

Splot

Splot User Guide

Document Number
731-00778A-EN-A

30 Mar 2006

Copyright 2006 Kodak. All rights reserved.

Revision History

Revision	Date	Author	Comments
DRAFT	9 Sept 2000	Holland Gidney	First draft of Splot User Guide
A	24 Mar 2006	Maureen Munroe	Updated to Kodak template. Edited text to meet style guide.



Splot
User Guide

CONTENTS

Revision History i

1. Introduction 1
   - 1.1 What is Splot? 1
   - 1.2 Who Should Use This Manual 1
   - 1.3 Splot Workflow Overview 2
   - 1.4 Splot Components 3
     - 1.4.1 Supported Devices 3
     - 1.4.2 Platforms and Operating Systems 3
     - 1.4.3 Terminal Program 3
     - 1.4.4 Built–in Editor 3
     - 1.4.5 C Language Script Interpreter 3
     - 1.4.6 Graphics Library 4
   - 1.5 Related Documents 4
   - 1.6 About Splot Online Help 4

2. Installation 5
   - 2.1 Installing Splot 5
   - 2.2 Installing a custom version of Splot 5

3. Windows in Splot 6
   - 3.1 Editor / Terminal Window 7
   - 3.2 Graphics Window 8
   - 3.3 Macro Buttons Window 8

4. Menus in Splot 9
   - 4.1 Main menu bar 9
   - 4.2 Main menu bar options 10

5. Using the Mouse to Navigate Splot 11
   - 5.1 Left mouse button 11
   - 5.2 Right mouse button 11
   - 5.3 Using the mouse on the Graphics window 11
     - 5.3.1 Drawing a rubber zoom box 12
     - 5.3.2 Inserting x and y coordinates into a text file 12
     - 5.3.3 Highlighting a graphical element and displaying the text that generated it 12
   - 5.4 Using the mouse on the Editor window 12
     - 5.4.1 Moving the cursor to the mouse pointer location 12

5.4.2 Displaying Help for functions or the variable value in a script 13
5.4.3 Highlighting the corresponding graphical element generated by a line of code 13
5.5 Using the mouse on the Macro Buttons window 13
5.5.1 To use a macro button to send a command: 13
5.5.2 To execute a procedure attached to a macro button: 13
5.5.3 To insert text attached to a macro button into a script:.....14
5.5.4 To select a macro button to edit:.....14

6. Configuring Splot using the Config Menu 14
6.1 Keymapping Items (Keymap List, Keymap Read, Keymap On/Off).....15
6.2 Colors.....16
6.3 Miscellaneous (Toggles).....17
6.3.1 Undo Limit.....17
6.3.2 Overwrite Warning.....17
6.3.3 Error Beep.....18
6.3.4 Tab Compress.....18
6.3.5 Alt Menu Bar Access.....19
6.3.6 Margins.....20
6.3.7 Auto Wrap and Auto Indent 20
6.3.8 Tab Compress.....22
6.3.9 Mouse Buttons.....22
6.3.10 Font.....23
6.3.11 Syntax Colour.....23
6.4 Splot Settings 24
6.4.1 Graphics page size.....24
6.4.2 Include files 24

7. Using Splot 25
7.1 Starting Splot.....25
7.2 Exiting Splot 25
7.3 Connecting to a Local Device.....26
7.3.1 Making a Local (Direct) Connection.....26
7.3.2 Troubleshooting local (direct) connections 26
7.4 Connecting to a Remote Device.....27
7.4.1 Configuring Splot to run as a remote server.....27
7.4.2 Connecting to a Remote Server 27
7.4.3 Splot and Service Shell Software 28
7.5 Loading and Unloading Custom DLL Language Extensions 28
7.5.1 Communicating with a Device 28
7.5.2 Using the command line to communicate with a device 28
7.5.3 Using Splot's command history to resend commands.....30
7.5.4 Using scripts to communicate with a device 30
7.5.5 Running already loaded scripts from a ring of files 31
7.5.6 Troubleshooting Script Errors 32
7.6 Using Command Macros 33

