`: Parameter `ZeroPad` of `RealFir` object. This menu option is a command.

  `ZeroPad` specifies the number of 0's that are added after each input sample.

- `Odd`: Parameter `Odd` of `RealFir` object. This menu option is a command. `Odd` determines if the filter is if odd (`Odd` =1) or even (`Odd` =0).
- `Coeff`: Parameter `Coeff` of `RealFir` object. This menu option is a command.

  `Coeff` is the list of filter coefficients. The filter is symmetric and only half (or half plus one for odd length filters) are specified. The first in the list is the first coefficient of the filter. The middle coefficient is at the end of the list. The default values for the coefficients define a low pass FIR filter with a pass band of .125 times the total bandwidth and transition band of .375 times the bandwidth. The pass band is extremely flat and the stop band is down over 100 db. This is an overdesigned filter for most practical applications but it provides a good test case. The performance will be degraded by 16 bit integer arithmetic.

13.17 Options for `RepackStream`

The commands in this menu are:

- `help`: Explain the use of `RepackStream`. This menu option is a command.
- `param`: Parameters of `RepackStream`. This menu option invokes the menu defined in Section 13.17.1 on page 81.
- `members`: Members of `RepackStream`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `RepackStream`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `RepackStream` with default parameters. This menu option is a command.
- `create`: Create instance of `RepackStream`. This menu option is a command.

13.17.1 Menu of Parameters for `RepackStream` Object

The commands in this menu are:

- `OutputWordSize`: Parameter `OutputWordSize` of `RepackStream` object. This menu option is a command.

  `OutputWordSize` is the number of bits in each output word. The output is treated as a continuous stream of bits made up of the least significant `OutputWordSize` bits from each physical output word.

- `InputWordSize`: Parameter `InputWordSize` of `RepackStream` object. This menu option is a command.

  `InputWordSize` is the number of bits in the input word. The input is treated as a continuous stream of bits made up of the least significant `OutputWordSize` bits from each physical output word.

- `SignedOutput`: Parameter `SignedOutput` of `RepackStream` object. This menu option is a command.

  If `SignedOutput` is set the output will be written as a signed two's compliment value in the full physical word size.

13.18 Options for `SampleDelay`

The commands in this menu are:

- `help`: Explain the use of `SampleDelay`. This menu option is a command.
- `param`: Parameters of `SampleDelay`. This menu option invokes the menu defined in Section 13.18.1 on page 82.
- `members`: Members of `SampleDelay`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `SampleDelay`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.

- `create default`: Create instance of `SampleDelay` with default parameters. This menu option is a command.
- `create`: Create instance of `SampleDelay`. This menu option is a command.

13.18.1 Menu of Parameters for `SampleDelay` Object

The commands in this menu are:

- `Delta`: Parameter `Delta` of `SampleDelay` object. This menu option is a command.

  `Delta` specifies the number of samples the output signal will be delayed relative to the input.
- `FillValue`: Parameter `FillValue` of `SampleDelay` object. This menu option is a command.

  `FillValue` is the value to be output for each sample component for the first `Delta` samples.

13.19 Options for `ToInteger`

The commands in this menu are:

- `help`: Explain the use of `ToInteger`. This menu option is a command.
- `members`: Members of `ToInteger`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `ToInteger`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ToInteger` with default parameters. This menu option is a command.

- `create`: Create instance of `ToInteger`. This menu option is a command.

13.20 Options for `ToMach`

The commands in this menu are:

- `help`: Explain the use of `ToMach`. This menu option is a command.
- `param`: Parameters of `ToMach`. This menu option invokes the menu defined in Section 13.20.1 on page 83.
- `members`: Members of `ToMach`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `ToMach`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ToMach` with default parameters. This menu option is a command.
- `create`: Create instance of `ToMach`. This menu option is a command.

13.20.1 Menu of Parameters for `ToMach` Object

The commands in this menu are:

- `SignedConversion`: Parameter `SignedConversion` of `ToMach` object. This menu option is a command.

  If `SignedConversion` is set the input integer will treated as a two's compliment signed value. Otherwise it will be considered unsigned.

13.21 Options for `Truncate`

The commands in this menu are:

- `help`: Explain the use of `Truncate`. This menu option is a command.
- `param`: Parameters of `Truncate`. This menu option invokes the menu defined in Section 13.21.1 on page 84.
- `variables`: Changeable variables of `Truncate`. This menu option invokes the menu defined in Section 13.21.2 on page 85.
- `members`: Members of `Truncate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Truncate`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Truncate` with default parameters. This menu option is a command.
- `create`: Create instance of `Truncate`. This menu option is a command.

13.21.1 Menu of Parameters for `Truncate` Object

The commands in this menu are:

- `Range`: Parameter `Range` of `Truncate` object. This menu option is a command.

  `Range` is the number of bits in the dynamic range of the output (not counting the sign bit).
- `Accuracy`: Parameter `Accuracy` of `Truncate` object. This menu option is a command.

`Accuracy` is the number of significant bits retained in the output (not counting the sign bit). `Accuracy` will always be <= `Range`. If you specify more accuracy then range accuracy will equal range.

- `OverflowMode`: Parameter `OverflowMode` of `Truncate` object. This menu option is a command.

  `OverflowMode` selects saturation (0) or truncation (1) mode. In saturation mode an overflow is replaced by the largest positive or negative number representable depending on the sign of the input value. In Truncation mode the high order bits are truncated as if the number was in sign magnitude form.

- `Round`: Parameter `Round` of `Truncate` object. This menu option is a command.

  `Round` if set rounds the result. Otherwise it is truncated. Truncation is always towards 0 and not two's compliment truncation.

13.21.2 Menu of Variables for Truncate Object

The commands in this menu are:

- `Range`: Changeable variable `Range` of `Truncate` object. This menu option is a command.

  `Range` is the number of bits in the dynamic range of the output (not counting the sign bit).

- `Accuracy`: Changeable variable `Accuracy` of `Truncate` object. This menu option is a command.

  `Accuracy` is the number of significant bits retained in the output (not counting the sign bit). `Accuracy` will always be <= `Range`. If you specify more accuracy then range accuracy will equal range.

- `OverflowMode`: Changeable variable `OverflowMode` of `Truncate` object. This menu option is a command.

  `OverflowMode` selects saturation (0) or truncation (1) mode. In saturation mode an overflow is replaced by the largest positive or negative number representable depending on the sign of the input value. In Truncation mode the high order bits are truncated as if the number was in sign magnitude form.

- `Round`: Changeable variable `Round` of `Truncate` object. This menu option is a command.

  `Round` if set rounds the result. Otherwise it is truncated. Truncation is always towards 0 and not two's compliment truncation.

13.22 Options for `UnpackWord`

The commands in this menu are:

- `help`: Explain the use of `UnpackWord`. This menu option is a command.
- `param`: Parameters of `UnpackWord`. This menu option invokes the menu defined in Section 13.22.1 on page 86.
- `members`: Members of `UnpackWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.21 on page 161 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `UnpackWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `UnpackWord` with default parameters. This menu option is a command.
- `create`: Create instance of `UnpackWord`. This menu option is a command.

13.22.1 Menu of Parameters for `UnpackWord` Object

The commands in this menu are:

- `OutputWordSize`: Parameter `OutputWordSize` of `UnpackWord` object. This menu option is a command.

  `OutputWordSize` is the number of bits in the output word.

- `OutputsPerInput`: Parameter `OutputsPerInput` of `UnpackWord` object. This menu option is a command.

  `OutputsPerInput` is the number of output words unpacked from each input word.

- `SignedOutput`: Parameter `SignedOutput` of `UnpackWord` object. This menu option is a command.

  If `SignedOutput` is set the output will be written as a signed two's compliment value in the full physical word size.

14 Signal generator objects

There are several ways in which test signals can be generated. The `signal` menu describes the signal generators and their parameters and instances of them.

The commands in this menu are:

- `ConstantData`: generate a `MachWord` constant. This menu option invokes the menu defined in Section 14.1 on page 88.

- `Cos`: Generates the real function `Amplitude` cos(`Phase` + N `Frequency`). This menu option invokes the menu defined in Section 14.2 on page 89.

- `CxCos`: Generates the function `Amplitude` e∧(2 Pi i(`Phase` + N `Frequency`)). This menu option invokes the menu defined in Section 14.3 on page 91.

- `CxImp`: Generates a periodic impulse or square wave. This menu option invokes the menu defined in Section 14.4 on page 93.

- `Normal`: Generate normally distributed noise samples. This menu option invokes the menu defined in Section 14.5 on page 94.

- `Ramp`: generates a linear ramp function. This menu option invokes the menu defined in Section 14.6 on page 96.
- `UniformNoise`: Generate uniformly distributed noise samples. This menu option invokes the menu defined in Section 14.7 on page 98.

14.1 Options for `ConstantData`

The commands in this menu are:

- `help`: Explain the use of `ConstantData`. This menu option is a command.
- `param`: Parameters of `ConstantData`. This menu option invokes the menu defined in Section 14.1.1 on page 88.
- `variables`: Changeable variables of `ConstantData`. This menu option invokes the menu defined in Section 14.1.2 on page 89.
- `members`: Members of `ConstantData`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `ConstantData`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ConstantData` with default parameters. This menu option is a command.
- `create`: Create instance of `ConstantData`. This menu option is a command.

14.1.1 Menu of Parameters for `ConstantData` Object

The commands in this menu are:

- `Value`: Parameter `Value` of `ConstantData` object. This menu option is a command.

  `Value` is output as an an `MachWord` constant.

14.1.2 Menu of Variables for ConstantData Object

The commands in this menu are:

- `Value`: Changeable variable `Value` of `ConstantData` object. This menu option is a command.

  `Value` is output as an an `MachWord` constant.

14.2 Options for `Cos`

The commands in this menu are:

- `help`: Explain the use of `Cos`. This menu option is a command.
- `param`: Parameters of `Cos`. This menu option invokes the menu defined in Section 14.2.1 on page 90.
- `variables`: Changeable variables of `Cos`. This menu option invokes the menu defined in Section 14.2.2 on page 90.
- `members`: Members of `Cos`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Cos`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Cos` with default parameters. This menu option is a command.
- `create`: Create instance of `Cos`. This menu option is a command.

14.2.1 Menu of Parameters for `Cos` Object

The commands in this menu are:

- `Frequency`: Parameter `Frequency` of `Cos` object. This menu option is a command.

  `Frequency` specifies the signal frequency in radians per sample. In other words the phase of a given sample is `Frequency` radians plus the phase of the previous sample.

- `Phase`: Parameter `Phase` of `Cos` object. This menu option is a command.

  `Phase` specifies the initial phase of the first sample of the signal.

- `Amplitude`: Parameter `Amplitude` of `Cos` object. This menu option is a command.

  `Amplitude` specifies the maximum amplitude of the continuous cosine function. This may not be the maximum amplitude of the samples generated. If the function is sampled at a phase that is an integer multiple of Pi, then the samples will obtain this maximum.

14.2.2 Menu of Variables for Cos Object

The commands in this menu are:

- `Frequency`: Changeable variable `Frequency` of `Cos` object. This menu option is a command.

  `Frequency` specifies the signal frequency in radians per sample. In other words the phase of a given sample is `Frequency` radians plus the phase of the previous sample.

- `Phase`: Changeable variable `Phase` of `Cos` object. This menu option is a command.

  `Phase` specifies the initial phase of the first sample of the signal.

- `Amplitude`: Changeable variable `Amplitude` of `Cos` object. This menu option is a command.

`Amplitude` specifies the maximum amplitude of the continuous cosine function. This may not be the maximum amplitude of the samples generated. If the function is sampled at a phase that is an integer multiple of Pi, then the samples will obtain this maximum.

14.3 Options for `CxCos`

The commands in this menu are:

- `help`: Explain the use of `CxCos`. This menu option is a command.
- `param`: Parameters of `CxCos`. This menu option invokes the menu defined in Section 14.3.1 on page 91.
- `variables`: Changeable variables of `CxCos`. This menu option invokes the menu defined in Section 14.3.2 on page 92.
- `members`: Members of `CxCos`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `CxCos`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `CxCos` with default parameters. This menu option is a command.
- `create`: Create instance of `CxCos`. This menu option is a command.

14.3.1 Menu of Parameters for `CxCos` Object

The commands in this menu are:

- `Frequency`: Parameter `Frequency` of `CxCos` object. This menu option is a command.

  `Frequency` specifies the signal frequency in radians per sample. In other words the phase of a given sample is `Frequency` radians plus the phase of the previous sample.

- `Phase`: Parameter `Phase` of `CxCos` object. This menu option is a command.

  `Phase` specifies the initial phase of the first sample of the signal.

- `Amplitude`: Parameter `Amplitude` of `CxCos` object. This menu option is a command.

  `Amplitude` specifies the maximum amplitude of the continuous cosine function. This may not be the maximum amplitude of the samples generated. If the function is sampled at a phase that is an integer multiple of Pi, then the samples will obtain this maximum.