8. Writing and Editing Scripts 36

8.1 Splot Default Mode 36
8.2 Writing Scripts 36
8.3 Editing an Existing Script 37
8.4 Built-in Functions Specific to Splot 37
8.4.1 transmit() 38
8.4.2 transmitf() 38
8.4.3 wait() 38
8.4.4 waitfor() 38
8.4.5 waitscan() 38
8.4.6 waitreg() 39
8.4.7 catch() 39

9. Splot Graphics 39
9.1 Producing Graphs 40
9.2 Making Drawings 40
9.3 Remote Graphical Output 41

10. Appendix A: How to Create Custom DLLs 43

11. Appendix B: Text Editor Keyboard Shortcuts 47
11.1 Unmapped Cursor Motion Keys: 47
11.2 Unmapped Control Keys: 47

1. INTRODUCTION

1.1 What is Splot?

Splot is a software program that can be used to interact with any Kodak output device that has a serial port diagnostic interface. Splot can interact locally or remotely with serial port devices at any standard baud rate up to 115K.

Splot consists of:

- a terminal program
- a built-in editor
- a C language script interpreter
- a graphics library
- an online help system

Splot can be used to send formatted commands to serial port devices and capture any responses from the device. Also, since Splot has a built-in programming language, scripts can be written to parse the data and produce graphs that are based on the device's response(s). See C Language Script Interpreter for more information on using Splot for calculations and Producing Graphs for more information on producing graphs.

Splot can also use the same parsing capabilities to read existing script files or drive firmware simulators. The results from running a script can be analyzed and displayed without the need for post-processing with another application. In addition, with Splot's built-in editor, it is possible to write new scripts and edit existing scripts. See Writing and Editing Scripts for more information on using Splot to write and edit scripts.

Originally designed for generating two–dimensional, publication–quality scientific drawings, Splot contains much of the functionality of PostScript™ and can be used as a general purpose drawing tool. Furthermore, it has functions specifically aimed at producing plots of x and y data pairs. Splot generates a drawing by interpreting a C program that specifies the drawing. See Splot Graphics for more information on using Splot to produce graphs and make drawings.

1.2 Who Should Use This Manual

This manual assumes that:

- You have a working knowledge of prepress operations and computer-to-plate (CTP) output devices.
- You are familiar with using a personal computer running the Windows Operating System (Windows 95, Windows 2000, or Windows NT).
- You have completed the appropriate device–specific training.
- You are familiar with basic firmware commands.

If you intend to use Splot to write scripts, this manual also assumes that:

- You have a basic knowledge of the C programming language

This manual gives you a basic overview of Splot and provides general procedures for using Splot with devices. For further information on what each command, macro button, or menu item does, use the Splot online help.

This manual is intended for anyone using Splot to communicate directly with devices and run, write, or edit scripts. This document supplements Splot's built-in online help and context-sensitive help systems, providing installation and configuration instructions, as well as information on navigating and using Splot.

You should use this manual to learn:

- how to install and configure Splot
- the features and capabilities of Splot
- how to navigate in Splot
- how to use Splot

This document describes Splot version 3.0. The details and features available in other versions may be slightly different.

1.3 Splot Workflow Overview

Part of Workflow	Workflow Step	Where to find more info
Setup	Configure Splot	Chapter 6 Configuring Splot using the Config Menu
Connecting to a Device	Make a local connection Or: Make a remote connection	Chapter 7 Using Splot
Running Scripts	From the **Term** menu, select **Echo to Log**. You are encouraged to enable the **Echo to log** setting, which means the serial port communication is logged to the `terminal.log` file. If **Echo to log** is not selected then port traffic is logged to the `Output` file. The advantage of this setting is that the serial port traffic log will be logged to a disk file in the **Temp** folder and will not be erased once another script is run. This is important as the `Output` file only contains the output from the last executed script and is always deleted when a new script is run. From the **File** menu, select **Open** or select a recently used script from the list of at the bottom of the **File** menu or recently run scripts found at the bottom of the **Exec** menu. Splot loads the script and displays it in the Editor/Terminal window.	Chapter 7.5.1 Communicating with a Device

	From the **Exec** menu, select **Execute** to execute the script currently visible in the Editor/Terminal window. Commands are sent out through the serial port. Any responses received through the serial port are displayed in the Editor window. **TIP:** From the **File** menu, select **Run** to load and execute a script in one step.	

1.4 Splot Components

1.4.1 Supported Devices

Splot primarily interacts with Kodak output devices and their components through the devices' serial ports. Through special DLL extensions, Splot can also communicate with stepper motors, frame grabbers and databases.

1.4.2 Platforms and Operating Systems

Splot 3.00 can be run on either an Intel or Alpha platform. It runs on Windows 95, Windows 98, Windows 2000, and Windows NT operating systems. For best results, use the Splot version that matches your hardware platform: Intel or Alpha.

1.4.3 Terminal Program

The Terminal Program operates in the background of Splot. The Terminal Program is what Splot uses to communicate with devices. The terminal.log is the dialog of this communication and is displayed in the Terminal Window. See Editor/Terminal Window for more information about the Terminal Window.

1.4.4 Built–in Editor

Splot's built–in editor is used for writing new scripts and editing existing ones. The editor is a full–featured, professional, folding, text and binary editor that understands the C programming language. In addition, the editor also understands a library of Splot–specific functions. See Writing Scripts and Editing an Existing Script for more information about using the built-in editor to write and edit scripts.

You write and edit scripts in the Editor window. You input scripting text by typing anywhere in the Editor window. Cursor movement is controlled by arrow keys or the auxiliary keypad, and can also be controlled by clicking the mouse button at the desired cursor location. See Editor/Terminal Window for more information about the Editor window.

With the default Windows mode enabled, the editor behaves much like any other Windows text editor. See Configuring Splot for more information on this feature.

1.4.5 C Language Script Interpreter

Splot's C Language Script Interpreter is an almost complete subset of the C language. It can function as a general-purpose programmable calculator, which can output publication-quality graphics from calculated results.

The interpreter has a built-in library of functions, which allows you to easily write robust scripts and then display the results. It is also extensible with user DLLs so that it can interact with almost any hardware. For example, using special DLLs, Splot can communicate with databases, video frame grabbers, general-purpose I/O cards and stepper motors. User DLLs can also be made to make custom extensions to the language for special purposes, such as image processing for a large bitmap.

See Creating DLLs for more information on how to create a DLL extension.

1.4.6 Graphics Library

Splot contains a built-in drawing and plotting function library whose members can be called by the C Language Script Interpreter to generate drawings. Graphics Library functions are inserted into scripts by using the **Func** menu's **Draw Functions** command. Consult Splot's Online Help for more information on the functions contained in the Graphics Library. Any graphical results from running a script are displayed in the Graphics Window. See Graphics Window for more information on the Graphics Window.

See Splot Graphics for more information on producing graphs and making drawings.

1.5 Related Documents

- For a list of CTP firmware commands and descriptions, see: http://techplanet.creo.com/output/computer_to_plate/all_ctp/nav/control-set.htm
- For list of existing scripts for Thermal 1.0/1.7x Heads, see: http://techplanet.creo.com/output/heads/thermal/nav/downloads.htm#scripts or http://techplanet.creo.com/output/service_tools/splot/nav/downloads.htm
- For more detailed information on using Splot to produce graph and make drawings, see the Splot Manual (located in the Splot\Doc folder, filename splotdoc.pdf)

1.6 About Splot Online Help

Splot online help is context–sensitive at the field-level. It also contains a main–level help system with a table of contents, index and search engine.

The Splot online help contains detailed information on what each menu command is and how to use it when writing scripts. It also contains detailed information on the purpose of the dialogs and macro buttons.

To access field–level online help:

From a menu, right-click on the menu command.

A window displays information about the selected menu command and its use.

Note: If right–clicking on a menu item does not display the help window, make sure that the splot.hlp file is in the same folder as the splotwin95.exe executable. In order for Splot to display the help window, the splot.hlp file must be in the same folder as the splotwin95.exe executable.