14.3.2 Menu of Variables for CxCos Object

The commands in this menu are:

- `Frequency`: Changeable variable `Frequency` of `CxCos` object. This menu option is a command.

  `Frequency` specifies the signal frequency in radians per sample. In other words the phase of a given sample is `Frequency` radians plus the phase of the previous sample.

- `Phase`: Changeable variable `Phase` of `CxCos` object. This menu option is a command.

  `Phase` specifies the initial phase of the first sample of the signal.

- `Amplitude`: Changeable variable `Amplitude` of `CxCos` object. This menu option is a command.

  `Amplitude` specifies the maximum amplitude of the continuous cosine function. This may not be the maximum amplitude of the samples generated. If the function is sampled at a phase that is an integer multiple of Pi, then the samples will obtain this maximum.

14.4 Options for `CxImp`

The commands in this menu are:

- `help`: Explain the use of `CxImp`. This menu option is a command.
- `param`: Parameters of `CxImp`. This menu option invokes the menu defined in Section 14.4.1 on page 93.
- `members`: Members of `CxImp`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `CxImp`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `CxImp` with default parameters. This menu option is a command.
- `create`: Create instance of `CxImp`. This menu option is a command.

14.4.1 Menu of Parameters for `CxImp` Object

The commands in this menu are:

- `Period`: Parameter `Period` of `CxImp` object. This menu option is a command.

  `Period` specifies the number of samples before the impulse is repeated.

- `Phase`: Parameter `Phase` of `CxImp` object. This menu option is a command.

  `Phase` the relative amplitude of the real and imaginary components of the signal. With `Phase` = 0 all the energy is in the real part. With `Phase` = pi/2 all the energy is in the imaginary part.

- `Amplitude`: Parameter `Amplitude` of `CxImp` object. This menu option is a command.

  `Amplitude` specifies the the magnitude of the impulse amplitude. It is the square root of the sum of the squares of the real and imaginary amplitudes.

- `Width`: Parameter `Width` of `CxImp` object. This menu option is a command.

  `Width` specifies the peak width as a fraction of sample period. `Width` = 0 produces an impulse one sample wide. `Width` = 1 results in a constant amplitude and phase signal. `Width` = .5 results in a standard square wave.

- `Transition`: Parameter `Transition` of `CxImp` object. This menu option is a command.

  `Transition` specifies the sample index where the first signal transition from 0 occurs. This may be longer than the sample `Period`.

14.5 Options for `Normal`

The commands in this menu are:

- `help`: Explain the use of `Normal`. This menu option is a command.
- `param`: Parameters of `Normal`. This menu option invokes the menu defined in Section 14.5.1 on page 95.
- `variables`: Changeable variables of `Normal`. This menu option invokes the menu defined in Section 14.5.2 on page 95.
- `members`: Members of `Normal`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Normal`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.

- `create default`: Create instance of `Normal` with default parameters. This menu option is a command.
- `create`: Create instance of `Normal`. This menu option is a command.

14.5.1 Menu of Parameters for `Normal` Object

The commands in this menu are:

- `Sigma`: Parameter `Sigma` of `Normal` object. This menu option is a command.

  `Sigma` specifies the standard deviation of the normally distributed samples created by this generator. `Sigma` is a scale factor for values generated in the standard normal distribution.

- `Mean`: Parameter `Mean` of `Normal` object. This menu option is a command.

  `Mean` specifies the mean of the normally distributed samples created by this generator. `Mean` is an offset for the values generated in the standard normal distribution.

- `ElementSize`: Parameter `ElementSize` of `Normal` object. This menu option is a command.

  `ElementSize` specifies the number of words in a single sample. It is most commonly 1 for real data or 2 for complex data.

- `Seed`: Parameter `Seed` of `Normal` object. This menu option is a command.

  `Seed` seeds the random number generator. Each object instance maintains a separate history state for the random number generator. If the same value of `Seed` is used in different object instances they will generate the same sequence.

14.5.2 Menu of Variables for Normal Object

The commands in this menu are:

- `Sigma`: Changeable variable `Sigma` of `Normal` object. This menu option is a command.

  `Sigma` specifies the standard deviation of the normally distributed samples created by this generator. `Sigma` is a scale factor for values generated in the standard normal distribution.

- `Mean`: Changeable variable `Mean` of `Normal` object. This menu option is a command.

  `Mean` specifies the mean of the normally distributed samples created by this generator. `Mean` is an offset for the values generated in the standard normal distribution.

14.6 Options for `Ramp`

The commands in this menu are:

- `help`: Explain the use of `Ramp`. This menu option is a command.
- `param`: Parameters of `Ramp`. This menu option invokes the menu defined in Section 14.6.1 on page 97.
- `variables`: Changeable variables of `Ramp`. This menu option invokes the menu defined in Section 14.6.2 on page 97.
- `members`: Members of `Ramp`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Ramp`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Ramp` with default parameters. This menu option is a command.
- `create`: Create instance of `Ramp`. This menu option is a command.

14.6.1 Menu of Parameters for `Ramp` Object

The commands in this menu are:

- `Min`: Parameter `Min` of `Ramp` object. This menu option is a command.

  `Min` is the minimum and initial value of the ramp function.

- `Max`: Parameter `Max` of `Ramp` object. This menu option is a command.

  `Max` is an upper bound on the ramp function. Before exceeding `Max` a sample will be reset to `Min`.

- `Increment`: Parameter `Increment` of `Ramp` object. This menu option is a command.

  `Increment` is the amount added to the previous sample to generate the next sample.

14.6.2 Menu of Variables for Ramp Object

The commands in this menu are:

- `Min`: Changeable variable `Min` of `Ramp` object. This menu option is a command.

  `Min` is the minimum and initial value of the ramp function.

- `Max`: Changeable variable `Max` of `Ramp` object. This menu option is a command.

  `Max` is an upper bound on the ramp function. Before exceeding `Max` a sample will be reset to `Min`.

- `Increment`: Changeable variable `Increment` of `Ramp` object. This menu option is a command.

  `Increment` is the amount added to the previous sample to generate the next sample.

14.7 Options for `UniformNoise`

The commands in this menu are:

- `help`: Explain the use of `UniformNoise`. This menu option is a command.
- `param`: Parameters of `UniformNoise`. This menu option invokes the menu defined in Section 14.7.1 on page 98.
- `variables`: Changeable variables of `UniformNoise`. This menu option invokes the menu defined in Section 14.7.2 on page 99.
- `members`: Members of `UniformNoise`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `UniformNoise`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `UniformNoise` with default parameters. This menu option is a command.
- `create`: Create instance of `UniformNoise`. This menu option is a command.

14.7.1 Menu of Parameters for `UniformNoise` Object

The commands in this menu are:

- `Maximum`: Parameter `Maximum` of `UniformNoise` object. This menu option is a command.

  `Maximum` is the largest value of uniformly distributed noise.
- `Minimum`: Parameter `Minimum` of `UniformNoise` object. This menu option is a command.

`Minimum` is the smallest value of uniformly distributed noise. Most commonly `Minimum` = - `Maximum`.

- `ElementSize`: Parameter `ElementSize` of `UniformNoise` object. This menu option is a command.

  `ElementSize` specifies the number of words in a single sample. It is most commonly 1 for real data or 2 for complex data.

- `Seed`: Parameter `Seed` of `UniformNoise` object. This menu option is a command.

  `Seed` seeds the random number generator. Each object instance maintains a separate history state for the random number generator. If the same value of `Seed` is used in different object instances they will generate the same.

14.7.2 Menu of Variables for UniformNoise Object

The commands in this menu are:

- `Maximum`: Changeable variable `Maximum` of `UniformNoise` object. This menu option is a command.

  `Maximum` is the largest value of uniformly distributed noise.

- `Minimum`: Changeable variable `Minimum` of `UniformNoise` object. This menu option is a command.

  `Minimum` is the smallest value of uniformly distributed noise. Most commonly `Minimum` = - `Maximum`.

15 Plotting objects

Frequently DSP output is most easily evaluated with a data plot. The `plot` menu lists the plotting objects their parameters and instances of them.

The commands in this menu are:

- `EyePlot`: `EyePlot` plots complex signal in eye plot (X versus Y) form. This menu option invokes the menu defined in Section 15.1 on page 100.
- `Plot`: `Plot` creates graphs of real, complex and two dimensional data streams. This menu option invokes the menu defined in Section 15.2 on page 101.

15.1 Options for `EyePlot`

The commands in this menu are:

- `help`: Explain the use of `EyePlot`. This menu option is a command.
- `param`: Parameters of `EyePlot`. This menu option invokes the menu defined in Section 15.1.1 on page 100.
- `members`: Members of `EyePlot`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `EyePlot`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `EyePlot` with default parameters. This menu option is a command.
- `create`: Create instance of `EyePlot`. This menu option is a command.

15.1.1 Menu of Parameters for `EyePlot` Object

The commands in this menu are:

- `SamplesPerPlot`: Parameter `SamplesPerPlot` of `EyePlot` object. This menu option is a command.

  `EyePlot` generates a series of plots with `SamplesPerPlot` in each display. If the input block size is not 1 then that value is overrides this parameter.

- `Caption`: Parameter `Caption` of `EyePlot` object. This menu option is a command.

  The plot caption is a string that will be displayed at the base of the plot. The default value of 0 causes the plot node name to be used for the caption.

15.2 Options for `Plot`

The commands in this menu are:

- `help`: Explain the use of `Plot`. This menu option is a command.
- `param`: Parameters of `Plot`. This menu option invokes the menu defined in Section 15.2.1 on page 101.
- `members`: Members of `Plot`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `Plot`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Plot` with default parameters. This menu option is a command.
- `create`: Create instance of `Plot`. This menu option is a command.

15.2.1 Menu of Parameters for `Plot` Object

The commands in this menu are:

- `Caption`: Parameter `Caption` of `Plot` object. This menu option is a command.

The `Caption` is displayed at the base of the plot. The default value of 0 causes the plot node name to be used for the caption. The caption cannot contain blanks. Use underscore instead.

16 Listing objects

It is often useful to provide numeric listings of DSP data streams. The `list` menu describes the listing objects their parameters and instances of them.

The commands in this menu are:

- `HexList`: `HexList` lists a specified number of channels to a display window. This menu option invokes the menu defined in Section 16.1 on page 102.
- `Listing`: `Listing` lists a specified number of channels to a display window. This menu option invokes the menu defined in Section 16.2 on page 103.

16.1 Options for `HexList`

The commands in this menu are:

- `help`: Explain the use of `HexList`. This menu option is a command.
- `param`: Parameters of `HexList`. This menu option invokes the menu defined in Section 16.1.1 on page 103.
- `members`: Members of `HexList`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `HexList`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.

- `create default`: Create instance of `HexList` with default parameters. This menu option is a command.
- `create`: Create instance of `HexList`. This menu option is a command.

16.1.1 Menu of Parameters for `HexList` Object

The commands in this menu are:

- `Channels`: Parameter `Channels` of `HexList` object. This menu option is a command.

  `HexList` lists `Channels` signals of integer data in a hexadecimal format.

- `Caption`: Parameter `Caption` of `HexList` object. This menu option is a command.

  `Caption` specifies a caption that will appear at the head of the listing. If no caption is specified (default 0) then the node name will be used. The caption cannot contain blanks. Use underscore instead.

16.2 Options for `Listing`

The commands in this menu are:

- `help`: Explain the use of `Listing`. This menu option is a command.
- `param`: Parameters of `Listing`. This menu option invokes the menu defined in Section 16.2.1 on page 104.
- `variables`: Changeable variables of `Listing`. This menu option invokes the menu defined in Section 16.2.2 on page 104.
- `members`: Members of `Listing`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.

- `instance`: Describe or delete an instance of `Listing`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Listing` with default parameters. This menu option is a command.
- `create`: Create instance of `Listing`. This menu option is a command.

16.2.1 Menu of Parameters for `Listing` Object

The commands in this menu are:

- `Hex`: Parameter `Hex` of `Listing` object. This menu option is a command.

  `Hex` , when set, displays data in hexadecimal format. If a value will not fit in a 32 bit integer it is hard limited.

- `Caption`: Parameter `Caption` of `Listing` object. This menu option is a command.

  `Caption` specifies a caption that will appear at the head of the listing. If no caption is specified (default 0) then the node name will be used. The caption cannot contain blanks. Use underscore instead.

16.2.2 Menu of Variables for Listing Object

The commands in this menu are:

- `Hex`: Changeable variable `Hex` of `Listing` object. This menu option is a command.

  `Hex` , when set, displays data in hexadecimal format. If a value will not fit in a 32 bit integer it is hard limited.

17 Nodes that access disk files

Options in the `disk` menu support reading and writing of disk files. You can save the output from any node in a network to a disk file. You can use data in a disk file as a signal source. This menu describes these objects, their parameters and instances of them.

The commands in this menu are:

- `ascii`: Nodes to read files. This menu option invokes the menu defined in Section 17.1 on page 105.
- `binary`: Nodes to write files. This menu option invokes the menu defined in Section 17.2 on page 111.

17.1 Nodes to read and write ascii files

The `ascii` menu provides access to nodes for reading and writing ascii formatted files. From the `ascii` menu you can list the existing objects, describe their parameters, create new objects and delete existing ones.

The commands in this menu are:

- `AsciiFile`: `AsciiFile` writes an ascii file of data sent to it. This menu option invokes the menu defined in Section 17.1.1 on page 106.
- `ImportData`: `ImportData` reads an ascii input file. This menu option invokes the menu defined in Section 17.1.2 on page 107.
- `ReadFloat`: `ReadFloat` reads an ascii float input file. This menu option invokes the menu defined in Section 17.1.3 on page 110.
- `ReadInt`: `ReadInt` reads an ascii integer input file. This menu option invokes the menu defined in Section 17.1.4 on page 110.

17.1.1 Options for `AsciiFile`

The commands in this menu are:

- `help`: Explain the use of `AsciiFile`. This menu option is a command.
- `param`: Parameters of `AsciiFile`. This menu option invokes the menu defined in Section 17.1.1.1 on page 106.
- `variables`: Changeable variables of `AsciiFile`. This menu option invokes the menu defined in Section 17.1.1.2 on page 107.
- `members`: Members of `AsciiFile`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `AsciiFile`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `AsciiFile` with default parameters. This menu option is a command.
- `create`: Create instance of `AsciiFile`. This menu option is a command.

17.1.1.1 Menu of Parameters for `AsciiFile` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `AsciiFile` object. This menu option is a command.

  `FileName` specifies the output ascii file name head of the listing. If no caption is specified (default 0) then the node name will be used.

- `Hex`: Parameter `Hex` of `AsciiFile` object. This menu option is a command.

  `Hex` , when set, writes output in hexadecimal format. If the value will not fit in a 32 bit integer it is hard limited and a warning is generated.

- `NoGroup`: Parameter `NoGroup` of `AsciiFile` object. This menu option is a command.

  `NoGroup` , if set, writes data one word per line. The default is to write the words in a sample on a single line (up to 20 words per sample) and to put brackets ‘ { } ’ around blocks if the block size is larger then the sample size. If `NoGroup` is set then the data is written one per line with no grouping information.

- `NoHeader`: Parameter `NoHeader` of `AsciiFile` object. This menu option is a command.

  The default is to write a data header that describes the data format, the time it was created and gives the names of the creating node and network. If this option is set this header is omitted.

17.1.1.2 Menu of Variables for AsciiFile Object The commands in this menu are:

- `Hex`: Changeable variable `Hex` of `AsciiFile` object. This menu option is a command.

  `Hex` , when set, writes output in hexadecimal format. If the value will not fit in a 32 bit integer it is hard limited and a warning is generated.

17.1.2 Options for `ImportData`

The commands in this menu are:

- `help`: Explain the use of `ImportData`. This menu option is a command.
- `param`: Parameters of `ImportData`. This menu option invokes the menu defined in Section 17.1.2.1 on page 108.
- `members`: Members of `ImportData`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.

- `instance`: Describe or delete an instance of `ImportData`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ImportData` with default parameters. This menu option is a command.
- `create`: Create instance of `ImportData`. This menu option is a command.

17.1.2.1 Menu of Parameters for `ImportData` Object

The commands in this menu are:

- `FileName`: Parameter `FileName` of `ImportData` object. This menu option is a command.

  `FileName` specifies the name of the disk file to be read. If no file name is specified (default 0) you will be prompted for a file name when execution starts.

- `Format`: Parameter `Format` of `ImportData` object. This menu option is a command.

  `Format` specifies the format to use in reading data. Any standard `C` input format can be used. The default, `%d` for integer decimal data, for octal data. If the format string ends in `X` or `x` integer data is written to the output stream on a floating point simulator. If the last character is an `s` and the format is a floating point format the data will be normalized on the 16 bit integer simulator, i. e. .5 will be converted to 16384 or half of full scale. If the last two characters of the format string are an underscore (_) and any other character they will be deleted from the format. You can use this to control the type of output or scaling independent of the format for reading data. The type of format is determined by looking for the first occurrence of `%` ad then the first occurrence of one the letters `x`, `d`, `o` ,'f' or `e` after that. Floating point format characters must be preceded by a `l` and others must not. The letter can be in either upper or lower case. If the format is not recognized and error will be generated. `x` and `o` formats will read data as unsigned

32 bit integers. `d` will it as signed 32 bit integers. `e` and 'f' will read it as a double floating point value. Overflows will be reported as warnings.

- `Fields`: Parameter `Fields` of `ImportData` object. This menu option is a command.

  `Fields` specifies the number of data fields on each line. If set to 0 then all data fields that are found on a line will be read (up to a maximum line width of 1024 characters.) A data field is any contiguous string of legal digits separated by white space (blank, tab, the beginning of a line or the end of a line) from other data fields or other information on the line. Decimal and octal fields can start with a `+` or `-`. Hexadecimal fields may start with an optional `0x` or `0X` provided the `Format` string is `%x` or `%X`. The value `Fields` is an upper limit on the fields on a line. There may be fewer fields on a line and even lines with no valid numeric fields.

- `RepeatFlag`: Parameter `RepeatFlag` of `ImportData` object. This menu option is a command.

  Setting `RepeatFlag` causes the file to be read at the beginning once the end of file is encountered. If the file is short it will only be read once and the data will be retained in memory.

- `SkipFields`: Parameter `SkipFields` of `ImportData` object. This menu option is a command.

  `SkipFields` is a list of fields (in increasing order) to skip. Fields start at 1. If all values are 0 no fields are skipped.

- `SkipColumns`: Parameter `SkipColumns` of `ImportData` object. This menu option is a command.

  `SkipColumns` is a list of pairs of column numbers in increasing order. All data in columns starting at the first column number in the pair and ending at the next column number after the second element in the pair will be ignored. Thus the list { 20, 24, 30, 32 } would cause the five columns 20 through 24 and the three columns 30 through 32 to be skipped. If there are an odd number of entries all columns at or following the last entry will be skipped. Columns start at 1. If only values of 0 are entered no columns will be skipped.

17.1.3 Options for `ReadFloat`

The commands in this menu are:

- `help`: Explain the use of `ReadFloat`. This menu option is a command.
- `param`: Parameters of `ReadFloat`. This menu option invokes the menu defined in Section 17.1.3.1 on page 110.
- `members`: Members of `ReadFloat`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `ReadFloat`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ReadFloat` with default parameters. This menu option is a command.
- `create`: Create instance of `ReadFloat`. This menu option is a command.

17.1.3.1 Menu of Parameters for `ReadFloat` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `ReadFloat` object. This menu option is a command.

  `FileName` specifies the disk file to be read. If no name is specified the node name will be used.

17.1.4 Options for `ReadInt`

The commands in this menu are:

- `help`: Explain the use of `ReadInt`. This menu option is a command.

- `param`: Parameters of `ReadInt`. This menu option invokes the menu defined in Section 17.1.4.1 on page 111.
- `members`: Members of `ReadInt`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `ReadInt`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `ReadInt` with default parameters. This menu option is a command.
- `create`: Create instance of `ReadInt`. This menu option is a command.

17.1.4.1 Menu of Parameters for `ReadInt` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `ReadInt` object. This menu option is a command.

  `FileName` specifies the the disk file to be read. If no name is specified the node name will be used.

- `Flags`: Parameter `Flags` of `ReadInt` object. This menu option is a command.

  `Flags` &1 specifies hex format file with optional 0x or 0X prefix for each value. `Flags` &2 will write 32 bit integer data on a floating point simulator. Decimal format files are read as signed integers and hexadecimal format as unsigned.

17.2 Nodes to read and write binary files

The `binary` menu provides access to nodes for reading and writing binary disk files. From the `binary` menu you can list the existing objects, describe their parameters, create new objects and delete existing ones.

The commands in this menu are:

- `CompareDisk`: `CompareDisk` compares input to a file written by an `OutputNode`. This menu option invokes the menu defined in Section 17.2.1 on page 112.
- `InputNode`: `InputNode` reads a disk file written by an `OutputNode`. This menu option invokes the menu defined in Section 17.2.2 on page 115.
- `InputWord`: `InputWord` reads words in a selected format from a binary file. This menu option invokes the menu defined in Section 17.2.3 on page 118.
- `OutputNode`: `OutputNode` writes a specified number of channels to a disk file. This menu option invokes the menu defined in Section 17.2.4 on page 120.
- `OutputWord`: `OutputWord` writes words in a selected format to a binary file. This menu option invokes the menu defined in Section 17.2.5 on page 121.
- `VoiceNode`: `VoiceNode` reads `Creative Voice` format files. This menu option invokes the menu defined in Section 17.2.6 on page 122.
- `VoiceStripOut`: `VoiceStripOut` writes a Creative Voice format file with no header. This menu option invokes the menu defined in Section 17.2.7 on page 124.

17.2.1 Options for `CompareDisk`

The commands in this menu are:

- `help`: Explain the use of `CompareDisk`. This menu option is a command.
- `param`: Parameters of `CompareDisk`. This menu option invokes the menu defined in Section 17.2.1.1 on page 113.
- `members`: Members of `CompareDisk`. This menu option invokes the menu defined in Section 17.2.1.2 on page 113. See Section 22.19 on page 160 for additional base class member functions in this menu.

- `instance`: Describe or delete an instance of `CompareDisk`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `CompareDisk` with default parameters. This menu option is a command.
- `create`: Create instance of `CompareDisk`. This menu option is a command.

17.2.1.1 Menu of Parameters for `CompareDisk` Object

The commands in this menu are:

- `FileName`: Parameter `FileName` of `CompareDisk` object. This menu option is a command.

  `FileName` specifies the name of the disk file to be compared with the data read from the input channels.
- `MaxReport`: Parameter `MaxReport` of `CompareDisk` object. This menu option is a command.

  Only the first `MaxReport` errors will be reported.
- `Tolerance`: Parameter `Tolerance` of `CompareDisk` object. This menu option is a command.

  `Tolerance` is the absolute value of the smallest difference that constitutes an error. Ordinarily this value is 0.0. It might be set to a value larger than 0 to compare slightly different algorithms or results on two different computers with different arithmetic.
- `ErrorFile`: Parameter `ErrorFile` of `CompareDisk` object. This menu option is a command.

  `ErrorFile` is a file in which errors will be reported instead of displaying them in a window.

17.2.1.2 Select a member of `CompareDisk` to describe

The commands in this menu are:

- `DisplayHeader`: Describe member `DisplayHeader` of `CompareDisk`. This menu option invokes the menu defined in Section 17.2.1.2.1 on page 114.
- `IgnoreHeaderCount`: Describe member `IgnoreHeaderCount` of `CompareDisk`. This menu option invokes the menu defined in Section 17.2.1.2.2 on page 114.

17.2.1.2.1 Describe member `DisplayHeader` of `CompareDisk` `DisplayHeader` displays the parameters read from file `FileName`. These include the original node name that generated the file, the caption for this node, the number of input channels, and the number of scalar elements in a sample. The arithmetic type is also shown. The output channels and sample size for this node are determined by these values. If the data in the file is in a different arithmetic format than that currently in use, the file data will be converted.

The commands in this menu are:

- `desc DisplayHeader`: Describe selected member of `CompareDisk`. This menu option is a command.

17.2.1.2.2 Describe member `IgnoreHeaderCount` of `CompareDisk` The data file header contains a count of the number of machine words in each channel. This count is written at the time the node creating the file is deleted. If ObjectProDSP exits abnormally then these counts may never be set and one will not be able to read any of the data in the file. This option causes these counts to be ignored. The result is that data will be read until the physical end of file. This may result in samples of all 0 being read at the end of the file that were never written to it.

The commands in this menu are:

- `desc IgnoreHeaderCount`: Describe selected member of `CompareDisk`. This menu option is a command.

17.2.2 Options for `InputNode`

The commands in this menu are:

- `help`: Explain the use of `InputNode`. This menu option is a command.
- `param`: Parameters of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.1 on page 115.
- `members`: Members of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.2 on page 116. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.3 on page 117.
- `create default`: Create instance of `InputNode` with default parameters. This menu option is a command.
- `create`: Create instance of `InputNode`. This menu option is a command.

17.2.2.1 Menu of Parameters for `InputNode` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `InputNode` object. This menu option is a command.

  `FileName` specifies the name of the disk file to be read. If no file name is specified (default 0) then the node name will be used.
- `Flags`: Parameter `Flags` of `InputNode` object. This menu option is a command.

  If `Flags` & 4 is set the sample rate is forced to 1, overwriting the default values.
- `DeltaOut`: Parameter `DeltaOut` of `InputNode` object. This menu option is a command.