To display Help or the variable value for functions in a script:

Double–click on the function, C keyword or the variable value displayed.

A dialog box with Help or the variable value for the item displays.

2. INSTALLATION

2.1 Installing Splot

A full installation of Splot requires approximately 3.5 MB of disk space on your hard drive.

1. Execute the appropriate self-extracting zip `Splot.zip` for your computer in a temp directory.

Note: There is an Intel version and an Alpha version of Splot. Execute the zipped version that matches your hardware.

2. After decompression, run the Splot setup.exe.
3. Follow the on-screen instructions in the Splot installation wizard.

   The wizard will:

   - add Splot to your computer's **Start** menu
   - create a Splot shortcut on your computer's desktop
   - install the following files and folders in `c:\Program\Files\Splot`:



Figure 1 Splot files

If the above installation procedure fails, the only files that need to be installed on your computer are the `splotw95.exe` executable and the splot.hlp file. (See below)

2.2 Installing a custom version of Splot

1. In `C:\ Program Files`, create a folder called `Splot`.
2. From the zip file extract `splotw95.exe` executable and the Splot.hlp files.
3. Copy the `splotw95.exe` and Splot.hlp files into the `Splot` folder.

Note: Despite its name, the `splotw95.exe` executable works with all versions of Windows after Windows 95, including Windows NT.

3. WINDOWS IN SPLOT

Splot consists of three windows, which by default fill the entire screen. They are:

- the Graphics Window (left-hand window)
- the Editor/Terminal Window (right-hand window)
- the Macro Button Window (bottom window)