  `DeltaOut` is the minimum output size. The node is not scheduled until space for `DeltaOut` words is available in the output buffer. It will only write multiples of `DeltaOut` samples.

17.2.2.2 Select a member of `InputNode` to describe

The commands in this menu are:

- `DisplayHeader`: Describe member `DisplayHeader` of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.2.1 on page 116.
- `IgnoreHeaderCount`: Describe member `IgnoreHeaderCount` of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.2.2 on page 116.

17.2.2.2.1 Describe member `DisplayHeader` of `InputNode`

`DisplayHeader` displays the parameters read from file `FileName`. These include the original node name that generated the file, the caption for this node, the number of input channels, and the number of scalar elements in a sample. The arithmetic type is also shown. The output channels and sample size for this node are determined by these values. If the data in the file is in a different arithmetic format than that currently in use, the file data will be converted.

The commands in this menu are:

- `desc DisplayHeader`: Describe selected member of `InputNode`. This menu option is a command.

17.2.2.2.2 Describe member `IgnoreHeaderCount` of `InputNode`

The data file header contains a count of the number of machine words in each channel. This count is written at the time the node creating the file is deleted. If ObjectProDSP exits abnormally then these counts may never be set and one will not be able to read any of the data in the file. This option causes these counts to be ignored. The result is that data will be read until the physical end of file. This may result in samples of all 0 being read at the end of the file that were never written to it.

The commands in this menu are:

- `desc IgnoreHeaderCount`: Describe selected member of `InputNode`. This menu option is a command.

17.2.2.3 Select an Instance of `InputNode`

The commands in this menu are:

- *instance of this class*: Select this instance of `InputNode`. This menu option invokes the menu defined in Section 17.2.2.3.1 on page 117.

17.2.2.3.1 Operations on an instance of object `InputNode`

The commands in this menu are:

- `desc`: Describe this instance of `InputNode`. This menu option is a command.
- `param`: Describe parameters of this `InputNode`. This menu option invokes the menu defined in Section 17.2.2.3.2 on page 117.
- `exec`: Select a member of `InputNode` to execute. This menu option invokes the menu defined in Section 17.2.2.3.3 on page 118. See , Section 22.24 on page 166 for additional base class member functions in this menu.
- `delete`: Delete this `InputNode`. This menu option is a command.

17.2.2.3.2 Menu of Parameters for `InputNode` Object Instance

The commands in this menu are:

- `FileName`: Parameter `FileName` of `InputNode` object instance. This menu option is a command.

  `FileName` specifies the name of the disk file to be read. If no file name is specified (default 0) then the node name will be used.

- `Flags`: Parameter `Flags` of `InputNode` object instance. This menu option is a command.

  If `Flags` & 4 is set the sample rate is forced to 1, overwriting the default values.

- `DeltaOut`: Parameter `DeltaOut` of `InputNode` object instance. This menu option is a command.

  `DeltaOut` is the minimum output size. The node is not scheduled until space for `DeltaOut` words is available in the output buffer. It will only write multiples of `DeltaOut` samples.

17.2.2.3.3 Select a member of this `InputNode` to execute The commands in this menu are:

- `DisplayHeader`: Execute selected member of `InputNode`. This menu option is a command.
- `IgnoreHeaderCount`: Execute selected member of `InputNode`. This menu option is a command.

17.2.3 Options for `InputWord`

The commands in this menu are:

- `help`: Explain the use of `InputWord`. This menu option is a command.
- `param`: Parameters of `InputWord`. This menu option invokes the menu defined in Section 17.2.3.1 on page 119.
- `members`: Members of `InputWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `InputWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `InputWord` with default parameters. This menu option is a command.
- `create`: Create instance of `InputWord`. This menu option is a command.

17.2.3.1 Menu of Parameters for `InputWord` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `InputWord` object. This menu option is a command.

  `FileName` is the binary input file to read. If no default(0) is given the node name will be used.

- `FormatIn`: Parameter `FormatIn` of `InputWord` object. This menu option is a command.

  `FormatIn` is the binary input format. The options are: `MachWord` (0), `int8` (1), `int16` (2), `int32` (3), `float` (4), `double` (5). Integer words are written in two's compliment format. Integer input values (`IntegerMachWord`) are treated as signed. If overflow occurs the data is hard limited and a warning is given. The input stream can have any value for sample size, (`ElementSize`)

- `IntegerOut`: Parameter `IntegerOut` of `InputWord` object. This menu option is a command.

  If `IntegerOut` is nonzero the output stream is written as `IntegerMachWord`. Otherwise it is written as `MachWord`. If overflow occurs the data is hard limited and a warning is given.

- `InitialSkip`: Parameter `InitialSkip` of `InputWord` object. This menu option is a command.

  The first `InitialSkip` bytes of the file are ignored.

- `ElementSize`: Parameter `ElementSize` of `InputWord` object. This menu option is a command.

  `ElementSize` is the number of words per sample in the input file.

- `BlockSize`: Parameter `BlockSize` of `InputWord` object. This menu option is a command.

  `BlockSize` is the number of samples per block in the input file.

17.2.4 Options for `OutputNode`

The commands in this menu are:

- `help`: Explain the use of `OutputNode`. This menu option is a command.
- `param`: Parameters of `OutputNode`. This menu option invokes the menu defined in Section 17.2.4.1 on page 120.
- `members`: Members of `OutputNode`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `OutputNode`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `OutputNode` with default parameters. This menu option is a command.
- `create`: Create instance of `OutputNode`. This menu option is a command.

17.2.4.1 Menu of Parameters for `OutputNode` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `OutputNode` object. This menu option is a command.

  `FileName` specifies the name of the disk file to be created. If no caption is specified (default 0) then the node name will be used.

- `Flags`: Parameter `Flags` of `OutputNode` object. This menu option is a command.

  if bit 1 of `Flags` is set then an existing file with the same name is overwritten without comment. If the overwrite option is not set and a file with the same name exists this node will not be initialized properly in non interactive mode and network execution will fail. In interactive mode, you will be asked to supply a new name. if bit 8 is set (Flags&8) is true then the data will be converted from whatever format the input channel is to 32 bit integer data and written in that format. The data is hard limited.

- `Channels`: Parameter `Channels` of `OutputNode` object. This menu option is a command.

  `OutputNode` writes `Channels` of data to a disk file.

- `FileBlockSize`: Parameter `FileBlockSize` of `OutputNode` object. This menu option is a command.

  `FileBlockSize` determines the number of consecutive samples written to each channel. Making this larger reduces the number of disk accesses at the cost of larger memory buffers.

- `Caption`: Parameter `Caption` of `OutputNode` object. This menu option is a command.

  `Caption` is a one line description of the contents of the file. It is displayed by executing a member function of an input node that reads this file.

- `DeltaIn`: Parameter `DeltaIn` of `OutputNode` object. This menu option is a command.

  The node will read chunks of multiples of `DeltaIn` samples. Note for complex data there are two words in each sample. The node will not be executed unless `DeltaIn` input samples are available.

17.2.5 Options for `OutputWord`

The commands in this menu are:

- `help`: Explain the use of `OutputWord`. This menu option is a command.
- `param`: Parameters of `OutputWord`. This menu option invokes the menu defined in Section 17.2.5.1 on page 122.
- `members`: Members of `OutputWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete

desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.

- `instance`: Describe or delete an instance of `OutputWord`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `OutputWord` with default parameters. This menu option is a command.
- `create`: Create instance of `OutputWord`. This menu option is a command.

17.2.5.1 Menu of Parameters for `OutputWord` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `OutputWord` object. This menu option is a command.

  `FileName` is the file to be created. If no default(0) is given the node name will be used.

- `FormatOut`: Parameter `FormatOut` of `OutputWord` object. This menu option is a command.

  `FormatOut` is the binary output format. The options are: `MachWord` (0), `int8` (1), `int16` (2), `int32` (3), `float` (4), `double` (5). Integer words are written in two's compliment format. Integer input values (`IntegerMachWord`) are treated as signed. If overflow occurs the data is hard limited and a warning is given. The input stream can have any value for sample size, (`ElementSize`)

17.2.6 Options for `VoiceNode`

The commands in this menu are:

- `help`: Explain the use of `VoiceNode`. This menu option is a command.

- `param`: Parameters of `VoiceNode`. This menu option invokes the menu defined in Section 17.2.6.1 on page 123.
- `members`: Members of `VoiceNode`. This menu option invokes the menu defined in Section 17.2.6.2 on page 123. See Section 22.23 on page 164 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `VoiceNode`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `VoiceNode` with default parameters. This menu option is a command.
- `create`: Create instance of `VoiceNode`. This menu option is a command.

17.2.6.1 Menu of Parameters for `VoiceNode` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `VoiceNode` object. This menu option is a command.

  `FileName` must be the name of an existing file in the Creative Voice File Format. Only uncompressed files of block type 1 (New Voice Block) are supported.
- `NoHeader`: Parameter `NoHeader` of `VoiceNode` object. This menu option is a command.

  `NoHeader` if set to 1 indicates the file does not contain a header. `VoiceStripOut` writes files in Creative Voice format files with no header.

17.2.6.2 Select a member of `VoiceNode` to describe The commands in this menu are:

- `DisplayHeader`: Describe member `DisplayHeader` of `VoiceNode`. This menu option invokes the menu defined in Section 17.2.6.2.1 on page 124.

17.2.6.2.1 Describe member `DisplayHeader` of `VoiceNode` `DisplayHeader` displays the information in the file header.

The commands in this menu are:

- `desc DisplayHeader`: Describe selected member of `VoiceNode`. This menu option is a command.

17.2.7 Options for `VoiceStripOut`

The commands in this menu are:

- `help`: Explain the use of `VoiceStripOut`. This menu option is a command.
- `param`: Parameters of `VoiceStripOut`. This menu option invokes the menu defined in Section 17.2.7.1 on page 124.
- `members`: Members of `VoiceStripOut`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `members`. See Section 22.19 on page 160 for additional base class member functions in this menu.
- `instance`: Describe or delete an instance of `VoiceStripOut`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `VoiceStripOut` with default parameters. This menu option is a command.
- `create`: Create instance of `VoiceStripOut`. This menu option is a command.

17.2.7.1 Menu of Parameters for `VoiceStripOut` Object The commands in this menu are:

- `FileName`: Parameter `FileName` of `VoiceStripOut` object. This menu option is a command.

`FileName` is the file to be created. If no default(0) is given the node name will be used.

18 Networks

This menu describes classes for networks and their controllers. It describes classes for building multiple network systems.

The commands in this menu are:

- `CircBufDes`: Circular buffer descriptor. This menu option invokes the menu defined in Section 18.1 on page 125.
- `DataFlow`: Data flow based network control and scheduling. This menu option invokes the menu defined in Section 18.2 on page 129.
- `Network`: Data flow network objects. This menu option invokes the menu defined in Section 18.3 on page 132.

18.1 Options for `CircBufDes`

The commands in this menu are:

- `help`: Explain the use of `CircBufDes`. This menu option is a command.
- `param`: Parameters of `CircBufDes`. This menu option invokes the menu defined in Section 18.1.1 on page 126.
- `variables`: Changeable variables of `CircBufDes`. This menu option invokes the menu defined in Section 18.1.2 on page 127.
- `members`: Members of `CircBufDes`. This menu option invokes the menu defined in Section 18.1.3 on page 128.
- `instance`: Describe or delete an instance of `CircBufDes`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.

- `create default`: Create instance of `CircBufDes` with default parameters. This menu option is a command.
- `create`: Create instance of `CircBufDes`. This menu option is a command.

18.1.1 Menu of Parameters for `CircBufDes` Object

The commands in this menu are:

- `Size`: Parameter `Size` of `CircBufDes` object. This menu option is a command.

  `Size` determines the buffer size for interactive execution. The larger the buffer is the more efficiently the nodes can execute and the greater the delay that is possible in the network. If the buffers are too small the network may lock up without processing all input data.

- `TargetSize`: Parameter `TargetSize` of `CircBufDes` object. This menu option is a command.

  `TargetSize` is the desired size of the buffer used in code prepared for execution on the target processor. An analysis will determine if this size is adequate and if it is larger than will be of benefit. The size will be optimized based on this target size as an approximate goal.

- `TargetSizeGoal`: Parameter `TargetSizeGoal` of `CircBufDes` object. This menu option is a command.

  `TargetSizeGoal` determines whether the buffer will be made larger or smaller then the selected size when the selected size is not optimal. If `TargetSizeGoal` is 0 that space will be minimized. If `TargetSizeGoal` is 1 then execution overhead will be minimized.

- `TargetControlGoal`: Parameter `TargetControlGoal` of `CircBufDes` object. This menu option is a command.

  `TargetControlGoal` determines how execution is controlled. If it is 0 then the buffer size is fixed and the amount of data available in the buffer is computed before each execution step. If `TargetControlGoal` is 1 then each node is executed according to a fixed predetermined schedule and the buffer size is optimized to this schedule. Nodes with feedback or that require excessively large buffers are defaulted to execute without fixed sequences. Fixed sequence execution provides more efficient execution at the expense of larger buffer requirements.

- `MaxTargetSize`: Parameter `MaxTargetSize` of `CircBufDes` object. This menu option is a command.

  `MaxTargetSize` specifies the maximum size allowed for any single buffer. It can force the use of a non fixed sequence scheduler or even result in an error message if the network cannot run without deadlock with this size buffer.

- `MinTargetSize`: Parameter `MinTargetSize` of `CircBufDes` object. This menu option is a command.

  `MinTargetSize` specifies the minimum size allowed for any single buffer. Setting this to a larger value can improve execution efficiency at the cost of more memory.

18.1.2 Menu of Variables for CircBufDes Object

The commands in this menu are:

- `TargetSize`: Changeable variable `TargetSize` of `CircBufDes` object. This menu option is a command.

  `TargetSize` is the desired size of the buffer used in code prepared for execution on the target processor. An analysis will determine if this size is adequate and if it is larger than will be of benefit. The size will be optimized based on this target size as an approximate goal.

- `TargetSizeGoal`: Changeable variable `TargetSizeGoal` of `CircBufDes` object. This menu option is a command.

  `TargetSizeGoal` determines whether the buffer will be made larger or smaller then the selected size when the selected size is not optimal. If `TargetSizeGoal` is 0 that space will be minimized. If `TargetSizeGoal` is 1 then execution overhead will be minimized.

- `TargetControlGoal`: Changeable variable `TargetControlGoal` of `CircBufDes` object. This menu option is a command.

  `TargetControlGoal` determines how execution is controlled. If it is 0 then the buffer size is fixed and the amount of data available in the buffer is computed before each execution step. If `TargetControlGoal` is 1 then each node is executed according to a fixed predetermined schedule and the buffer size is optimized to this schedule. Nodes with feedback or that require excessively large buffers are defaulted to execute without fixed sequences. Fixed sequence execution provides more efficient execution at the expense of larger buffer requirements.

- `MaxTargetSize`: Changeable variable `MaxTargetSize` of `CircBufDes` object. This menu option is a command.

  `MaxTargetSize` specifies the maximum size allowed for any single buffer. It can force the use of a non fixed sequence scheduler or even result in an error message if the network cannot run without deadlock with this size buffer.

- `MinTargetSize`: Changeable variable `MinTargetSize` of `CircBufDes` object. This menu option is a command.

  `MinTargetSize` specifies the minimum size allowed for any single buffer. Setting this to a larger value can improve execution efficiency at the cost of more memory.

18.1.3 Select a member of `CircBufDes` to describe

The commands in this menu are:

- `AssignToEdit`: Describe member `AssignToEdit` of `CircBufDes`. This menu option invokes the menu defined in Section 18.1.3.1 on page 128.

18.1.3.1 Describe member `AssignToEdit` of `CircBufDes`

`AssignToEdit` makes this the descriptor for the network currently being edited. If a previous descriptor was assigned it will be overwritten.

The commands in this menu are:

- `desc AssignToEdit`: Describe selected member of `CircBufDes`. This menu option is a command.

18.2 Options for `DataFlow`

The commands in this menu are:

- `help`: Explain the use of `DataFlow`. This menu option is a command.
- `param`: Parameters of `DataFlow`. This menu option invokes the menu defined in Section 18.2.1 on page 129.
- `members`: Members of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2 on page 129.
- `instance`: Describe or delete an instance of `DataFlow`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create`: Create instance of `DataFlow`. This menu option is a command.

18.2.1 Menu of Parameters for `DataFlow` Object

The commands in this menu are:

- `TheNet`: Parameter `TheNet` of `DataFlow` object. This menu option is a command.

  `TheNet` is the `Network` to be controlled.

18.2.2 Select a member of `DataFlow` to describe

The commands in this menu are:

- `GraphDisplay`: Describe member `GraphDisplay` of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2.1 on page 130.

- `Execute`: Describe member `Execute` of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2.2 on page 130.
- `AssignBuffers`: Describe member `AssignBuffers` of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2.3 on page 131.
- `ClearBuffers`: Describe member `ClearBuffers` of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2.4 on page 132.
- `ClearNetwork`: Describe member `ClearNetwork` of `DataFlow`. This menu option invokes the menu defined in Section 18.2.2.5 on page 132.

18.2.2.1 Describe member `GraphDisplay` of `DataFlow`

Member function `GraphDisplay` displays the network topology and timing.

The commands in this menu are:

- `desc GraphDisplay`: Describe selected member of `DataFlow`. This menu option is a command.
- `param GraphDisplay`: Describe the parameters of member `GraphDisplay`. This menu option invokes the menu defined in Section 18.2.2.1.1 on page 130.

18.2.2.1.1 Select parameter of `DataFlow` member `GraphDisplay` to describe

The commands in this menu are:

- `Option`: Select this parameter of `DataFlow` member `GraphDisplay` to describe. This menu option is a command.

18.2.2.2 Describe member `Execute` of `DataFlow`

Member function `Execute` executes the network being controlled. It causes the signal generator (or first node in the network) to produce a specified number of input samples. Each node is executed for as many iterations as possible given the available input data and output buffer space. Execution halts when no node generates any new samples after a complete pass throughout the network.

The commands in this menu are:

- `desc Execute`: Describe selected member of `DataFlow`. This menu option is a command.
- `param Execute`: Describe the parameters of member `Execute`. This menu option invokes the menu defined in Section 18.2.2.2.1 on page 131.

18.2.2.2.1 Select parameter of `DataFlow` member `Execute` to describe The commands in this menu are:

- `InputSamples`: Select this parameter of `DataFlow` member `Execute` to describe. This menu option is a command.

18.2.2.3 Describe member `AssignBuffers` of `DataFlow` Member function `AssignBuffers` buffers to a completely defined data flow network. The network is first checked for completeness. Buffers will not be assigned if the network fails this test. The single parameter of this function specifies the buffer characteristics.

The commands in this menu are:

- `desc AssignBuffers`: Describe selected member of `DataFlow`. This menu option is a command.
- `param AssignBuffers`: Describe the parameters of member `AssignBuffers`. This menu option invokes the menu defined in Section 18.2.2.3.1 on page 131.

18.2.2.3.1 Select parameter of `DataFlow` member `AssignBuffers` to describe The commands in this menu are:

- `Descriptor`: Select this parameter of `DataFlow` member `AssignBuffers` to describe. This menu option is a command.

18.2.2.4 Describe member `ClearBuffers` of `DataFlow` You can not change the topology of a network while buffers are assigned. Member function `ClearBuffers` removes all buffers so that the network can be edited or different buffers assigned.

The commands in this menu are:

- `desc ClearBuffers`: Describe selected member of `DataFlow`. This menu option is a command.

18.2.2.5 Describe member `ClearNetwork` of `DataFlow` Member function `ClearNetwork` removes all links in the network being controlled. The nodes freed in this way can then be used in a different network.

The commands in this menu are:

- `desc ClearNetwork`: Describe selected member of `DataFlow`. This menu option is a command.

18.3 Options for `Network`

The commands in this menu are:

- `help`: Explain the use of `Network`. This menu option is a command.
- `members`: Members of `Network`. This menu option invokes the menu defined in Section 18.3.1 on page 133.
- `instance`: Describe or delete an instance of `Network`. See Section 13.1 on page 56 for an example of the menu tree from this command and[3] for a complete desciption of all options for this object `instance`.
- `create default`: Create instance of `Network` with default parameters. This menu option is a command.
- `create`: Create instance of `Network`. This menu option is a command.

18.3.1 Select a member of `Network` to describe

The commands in this menu are:

- `GraphDisplay`: Describe member `GraphDisplay` of `Network`. This menu option invokes the menu defined in Section 18.3.1.1 on page 134.
- `Execute`: Describe member `Execute` of `Network`. This menu option invokes the menu defined in Section 18.3.1.2 on page 135.
- `Raise`: Describe member `Raise` of `Network`. This menu option invokes the menu defined in Section 18.3.1.3 on page 135.
- `ReplaceNode`: Describe member `ReplaceNode` of `Network`. This menu option invokes the menu defined in Section 18.3.1.4 on page 136.
- `MakeTarget`: Describe member `MakeTarget` of `Network`. This menu option invokes the menu defined in Section 18.3.1.5 on page 136.
- `MakeValidate`: Describe member `MakeValidate` of `Network`. This menu option invokes the menu defined in Section 18.3.1.6 on page 137.
- `TargetValidate`: Describe member `TargetValidate` of `Network`. This menu option invokes the menu defined in Section 18.3.1.7 on page 138.
- `SetTimingExact`: Describe member `SetTimingExact` of `Network`. This menu option invokes the menu defined in Section 18.3.1.8 on page 139.
- `ReplaceWithOutput`: Describe member `ReplaceWithOutput` of `Network`. This menu option invokes the menu defined in Section 18.3.1.9 on page 140.
- `ReplaceWithCompare`: Describe member `ReplaceWithCompare` of `Network`. This menu option invokes the menu defined in Section 18.3.1.10 on page 140.
- `operator+`: Describe member `operator+` of `Network`. This menu option invokes the menu defined in Section 18.3.1.11 on page 141.
- `operator>>`: Describe member `operator>>` of `Network`. This menu option invokes the menu defined in Section 18.3.1.12 on page 141.

- `GraphDisplayWindow`: Describe member `GraphDisplayWindow` of `Network`. This menu option invokes the menu defined in Section 18.3.1.13 on page 142.
- `DisplayNames`: Describe member `DisplayNames` of `Network`. This menu option invokes the menu defined in Section 18.3.1.14 on page 143.
- `SetBufferDescriptor`: Describe member `SetBufferDescriptor` of `Network`. This menu option invokes the menu defined in Section 18.3.1.15 on page 143.
- `AssociateNode`: Describe member `AssociateNode` of `Network`. This menu option invokes the menu defined in Section 18.3.1.16 on page 143.
- `Link`: Describe member `Link` of `Network`. This menu option invokes the menu defined in Section 18.3.1.17 on page 144.
- `SelfLink`: Describe member `SelfLink` of `Network`. This menu option invokes the menu defined in Section 18.3.1.18 on page 145.
- `AssignBuffers`: Describe member `AssignBuffers` of `Network`. This menu option invokes the menu defined in Section 18.3.1.19 on page 145.
- `GetBufferDescriptor`: Describe member `GetBufferDescriptor` of `Network`. This menu option invokes the menu defined in Section 18.3.1.20 on page 146.
- `ClearBuffers`: Describe member `ClearBuffers` of `Network`. This menu option invokes the menu defined in Section 18.3.1.21 on page 146.
- `GetNetController`: Describe member `GetNetController` of `Network`. This menu option invokes the menu defined in Section 18.3.1.22 on page 146.
- `ClearNetwork`: Describe member `ClearNetwork` of `Network`. This menu option invokes the menu defined in Section 18.3.1.23 on page 147.

18.3.1.1 Describe member `GraphDisplay` of `Network` Member function `GraphDisplay` displays network topology.

The commands in this menu are:

- `desc GraphDisplay`: Describe selected member of `Network`. This menu option is a command.

18.3.1.2 Describe member `Execute` of `Network`

Member function `Execute` executes this network. It causes the first node in the network to produce a specified number of input blocks. Each node is executed for as many iterations as possible given the available input data and output buffer space. Execution halts when no node generates any new samples after a complete pass throughout the network.

The commands in this menu are:

- `desc Execute`: Describe selected member of `Network`. This menu option is a command.
- `param Execute`: Describe the parameters of member `Execute`. This menu option invokes the menu defined in Section 18.3.1.2.1 on page 135.

18.3.1.2.1 Select parameter of `Network` member `Execute` to describe

The commands in this menu are:

- `InputSamples`: Select this parameter of `Network` member `Execute` to describe. This menu option is a command.

18.3.1.3 Describe member `Raise` of `Network`

`Raise` will cause a window displaying this network to be raised to the top level over any overlapping windows.

The commands in this menu are:

- `desc Raise`: Describe selected member of `Network`. This menu option is a command.

18.3.1.4 Describe member `ReplaceNode` of `Network`

`ReplaceNode` will substitute node `Replacement` for node `ToReplace` in the network. The nodes must have the same number of input and output channels and be compatible in all other respects. If an error occurs the original node will remain in the network.

The commands in this menu are:

- `desc ReplaceNode`: Describe selected member of `Network`. This menu option is a command.
- `param ReplaceNode`: Describe the parameters of member `ReplaceNode`. This menu option invokes the menu defined in Section 18.3.1.4.1 on page 136.

18.3.1.4.1 Select parameter of `Network` member `ReplaceNode` to describe

The commands in this menu are:

- `ToReplace`: Select this parameter of `Network` member `ReplaceNode` to describe. This menu option is a command.
- `Replacement`: Select this parameter of `Network` member `ReplaceNode` to describe. This menu option is a command.