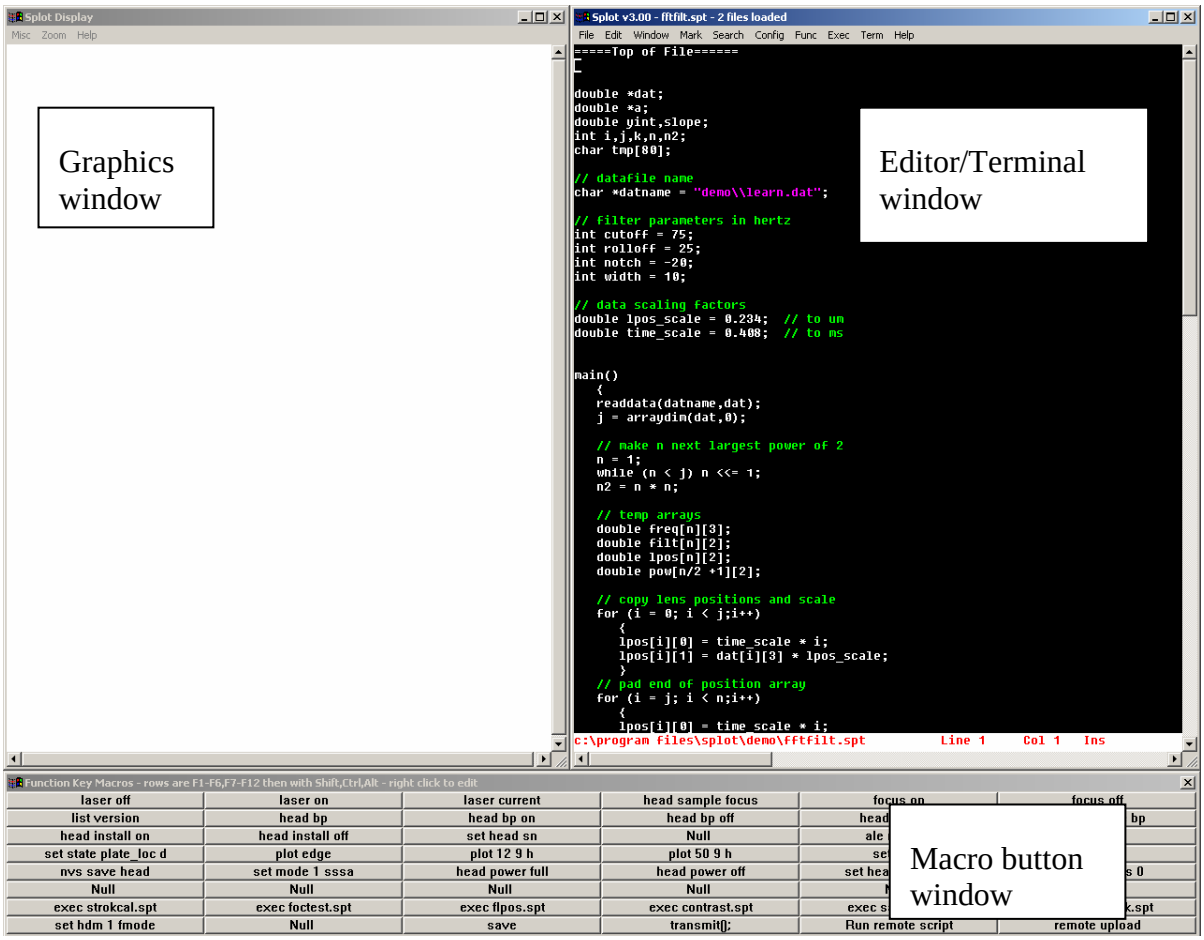


Figure 2 Splot Windows

These three windows are tightly intertwined, even though each window can be individually maximized or minimized and used independently of each other.

For example, if you create and run a script to produce a graph, you'll write and edit the script in the Editor window, the dialog between your computer and the device you're connected to will display in the Terminal window, and the graph will be displayed in the Graphics window.

3.1 Editor / Terminal Window

As the default window, the Terminal window displays when you first open Splot. To display the Editor window:

- load an existing Splot file by selecting **Open** from the **File** menu

Or:

- create a new Splot file by selecting **New** from the **File** menu

Or:

- view a file by selecting the file from the **Window** menu

If you have a number of files already open, to switch to the Editor window:

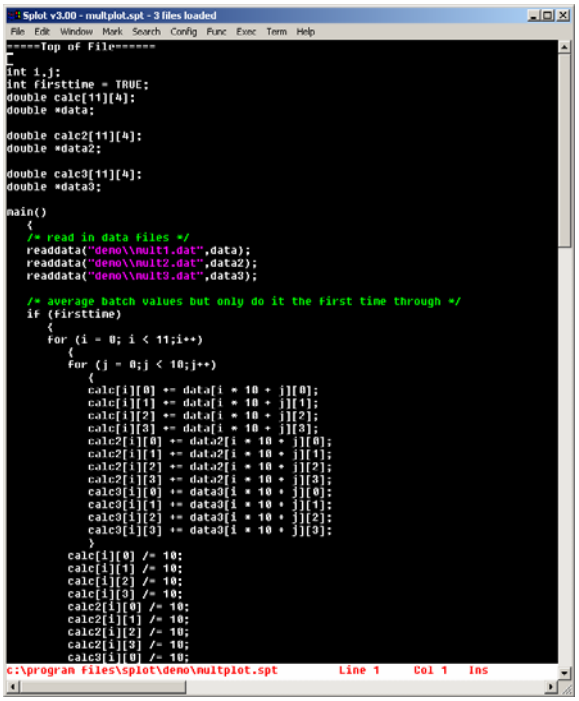
- from the **Edit** menu, select **Next** or **Prev** until the Editor window displays