18.3.1.5 Describe member `MakeTarget` of `Network`

Member function `MakeTarget` will create source and executable code for this network for a supported target. See the description of parameter `Target` for a list of the available targets. See parameter `Directory` for a description of the files created.

The commands in this menu are:

- `desc MakeTarget`: Describe selected member of `Network`. This menu option is a command.
- `param MakeTarget`: Describe the parameters of member `MakeTarget`. This menu option invokes the menu defined in Section 18.3.1.5.1 on page 137.

18.3.1.5.1 Select parameter of `Network` member `MakeTarget` to describe

The commands in this menu are:

- `Target`: Select this parameter of `Network` member `MakeTarget` to describe. This menu option is a command.
- `Create`: Select this parameter of `Network` member `MakeTarget` to describe. This menu option is a command.
- `Directory`: Select this parameter of `Network` member `MakeTarget` to describe. This menu option is a command.

18.3.1.6 Describe member `MakeValidate` of `Network`

Member function `MakeValidate` first replaces each display output node in this network (such as plotting or listing nodes) with an `OutputNode`. This network is then saved to directory `DirName`. Next the same nodes are replaced with a `CompareDisk` node. This new network is written to the same directory under a different name. The first network saved state will include a statement to execute for `ExecuteCount` + `ExtraCountCreator` blocks of input. The second network will execute for `ExecuteCount` blocks. These networks are used to generate baseline regression test data and to run tests against this data.

The commands in this menu are:

- `desc MakeValidate`: Describe selected member of `Network`. This menu option is a command.
- `param MakeValidate`: Describe the parameters of member `MakeValidate`. This menu option invokes the menu defined in Section 18.3.1.6.1 on page 137.

18.3.1.6.1 Select parameter of `Network` member `MakeValidate` to describe

The commands in this menu are:

- `DirName`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.

- `ExecuteCount`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.
- `ExtraCountCreator`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.
- `MaxReport`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.
- `Tolerance`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.
- `errorFile`: Select this parameter of `Network` member `MakeValidate` to describe. This menu option is a command.

18.3.1.7 Describe member `TargetValidate` of `Network`

Member function `TargetValidate` first replaces each display output node in this network (such as plotting or listing nodes) with an `OutputNode`. This network is then used to generate a target system in directory `DirName` /create'. Next the same nodes are replaced with a `CompareDisk` node. The code for this network is written to the directory `DirName` /test. Shell scripts will be written to directory `DirName` to execute the networks for `ExecuteCount` blocks (test network) and (`ExecuteCount` + `ExtraCountCreator`) blocks (test data creation network). `DirName` will contain all test data and error files and network state descriptions.

The commands in this menu are:

- `desc TargetValidate`: Describe selected member of `Network`. This menu option is a command.
- `param TargetValidate`: Describe the parameters of member `TargetValidate`. This menu option invokes the menu defined in Section 18.3.1.7.1 on page 138.

18.3.1.7.1 Select parameter of `Network` member `TargetValidate` to describe

The commands in this menu are:

- `Target`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `Create`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `DirName`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `ExecuteCount`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `ExtraCountCreator`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `MaxReport`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `Tolerance`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.
- `ErrorFile`: Select this parameter of `Network` member `TargetValidate` to describe. This menu option is a command.

18.3.1.8 Describe member `SetTimingExact` of `Network` If the timing analysis cannot resolve a network it may still execute correctly. If `Exact` is one no attempt will be made to execute the network. Instead an error will be generated. This is usually set for for validation tests to make sure that timing analysis errors are not overlooked.

The commands in this menu are:

- `desc SetTimingExact`: Describe selected member of `Network`. This menu option is a command.
- `param SetTimingExact`: Describe the parameters of member `SetTimingExact`. This menu option invokes the menu defined in Section 18.3.1.8.1 on page 140.

18.3.1.8.1 Select parameter of `Network` member `SetTimingExact` to describe

The commands in this menu are:

- `Exact`: Select this parameter of `Network` member `SetTimingExact` to describe. This menu option is a command.

18.3.1.9 Describe member `ReplaceWithOutput` of `Network`

Member function `ReplaceWithOutput` replaces all plot and listing nodes with a new output node with a name derived from the node it is replacing. The file name is the same as the node name. This is used to create regression tests. First `ReplaceWithOutput` creates a network to generate test data. Then `ReplaceWithCompare` creates a network for running a regression test against the data. All three networks should be saved in separate state files.

The commands in this menu are:

- `desc ReplaceWithOutput`: Describe selected member of `Network`. This menu option is a command.

18.3.1.10 Describe member `ReplaceWithCompare` of `Network`

Member function `ReplaceWithCompare` replaces each `OutputNode` in a network with a `CompareDisk` node. If an `OutputNode` has more than one input channel the operation will fail. This is useful in converting a network used to generate a regression test case to a network for running the regression test. The new node name is created from the node replaced. The file name is the output file written by the node being replaced. The other parameters are set to be the same as the corresponding parameters of this function.

The commands in this menu are:

- `desc ReplaceWithCompare`: Describe selected member of `Network`. This menu option is a command.
- `param ReplaceWithCompare`: Describe the parameters of member `ReplaceWithCompare`. This menu option invokes the menu defined in Section 18.3.1.10.1 on page 141.

18.3.1.10.1 Select parameter of `Network` member `ReplaceWithCompare` to describe

The commands in this menu are:

- `MaxReport`: Select this parameter of `Network` member `ReplaceWithCompare` to describe. This menu option is a command.
- `Tolerance`: Select this parameter of `Network` member `ReplaceWithCompare` to describe. This menu option is a command.
- `ErrorFile`: Select this parameter of `Network` member `ReplaceWithCompare` to describe. This menu option is a command.

18.3.1.11 Describe member `operator+` of `Network`

The `+` operator appends its right operand (a signal generation node) `TheNode` to its left operand (a data flow `Network`).

The commands in this menu are:

- `desc operator+`: Describe selected member of `Network`. This menu option is a command.
- `param operator+`: Describe the parameters of member `operator+`. This menu option invokes the menu defined in Section 18.3.1.11.1 on page 141.

18.3.1.11.1 Select parameter of `Network` member `operator+` to describe

The commands in this menu are:

- `TheNode`: Select this parameter of `Network` member `operator+` to describe. This menu option is a command.

18.3.1.12 Describe member `operator>>` of `Network`

The >> operator appends its right operand (a processing or signal generation node) `TheNode` to its left operand (a `DataFlow` Network).

The commands in this menu are:

- `desc operator>>`: Describe selected member of `Network`. This menu option is a command.
- `param operator>>`: Describe the parameters of member `operator>>`. This menu option invokes the menu defined in Section 18.3.1.12.1 on page 142.

18.3.1.12.1 Select parameter of `Network` member `operator>>` to describe The commands in this menu are:

- `TheNode`: Select this parameter of `Network` member `operator>>` to describe. This menu option is a command.

18.3.1.13 Describe member `GraphDisplayWindow` of `Network` Member function `GraphDisplay` displays the network topology. In a window of up to `Width` x `Height` pixels. The window may start out smaller and grow larger as nodes are added to it but it will not exceed these dimensions. (If needed a vertical scrollbar will be added to the window.)

The commands in this menu are:

- `desc GraphDisplayWindow`: Describe selected member of `Network`. This menu option is a command.
- `param GraphDisplayWindow`: Describe the parameters of member `GraphDisplayWindow`. This menu option invokes the menu defined in Section 18.3.1.13.1 on page 142.

18.3.1.13.1 Select parameter of `Network` member `GraphDisplayWindow` to describe The commands in this menu are:

- `Width`: Select this parameter of `Network` member `GraphDisplayWindow` to describe. This menu option is a command.
- `Height`: Select this parameter of `Network` member `GraphDisplayWindow` to describe. This menu option is a command.

18.3.1.14 Describe member `DisplayNames` of `Network`

Member function `DisplayNames` displays the names of the controller and buffer descriptor for this node in the help window.

The commands in this menu are:

- `desc DisplayNames`: Describe selected member of `Network`. This menu option is a command.

18.3.1.15 Describe member `SetBufferDescriptor` of `Network`

`SetBufferDescriptor` assigns descriptor `Descriptor` to this network. The network need not be complete. No buffers are allocated.

The commands in this menu are:

- `desc SetBufferDescriptor`: Describe selected member of `Network`. This menu option is a command.
- `param SetBufferDescriptor`: Describe the parameters of member `SetBufferDescriptor`. This menu option invokes the menu defined in Section 18.3.1.15.1 on page 143.

18.3.1.15.1 Select parameter of `Network` member `SetBufferDescriptor` to describe

The commands in this menu are:

- `Descriptor`: Select this parameter of `Network` member `SetBufferDescriptor` to describe. This menu option is a command.

18.3.1.16 Describe member `AssociateNode` of `Network`

`AssociateNode` associates node `TheNode` with this network. It does not link the node into the network. Its only effect is to have the node displayed in the window in which the network appears.

The commands in this menu are:

- `desc AssociateNode`: Describe selected member of `Network`. This menu option is a command.
- `param AssociateNode`: Describe the parameters of member `AssociateNode`. This menu option invokes the menu defined in Section 18.3.1.16.1 on page 144.

18.3.1.16.1 Select parameter of `Network` member `AssociateNode` to describe

The commands in this menu are:

- `TheNode`: Select this parameter of `Network` member `AssociateNode` to describe. This menu option is a command.

18.3.1.17 Describe member `Link` of `Network`

In building a data flow topology network the connection operator >> always links the output of the last node accessed (from the network to the left of >>) to the node on the right side of >>. For simple linear networks this is adequate. For nodes with more than one output one must specify when to use channels other than 0 using `Link`. `Link` causes the next link from the network to begin at the specified `TheNode` and `OutChannel`.

The commands in this menu are:

- `desc Link`: Describe selected member of `Network`. This menu option is a command.
- `param Link`: Describe the parameters of member `Link`. This menu option invokes the menu defined in Section 18.3.1.17.1 on page 144.

18.3.1.17.1 Select parameter of `Network` member `Link` to describe

The commands in this menu are:

- `TheNode`: Select this parameter of `Network` member `Link` to describe. This menu option is a command.
- `OutChannel`: Select this parameter of `Network` member `Link` to describe. This menu option is a command.

18.3.1.18 Describe member `SelfLink` of `Network` `SelfLink` establishes a feedback link in a data flow network. Input parameters include the source node and its output channel and the destination node and its input channel.

The commands in this menu are:

- `desc SelfLink`: Describe selected member of `Network`. This menu option is a command.
- `param SelfLink`: Describe the parameters of member `SelfLink`. This menu option invokes the menu defined in Section 18.3.1.18.1 on page 145.

18.3.1.18.1 Select parameter of `Network` member `SelfLink` to describe The commands in this menu are:

- `NodeOut`: Select this parameter of `Network` member `SelfLink` to describe. This menu option is a command.
- `NodeIn`: Select this parameter of `Network` member `SelfLink` to describe. This menu option is a command.
- `ChannelOut`: Select this parameter of `Network` member `SelfLink` to describe. This menu option is a command.
- `ChannelIn`: Select this parameter of `Network` member `SelfLink` to describe. This menu option is a command.

18.3.1.19 Describe member `AssignBuffers` of `Network` Member function `AssignBuffers` buffers to a complete network. Buffers will not be assigned if the network is not complete. `Descriptor` determines the buffer characteristics.

The commands in this menu are:

- `desc AssignBuffers`: Describe selected member of `Network`. This menu option is a command.

- `param AssignBuffers`: Describe the parameters of member `AssignBuffers`. This menu option invokes the menu defined in Section 18.3.1.19.1 on page 146.

18.3.1.19.1 Select parameter of `Network` member `AssignBuffers` to describe

The commands in this menu are:

- `Descriptor`: Select this parameter of `Network` member `AssignBuffers` to describe. This menu option is a command.

18.3.1.20 Describe member `GetBufferDescriptor` of `Network`

Member function `GetBufferDescriptor` returns the buffer descriptor associated with this network.

The commands in this menu are:

- `desc GetBufferDescriptor`: Describe selected member of `Network`. This menu option is a command.

18.3.1.21 Describe member `ClearBuffers` of `Network`

You can not change the topology of a network while buffers are assigned. Member function `ClearBuffers` removes all buffers so that the network can be edited or different buffers assigned.

The commands in this menu are:

- `desc ClearBuffers`: Describe selected member of `Network`. This menu option is a command.

18.3.1.22 Describe member `GetNetController` of `Network`

Member function `GetNetController` returns the network controller associated with this network.

The commands in this menu are:

- `desc GetNetController`: Describe selected member of `Network`. This menu option is a command.

18.3.1.23 Describe member `ClearNetwork` of `Network`

Member function `ClearNetwork` removes all links in the network being controlled. The nodes freed in this way can then be used in a different network.

The commands in this menu are:

- `desc ClearNetwork`: Describe selected member of `Network`. This menu option is a command.

19 Display simple variables

The `variables` menu displays the simple variables (integer and double precision floating point).

The commands in this menu are:

- `list int`: List integer variables. This menu option is a command.

  `list int` displays the names and values of integer variables.

- `list float`: List floating point variables. This menu option is a command.

  `list float` displays the names and values of floating point double precision variables.

- `list mach`: List MachWord variables. This menu option is a command.

  `list mach` displays the names and values of MachWord variables. MachWord is the type for single precision objects in the machine being simulated.

- `list accmach`: List AccMachWord variables. This menu option is a command.

`list accmach` displays the names and values of AccMachWord variables. AccMachWord is the type for double precision (typically in the accumulator) objects in the machine being simulated.

20 Read files

From the `setup` menu you may read and execute a ObjectProDSP state file created in a previous session or created manually. You may also read a plot file created in a previous session and control debugging options.

The commands in this menu are:

- `read state`: Read a program state file. This menu option is a command.

  `read state` reads a text file of ObjectProDSP language statements created either automatically or manually. The objects defined in the file are added to existing objects. Existing objects will not be overwritten. If an error occurs processing will stop and the error and line number where it occurs will be displayed.

- `read over state`: Read a program state file. This menu option is a command.

  `read over state` reads a text file of ObjectProDSP language statements created either automatically or manually. The objects defined in the file will be added to any objects already defined. If the file defines an existing object, it will be deleted and replaced by the object in the file. If an error occurs processing will stop and the error and line number where it occurs will be displayed.

- `read plot`: Read and display a saved plot file. This menu option is a command.

  `read plot` reads and displays a plot file that was previously saved.

- `debug`: control debugging options. This menu option invokes the menu defined in Section 20.1 on page 149.

20.1 Control debugging options

The `debug` menu allows you to control network tracing, reporting of overflows and check dynamic memory allocation integrity.

The commands in this menu are:

- `trace`: Turn on node tracing. This menu option is a command.

  `trace` causes the name of each node to be output to the log file `dsp.messages` just before it is executed. If you are debugging multiple nodes and the program crashes this will allow you to determine the node in which the problem occurs.

- `trace off`: Turn off node tracing. This menu option is a command.

  `trace off` disables the trace of node execution written to the log file `dsp.messages`.

- `heap ck`: Turn on heap integrity checking. This menu option is a command.

  `heap ck` causes a check to be made on the heap integrity before and after each node execution. This can be helpful in tracking errors related to dynamic memory allocation that may not cause a problem immediately. This check is not certain to find all problems with the heap but in practice it seems to detect most of them and produce some system error. Turning this on also turns on trace. This option can significantly slow program execution.

- `heap ck off`: Turn off heap checking. This menu option is a command.

  `heap ck off` turns off the heap integrity check. If you are turning this off you may also want to turn off tracing. Both these options are turned on by `heap ck` but this command only turns of heap checking.

- `over lim`: Control overflow reporting. This menu option is a command.

  The first time overflows occur in a node they are reported in the help information window. Later overflow are reported every time the total number of new overflows exceeds a user specified limit. `over lim` sets this limit. If trace is enabled then the count of overflows in each node are written to the log file `dsp.messages`.

21 Program State

In the `state` menu you can save the DSP++ program state to a disk file, set ObjectProDSP to automatically save the program state periodically and control and display the name of the file in which to save the state. You can also record and play back ObjectProDSP sessions.

The commands in this menu are:

- `save`: Save state. This menu option is a command.

  Save the state to the latest version of the input file name (or the name last set by the `set name` command). The default name is `dsppp`. If the file name has a `.` in it then multiple backup versions will not be generated and the state will be saved to `stat.lxxxxx` where `l` is a letter and `x` a digit. Use `save as` if you want to be prompted for a file name with no support for multiple versions.

- `save as`: Save state with prompt for file. This menu option is a command.

  Save the state to a file that you will be prompted for. If the file already exist you will be asked of you want to overwrite it.

- `auto save`: Save state periodically. This menu option is a command.

- `set name`: Display or change state file name and number of backups. This menu option is a command.

  `set name` displays the current base name for saving DSP++ objects and the number of backups saved. You can change either of these. If the file name name does not contain a `.`, the state will be saved in multiple versions called `NAME.0`, `NAME.1`, etc. The LOWEST numbered version is the most recent. Thus `NAME.0` will always be the current or last saved version. The default value for `NAME` is `dsppp`. If you specify a file name with a `.` the state will be saved to a unique name of the form `stat.XXXXX` where `XXXXX` is a string created by the system utility `mktemp`. The original file you specify as input will never be overwritten if it contains a `.`.

- `session`: Record and playback sessions. This menu option invokes the menu defined in Section 21.1 on page 151.
- `abort dsp`: Abort DSP execution. This menu option is a command.

  `abort dsp` aborts execution of the DSP process by generating a user error. If the DSP process is not executing, this command has no effect.
- `plot err`: display plot data error counts. This menu option is a command.

  `plot err` displays the count of non numeric values (NAN and infinity) in all plot data streams. This should not happen unless you are debugging a node that generates such values. There is only limited protection against this. Such values can cause the system to crash with a floating point exception.
- `freeze`: Freeze DSP execution. This menu option is a command.

  `freeze` stops DSP process execution to temporarily reduce system load.
- `thaw`: Thaw DSP execution. This menu option is a command.

  `thaw` resumes DSP process execution after a `freeze` .
- `exit`: Exit ObjectProDSP with optional state save. This menu option is a command.

  `exit` exits to the operating system. If you have not saved the state of since you last changed something, you will be asked if you wish to save the state. If the DSP process is hung you may need to do this twice to get information about its state.

21.1 Record or playback a session

The `session` menu allows you to record and play back entire ObjectProDSP sessions.

The commands in this menu are:

- `rec`: Begin recording all session inputs. This menu option is a command.

If you have previously been recoding and have not closed the file recording will continue in the same file. Otherwise you will be prompted for a file name.

- `rec off`: Stop recording. This menu option is a command.

  You can resume recording to the same file with this option.

- `rec close`: Stop recording and close the file. This menu option is a command.

  You can not resume recording to the same file after this selection.

- `play`: Play back a recorded session. This menu option is a command.

  To abort playback hold the control key (Ctrl) down and hit the `Delete` key with the main ObjectProDSP window selected.

- `pause`: Specify delay in seconds between commands. This menu option is a command.

  `pause` prompts you for a delay in seconds between session commands. This allows you to view recorded ObjectProDSP sessions as a tutorial movie.

22 Base class member functions

This section contains member functions of ObjectProDSP base classes that are not used directly. These member functions are shared by several derived classes and are referenced from those classes.

22.1 Describe member `DisplayInputTiming` of `DisplayNodeStr`

Member function `DisplayInputTiming` displays the timing of the selects input channel for this node.

The commands in this menu are:

- `desc DisplayInputTiming`: Describe selected member of `DisplayNodeStr`. This menu option is a command.
- `param DisplayInputTiming`: Describe the parameters of member `DisplayInputTiming`. This menu option invokes the menu defined in Section 22.1.1 on page 153.

22.1.1 Select parameter of `DisplayNodeStr` member `DisplayInputTiming` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `DisplayNodeStr` member `DisplayInputTiming` to describe. This menu option is a command.

22.2 Describe member `DisplayInputTiming` of `ProcessNodeStr`

Member function `DisplayInputTiming` displays the timing of the selected input channel for this node.

The commands in this menu are:

- `desc DisplayInputTiming`: Describe selected member of `ProcessNodeStr`. This menu option is a command.
- `param DisplayInputTiming`: Describe the parameters of member `DisplayInputTiming`. This menu option invokes the menu defined in Section 22.2.1 on page 153.

22.2.1 Select parameter of `ProcessNodeStr` member `DisplayInputTiming` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `ProcessNodeStr` member `DisplayInputTiming` to describe. This menu option is a command.

22.3 Select parameter of `DisplayNodeStr` member `LinkIn` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `DisplayNodeStr` member `LinkIn` to describe. This menu option is a command.

22.4 Describe member `DisplayOutputTiming` of `ProcessNodeStr`

Member function `DisplayOutputTiming` displays the timing of the selected output channel for this node.

The commands in this menu are:

- `desc DisplayOutputTiming`: Describe selected member of `ProcessNodeStr`. This menu option is a command.
- `param DisplayOutputTiming`: Describe the parameters of member `DisplayOutputTiming`. This menu option invokes the menu defined in Section 22.4.1 on page 154.

22.4.1 Select parameter of `ProcessNodeStr` member `DisplayOutputTiming` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `ProcessNodeStr` member `DisplayOutputTiming` to describe. This menu option is a command.

22.5 Describe member `DisplayOutputTiming` of `SignalStr`

Member function `DisplayOutputTiming` displays the timing of the selected output channel for this node.

The commands in this menu are:

- `desc DisplayOutputTiming`: Describe selected member of `SignalStr`. This menu option is a command.
- `param DisplayOutputTiming`: Describe the parameters of member `DisplayOutputTiming`. This menu option invokes the menu defined in Section 22.5.1 on page 155.

22.5.1 Select parameter of `SignalStr` member `DisplayOutputTiming` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `SignalStr` member `DisplayOutputTiming` to describe. This menu option is a command.

22.6 Describe member `Edit` of `DisplayNodeStr`

If this node is not linked in an existing network it will be added to the display of the network currently being edited. If there is no such network one will be created.

The commands in this menu are:

- `desc Edit`: Describe selected member of `DisplayNodeStr`. This menu option is a command.

22.7 Describe member `Edit` of `ProcessNodeStr`

If this node is not linked in an existing network it will be added to the display of the network currently being edited. If there is no such network one will be created.

The commands in this menu are:

- `desc Edit`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

22.8 Describe member `Edit` of `SignalStr`

If this node is not linked in an existing network it will be added to the display of the network currently being edited. If there is no such network one will be created.

The commands in this menu are:

- `desc Edit`: Describe selected member of `SignalStr`. This menu option is a command.

22.9 Describe member `LinkIn` of `DisplayNodeStr`

The `LinkIn` member function selects the next input channel to link to. It's single parameter (`Channel`) specifies the channel index. Ordinarily the first unused channel is linked to. This function overrides that default.

The commands in this menu are:

- `desc LinkIn`: Describe selected member of `DisplayNodeStr`. This menu option is a command.
- `param LinkIn`: Describe the parameters of member `LinkIn`. This menu option invokes the menu defined in Section 22.3 on page 154.

22.10 Describe member `LinkIn` of `ProcessNodeStr`

The `LinkIn` member function selects the next input channel to link to. It's single parameter (`Channel`) specifies the channel index. Ordinarily the first unused channel is linked to. This function overrides that default.

The commands in this menu are:

- `desc LinkIn`: Describe selected member of `ProcessNodeStr`. This menu option is a command.
- `param LinkIn`: Describe the parameters of member `LinkIn`. This menu option invokes the menu defined in Section 22.10.1 on page 157.

22.10.1 Select parameter of `ProcessNodeStr` member `LinkIn` to describe

The commands in this menu are:

- `Channel`: Select this parameter of `ProcessNodeStr` member `LinkIn` to describe. This menu option is a command.

22.11 Describe member `NextFreeInput` of `DisplayNodeStr`

This function displays the next available input link for this node.

The commands in this menu are:

- `desc NextFreeInput`: Describe selected member of `DisplayNodeStr`. This menu option is a command.

22.12 Describe member `NextFreeInput` of `ProcessNodeStr`

This function displays the next available input link for this node.

The commands in this menu are:

- `desc NextFreeInput`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

22.13 Describe member `NextFreeOutput` of `ProcessNodeStr`

This function displays the next available output link for this node.

The commands in this menu are:

- `desc NextFreeOutput`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

22.14 Describe member `NextFreeOutput` of `SignalStr`

This function displays the next available output link for this node.

The commands in this menu are:

- `desc NextFreeOutput`: Describe selected member of `SignalStr`. This menu option is a command.

22.15 Select parameter of `ProcessNodeStr` member `SetSampleRate` to describe

The commands in this menu are:

- `Rate`: Select this parameter of `ProcessNodeStr` member `SetSampleRate` to describe. This menu option is a command.
- `Channel`: Select this parameter of `ProcessNodeStr` member `SetSampleRate` to describe. This menu option is a command.

22.16 Describe member `Raise` of `DisplayNodeStr`

`Raise` will cause a window displaying this network to be raised to the top level over any overlapping windows. Examples of windows that will be affected are a network display containing this node or a plot window for this node.

The commands in this menu are:

- `desc Raise`: Describe selected member of `DisplayNodeStr`. This menu option is a command.

22.17 Describe member `Raise` of `ProcessNodeStr`

`Raise` will cause a displayed window referencing this node to raised to the top level over any overlapping windows. Examples of windows that will be affected are a network display containing this node or a plot window for this node.

The commands in this menu are:

- `desc Raise`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

22.18 Describe member `Raise` of `SignalStr`

`Raise` will cause a window displaying this network to be raised to the top level over any overlapping windows. Examples of windows that will be affected are a network display containing this node or a plot window for this node.