To display the Terminal Window

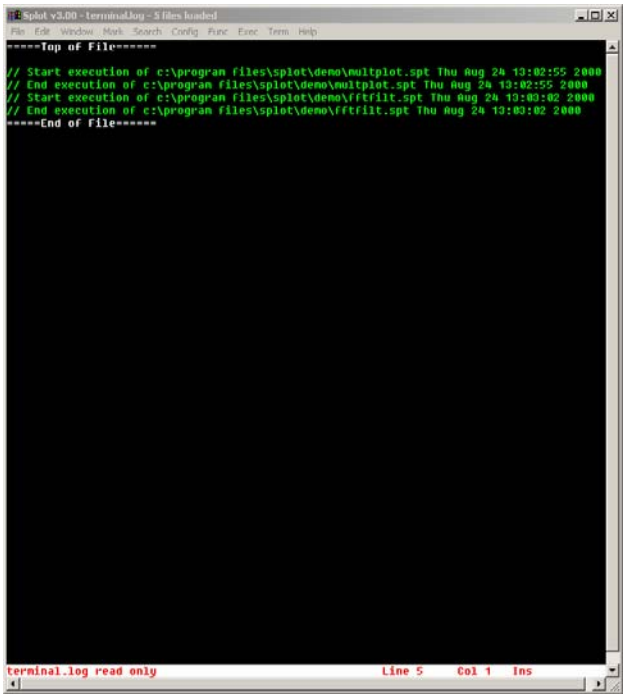
- From the **Term** menu, make sure that **Terminal File** is checked.

Or:

- From the **Window** menu, select **Terminal**.



Editor Window



Terminal Window

The Editor / Terminal window is the main window used in Splot. Depending on the view selected, either the Editor window, containing the text editor, will display or the Terminal window, displaying the terminal.log file, will display.

Note: The Terminal window is the default window.

Typing in the Editor and Terminal windows allows you to do the following:

- communicate directly with a device
- write new scripts
- edit existing scripts

See Making a Direct Connection for information about using Splot to communicate directly with a device. See Writing and Editing Scripts for more information about using the built-in editor to write and edit scripts.

3.2 Graphics Window

To display the Graphics window:

- From the **Window** menu, select **Graphics Window**

See Producing Graphs for more information about using Splot to produce graphs. See Making Drawings for more information about using Splot to make drawings.

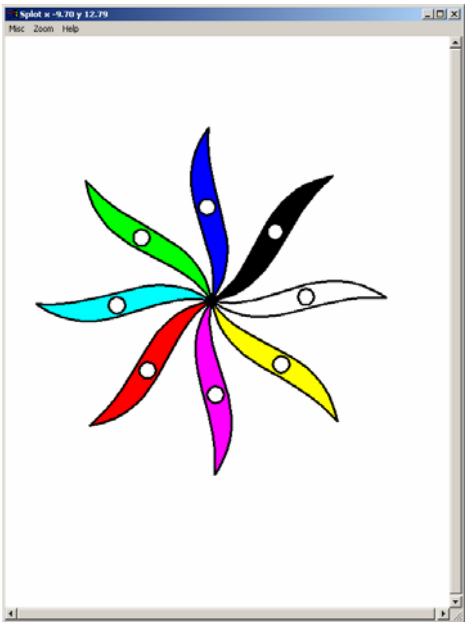


Figure 3 Graphics window

The Graphics window is the drawing area where Splot draws images and displays graphs. Only scripts that produce graphics need the Graphics window to be visible. The Graphics window opens automatically when Splot produces a drawing or graph. However, you can also display the window manually.

3.3 Macro Buttons Window

To display the Macro Buttons window:

- From the **Window** menu, select **Macro Dialog**.

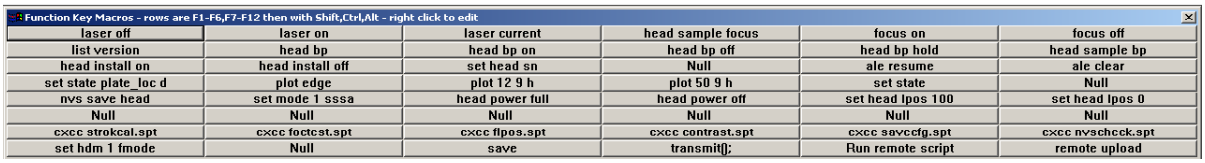


Figure 4 Macro buttons window

Macros are user-definable sequences of keystrokes that once attached to a button can be reused by clicking that button or by pressing its corresponding shortcut keys. In Splot, macros are used mainly as timesavers.

Macros can be used:

- to communicate directly with a device
- as accelerators
- to insert a command or function into a script

The Macro Buttons window (shown above) displays buttons to which macros have been attached. Depending on how it has been formatted, clicking on a button displayed in the window, or pressing its corresponding shortcut keys, will:

- send the attached command to the device
- execute the attached procedure (for example, save the file)
- insert the attached command or function into the text region of the Editor window at the current cursor location

See Using Command Macros for more information about using macros.

A number of macro buttons displayed in the Macro Button window come pre-formatted with common commands and functions, but any or all of them can be edited to suit your needs. See Using Command Macros for more information about editing the macro buttons.

4. MENUS IN SPLOT

Splot has two menu bars, one on the Editor/Terminal window and the other on the Graphics window.

The menu bar on the Editor/Terminal window is the main menu bar in Splot. It is the menu bar from where you configure Splot, and open and run scripts.

The Graphics window also has a menu bar whose menus are used to select drawing functions to insert into scripts. See Splot Graphics for more information on using this menu bar.

4.1 Main menu bar



Figure 5 Main menu bar

This section describes the menus accessible from Splot's main menu bar, which is located at the top of the Editor/Terminal window. It provides a brief overview of the uses of each menu, as well as some specifics on important menu items. Consult Splot's Online Help for more information on Splot's menus and menu commands.

4.2 Main menu bar options

The following menus are displayed on Splot's main menu bar:

File	Contains commands for working with scripts, choosing file formats for graphics you are exporting, and printing. **Note**: The list of script files found at the bottom of the **File** menu are all the files that have been recently used. The scripts files may not be currently loaded into Splot.
Edit	Contains commands for writing and editing the script text, finding script keywords. **IMPORTANT: Enable Edit** must be selected in order to write and edit scripts using Splot's built-in editor. **Enable Edit** is also used to display the **Mark**, **Search**, **Config** and **Func** menus (in this table, these menus are marked with an *), which are hidden if **Enable Edit** is not selected.
Window	Contains commands for displaying the Graphics window, the Macro Buttons window, and the Editor/Terminal window if they are not already displayed. Also contains a command for tiling the three windows so that they fill the screen, and commands for displaying scripts in a ring of files. In addition, the **Window** menu contains commands for displaying the `Output` file, `ErrList` file, and `terminal.log` file. **Note**: The list of script files found at the bottom of the **Window** menu are the script files that are currently loaded, opened or run.
Mark*	Contains commands for marking and unmarking regions of a script, editing text in a script, and the Block Ops functions.
Search*	Contains commands used to find and replace sections of scripts.
Config*	Contains commands for configuring Splot to your specifications.
Func*	Contains commands for inserting Splot-specific functions, standard C functions, C Language keywords, constants and set options into scripts.
Exec	Contains commands for executing a script, or a portion of a script, stopping the execution of a script, and setting a breakpoint in a script*. It also contains script-debugging utilities* and lists of global and local variables used in a particular script*. The **Exec** menu is also used to load custom DLLs. **Note**: The list of script files found at the bottom of the **Exec** menu are the script files that have been run.

	The debugging options (*) only appear if **Enable Edit** is selected.
Term	Contains commands for enabling Terminal mode and input mode, and setting up and using Splot for remote connections.
Help	Contains commands for accessing Splot's main-level Online Help.

Note: The menus marked with an * are only visible in Splot's menu bar if **Enable Edit**, from the **Edit** menu, has been selected. Selecting **Enable Edit** displays the **Mark**, **Search**, **Config** and **Func** menus, which are otherwise hidden.

5. USING THE MOUSE TO NAVIGATE SPLOT

Using the mouse is one of the easiest ways to navigate Splot.

You can use the mouse to:

- Insert functions into scripts displayed in the Editor window
- highlight elements of a drawing
- insert x and y coordinates into a script
- draw a rubber zoom box in the Graphics window
- display Online Help
- find the corresponding code in the Editor window for graphical elements and vice-versa
- changing coordinate points for graphs

5.1 Left mouse button

If a particular window does not already have the input focus, clicking the left mouse button on a window changes the current focus to that window. The left mouse button is also used to select items in the drop-down menus and to select command macros for insertion.