The commands in this menu are:

- `desc Raise`: Describe selected member of `SignalStr`. This menu option is a command.

22.19 Select a member of `DisplayNodeStr` to describe

The following member functions can be selected to be described in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Describe member `Raise` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.16 on page 159.
- `DisplayInputTiming`: Describe member `DisplayInputTiming` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.1 on page 152.
- `Edit`: Describe member `Edit` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.6 on page 155.
- `Unlink`: Describe member `Unlink` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.19.1 on page 160.
- `LinkIn`: Describe member `LinkIn` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.9 on page 156.
- `NextFreeInput`: Describe member `NextFreeInput` of `DisplayNodeStr`. This menu option invokes the menu defined in Section 22.11 on page 157.

22.19.1 Describe member `Unlink` of `DisplayNodeStr`

Member function `Unlink` disconnects this node from the DSP network it is linked in.

The commands in this menu are:

- `desc Unlink`: Describe selected member of `DisplayNodeStr`. This menu option is a command.

22.20 Select a member of this `DisplayNodeStr` to execute

The following member functions can be selected for execution in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Execute selected member of `DisplayNodeStr`. This menu option is a command.
- `DisplayInputTiming`: Execute selected member of `DisplayNodeStr`. This menu option is a command.
- `Edit`: Execute selected member of `DisplayNodeStr`. This menu option is a command.
- `Unlink`: Execute selected member of `DisplayNodeStr`. This menu option is a command.
- `LinkIn`: Execute selected member of `DisplayNodeStr`. This menu option is a command.
- `NextFreeInput`: Execute selected member of `DisplayNodeStr`. This menu option is a command.

22.21 Select a member of `ProcessNodeStr` to describe

The following member functions can be selected to be described in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Describe member `Raise` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.17 on page 159.

- `SetSampleRate`: Describe member `SetSampleRate` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.21.1 on page 162.
- `DisplayInputTiming`: Describe member `DisplayInputTiming` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.2 on page 153.
- `DisplayOutputTiming`: Describe member `DisplayOutputTiming` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.4 on page 154.
- `Edit`: Describe member `Edit` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.7 on page 156.
- `Unlink`: Describe member `Unlink` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.21.2 on page 163.
- `LinkIn`: Describe member `LinkIn` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.10 on page 157.
- `NextFreeInput`: Describe member `NextFreeInput` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.12 on page 158.
- `NextFreeOutput`: Describe member `NextFreeOutput` of `ProcessNodeStr`. This menu option invokes the menu defined in Section 22.13 on page 158.

22.21.1 Describe member `SetSampleRate` of `ProcessNodeStr`

The `SetSampleRate` member function sets the sample rate for the specified output channel of this node. In turn the rates for all input and output channels connected to this node are adjusted with one exception. The adjustment will $\times$ b + c not be made through a node output channel that specifies a timing relationship of `TimingTypeRandom`.

The commands in this menu are:

- `desc SetSampleRate`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

- `param SetSampleRate`: Describe the parameters of member `SetSampleRate`. This menu option invokes the menu defined in Section 22.15 on page 158.

22.21.2 Describe member `Unlink` of `ProcessNodeStr`

Member function `Unlink` disconnects this node from the DSP network it is linked in. All nodes that are connected as outputs from this node will be unlinked. This process continues recursively up to the terminal output nodes of all affected threads. Unlinking the first node in a single thread network will unlink every node in the network. If a node has two or more inputs and only one of these is unlinked the node will remain connected to the network on the unaffected input channel or channels.

The commands in this menu are:

- `desc Unlink`: Describe selected member of `ProcessNodeStr`. This menu option is a command.

22.22 Select a member of this `ProcessNodeStr` to execute

The following member functions can be selected for execution in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `SetSampleRate`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `DisplayInputTiming`: Execute selected member of `ProcessNodeStr`. This menu option is a command.

- `DisplayOutputTiming`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `Edit`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `Unlink`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `LinkIn`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `NextFreeInput`: Execute selected member of `ProcessNodeStr`. This menu option is a command.
- `NextFreeOutput`: Execute selected member of `ProcessNodeStr`. This menu option is a command.

22.23 Select a member of `SignalStr` to describe

The following member functions can be selected to be described in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Describe member `Raise` of `SignalStr`. This menu option invokes the menu defined in Section 22.18 on page 159.
- `SetSampleRate`: Describe member `SetSampleRate` of `SignalStr`. This menu option invokes the menu defined in Section 22.23.1 on page 165.
- `DisplayOutputTiming`: Describe member `DisplayOutputTiming` of `SignalStr`. This menu option invokes the menu defined in Section 22.5 on page 155.
- `Edit`: Describe member `Edit` of `SignalStr`. This menu option invokes the menu defined in Section 22.8 on page 156.
- `Unlink`: Describe member `Unlink` of `SignalStr`. This menu option invokes the menu defined in Section 22.23.2 on page 165.

- `NextFreeOutput`: Describe member `NextFreeOutput` of `SignalStr`. This menu option invokes the menu defined in Section 22.14 on page 158.

22.23.1 Describe member `SetSampleRate` of `SignalStr`

The `SetSampleRate` member function sets the sample rate for the specified output channel of this node. In turn the rates for all input and output channels connected to this node are adjusted with one exception. The adjustment will not be made through a node output channel that specifies a timing relationship of `TimingTypeRandom`.

The commands in this menu are:

- `desc SetSampleRate`: Describe selected member of `SignalStr`. This menu option is a command.
- `param SetSampleRate`: Describe the parameters of member `SetSampleRate`. This menu option invokes the menu defined in Section 22.23.1.1 on page 165.

22.23.1.1 Select parameter of `SignalStr` member `SetSampleRate` to describe The commands in this menu are:

- `Rate`: Select this parameter of `SignalStr` member `SetSampleRate` to describe. This menu option is a command.
- `Channel`: Select this parameter of `SignalStr` member `SetSampleRate` to describe. This menu option is a command.

22.23.2 Describe member `Unlink` of `SignalStr`

Member function `Unlink` disconnects this node from the DSP network it is linked in. All nodes that are connected as outputs from this node will be unlinked. This process continues recursively up to the terminal output nodes of all affected threads. Unlinking the first node in a single thread network will unlink every node in the network. If a node has two or more inputs and only one of these is unlinked the node will remain connected to the network on the unaffected input channel or channels.

The commands in this menu are:

- `desc Unlink`: Describe selected member of `SignalStr`. This menu option is a command.

22.24 Select a member of this `SignalStr` to execute

The following member functions can be selected for execution in the clases derived from them. See Section 23 on page 167 for a description of the ObjectProDSP class structure. The commands in this menu are:

- `Raise`: Execute selected member of `SignalStr`. This menu option is a command.
- `SetSampleRate`: Execute selected member of `SignalStr`. This menu option is a command.
- `DisplayOutputTiming`: Execute selected member of `SignalStr`. This menu option is a command.
- `Edit`: Execute selected member of `SignalStr`. This menu option is a command.
- `Unlink`: Execute selected member of `SignalStr`. This menu option is a command.
- `NextFreeOutput`: Execute selected member of `SignalStr`. This menu option is a command.

23 Class hierarchy

This section describes the ObjectProDSP class hierarchy for user objects. C++ allow a set of member functions to be available in all classes derived form the base class in which the member functions are defined. for CircBufDes not found.

23.1 `ProcessNodeStr` class hierarchy

Following is a list of the classes derived from `ProcessNodeStr`. These classes share the member functions listed in Section 22.21 on page 161 and Section 22.22 on page 163. The derived class at each level are listed. Each of these that is a base class has its derived classes listed in a later section.

1. `Add`, `Block`, `CxFFT`, `FindStartTail`, `Gain`, `Integrate`, `Interpolate`, `MaskWord`, `PackWord`, `Power`, `ProcessNode`, `SampleDelay`, `ToInteger`, `ToMach`, `Truncate`, `UnpackWord`
2. `CxFir`, `Demod`, `Demux`, `GainPad`, `Mux`, `RealFir`, `RepackStream`

23.2 `SignalStr` class hierarchy

Following is a list of the classes derived from `SignalStr`. These classes share the member functions listed in Section 22.23 on page 164 and Section 22.24 on page 166. The derived class at each level are listed. Each of these that is a base class has its derived classes listed in a later section.

1. `InputNode`, `InputWord`, `Signal`
2. `ConstantData`, `Cos`, `CxCos`, `CxImp`, `ImportData`, `Normal`, `Ramp`, `ReadFloat`, `ReadInt`, `UniformNoise`, `VoiceNode`

23.3 `DisplayNodeStr` class hierarchy

Following is a list of the classes derived from `DisplayNodeStr`. These classes share the member functions listed in Section 22.19 on page 160 and Section 22.20 on page 161. The derived class at each level are listed. Each of these that is a base class has its derived classes listed in a later section.

1. `AsciiFile`, `CompareDisk`, `DisplayNode`, `HexList`, `Listing`, `OutputNode`, `OutputWord`, `PlotNode`, `VoiceStripOut`
2. `GenericPlotStr`
3. `EyePlot`, `GenericBlockPlotStr`, `GenericPlot`
4. `GenericBlockPlot`, `Plot`

APPENDIXES

A Environmental variables

ObjectProDSP uses environmental variables to know what directories to search for various files.

The most important of these is `$OPD_ROOT`. This is the root directory under which ObjectProDSP is installed.

Following is a list of all environmental variables created by executing ObjectProDSP with the `-e` option. Using this option is the best way to get accurate information about your installation. The information below may not be accurate for your installation.

```
OPD_ROOT is now set to '/usrb/dist/ftp/opd-0.1'.
The ObjectProDSP default value for 'OPD_ROOT' is '/usr/local/lib/opd_root'.
This variable is the ObjectProDSP root directory.

The ObjectProDSP default value for 'OPD_TEMP' is '/tmp'.
This variable is the system directory for temporary files.

The ObjectProDSP default value for 'OPD_SHELL' is '/bin/sh'.
This variable is the executable for the ANSI standard shell.

The ObjectProDSP default value for 'OPD_X_BIN' is '/usr/bin/X11'.
This variable is the directory for X-windows executables.

HOME is now set to '/home/paul'.
HOME is not set to a default value by ObjectProDSP.
This variable is the user home directory.
```

B Commands index

This section contains an index of ObjectProDSP commands with a cross reference to the menu they occur in.

`param AssignBuffers`: Describe the parameters of member `AssignBuffers`. Section 18.3.1.19.1 on page 146.

`param AssociateNode`: Describe the parameters of member `AssociateNode`. Section 18.3.1.16.1 on page 144.

`param DisplayInputTiming`: Describe the parameters of member `DisplayInputTiming`. Section 22.1.1 on page 153.

`param DisplayInputTiming`: Describe the parameters of member `DisplayInputTiming`. Section 22.2.1 on page 153.

`param DisplayOutputTiming`: Describe the parameters of member `DisplayOutputTiming`. Section 22.4.1 on page 154.

`param DisplayOutputTiming`: Describe the parameters of member `DisplayOutputTiming`. Section 22.5.1 on page 155.

`param Execute`: Describe the parameters of member `Execute`. Section 18.2.2.2.1 on page 131.

`param Execute`: Describe the parameters of member `Execute`. Section 18.3.1.2.1 on page 135.

`param GraphDisplay`: Describe the parameters of member `GraphDisplay`. Section 18.2.2.1.1 on page 130.

`param GraphDisplayWindow`: Describe the parameters of member `GraphDisplayWindow`. Section 18.3.1.13.1 on page 142.

`param Link`: Describe the parameters of member `Link`. Section 18.3.1.17.1 on page 144.

`param LinkIn`: Describe the parameters of member `LinkIn`. Section 22.3 on page 154.

`param LinkIn`: Describe the parameters of member `LinkIn`. Section 22.10.1 on page 157.

`param MakeTarget`: Describe the parameters of member `MakeTarget`. Section 18.3.1.5.1 on page 137.

D GNU GENERAL PUBLIC LICENSE

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in

accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance

by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR

ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

> <one line to give the program’s name and a brief idea of what it does.> Copyright (C) 19yy <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

References

[1] P. Budnik, *ObjectProDSP Overview and Tutorial* Mountain Math Software, September 1994

[2] P. Budnik, DppUser, Mountain Math Software, September 1994

[3] P. Budnik, *ObjectProDSP Library Reference*, Mountain Math Software, September 1994

[4] P. Budnik, *ObjectProDSP Developer's Reference*, Mountain Math Software, September 1994

[5] Brian W. Kernighan and Dennis R. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.

[6] The ANSI X3J11 committe and Herbert Schildt *The Annotated ANSI C Standard, ANSI/ISO 9899-1990*, Osborne McGraw-Hill, 1990.

[7] Edward A. Lee and David G. Messerschmitt, Synchronous Data Flow, *Proceedings of the IEEE*, Vol 75, No. 9, September 1987.

[8] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986.

Index