5.2 Right mouse button

Right-clicking on a menu item will display Help for that item.

5.3 Using the mouse on the Graphics window

In the Graphics window, you use the mouse to:

- draw a rubber zoom box to zoom in on a particular section of a graph or drawing
- insert the current x and y coordinates of the mouse pointer into the text file at the current text cursor location
- highlight a graphical element and move the text cursor to the corresponding line in the text in the Editor window that generated that element

5.3.1 Drawing a rubber zoom box

1. With the mouse pointed in the Graphics window, hold down the left mouse button.
2. Drag the mouse to draw a box around the area of the graph or drawing.
3. From the **Zoom** menu select **Box Zoom** and Splot zooms in on the selected region and magnifies it.

5.3.2 Inserting x and y coordinates into a text file

1. In the Editor window, position the text cursor where you would like the coordinates inserted.
2. Double-click the left mouse button on the desired location in the Graphics window.
3. On the Editor window, the x and y coordinates for the spot clicked are inserted into the text file.

Note: If the text cursor is positioned at the beginning of a pair of x and y coordinates, the new coordinates replace the existing coordinate

5.3.3 Highlighting a graphical element and displaying the text that generated it

- In the Graphics window, right-click the mouse button on a displayed element.

The element is highlighted.

In the Editor window, the text cursor jumps to the corresponding line in the text file that generated the highlighted element.

Note: This feature is useful for finding the code associated with a particular element of a drawing and vice-versa.

5.4 Using the mouse on the Editor window

In the Editor window, you use the mouse to:

- move the text cursor
- display Help or the variable value for functions or keywords inserted in a script
- display the variable value for variable names inserted in a script
- highlight the graphic element(s), if any, generated by a line of code

5.4.1 Moving the cursor to the mouse pointer location

1. In the text region of the Editor window, position the mouse pointer where you would like to move the cursor.
2. Left-click.

   The text cursor moves to the mouse pointer location.

5.4.2 Displaying Help for functions or the variable value in a script

- Double-click on the function C keyword or variable name for which you would like Help or the variable value displayed.

A dialog box with Help or the variable value for the item displays.

5.4.3 Highlighting the corresponding graphical element generated by a line of code

- In the text region of the Editor window, right-click on a line of code.

The text cursor jumps to the beginning of the line. In the Graphics window, the corresponding graphical element generated by the selected line of code is highlighted.

Note: This feature is useful for finding the drawing element associated with a particular line of code and vice-versa.

5.5 Using the mouse on the Macro Buttons window

In the Macro Buttons window, you use the mouse to:

- select a macro button, which can:
- send the attached command to the device

Or:

- execute the attached procedure (for example, save the file or send a series of commands)

Or:

- insert the attached text into the text region of the Editor window at the current cursor location

Or:

- select a macro button to edit the attached function or text

5.5.1 To use a macro button to send a command:

1. In the Macro Buttons window, click the macro button to which the command you wish to send is attached.

   The command is sent to the device.

2. Any responses received from the device through the serial port are displayed in the Editor window.

See Using the command line to communicate with a device for more information on communicating directly with a device.

5.5.2 To execute a procedure attached to a macro button:

- In the Macro Buttons window, click the macro button to which the procedure you wish Splot to execute is attached.

Splot executes the procedure.

5.5.3 To insert text attached to a macro button into a script:

1. In the text region of the Editor window, position the text cursor where you would like the text inserted.
2. In the Macro Buttons window, left-click on the macro button with the text you want to insert into your script.

The text attached to that macro is inserted into the script in the Editor window at the current cursor location.

5.5.4 To select a macro button to edit:

- In the Macro Buttons window, right-click on the macro button you wish to edit.

  The Macro Edit dialog box appears.

For information about editing Command Macros, see Using Command Macros.

6. CONFIGURING SPLOT USING THE CONFIG MENU

When using Splot for the first time, it is useful to configure Splot. You configure Splot using the menu items under the **Config** menu.

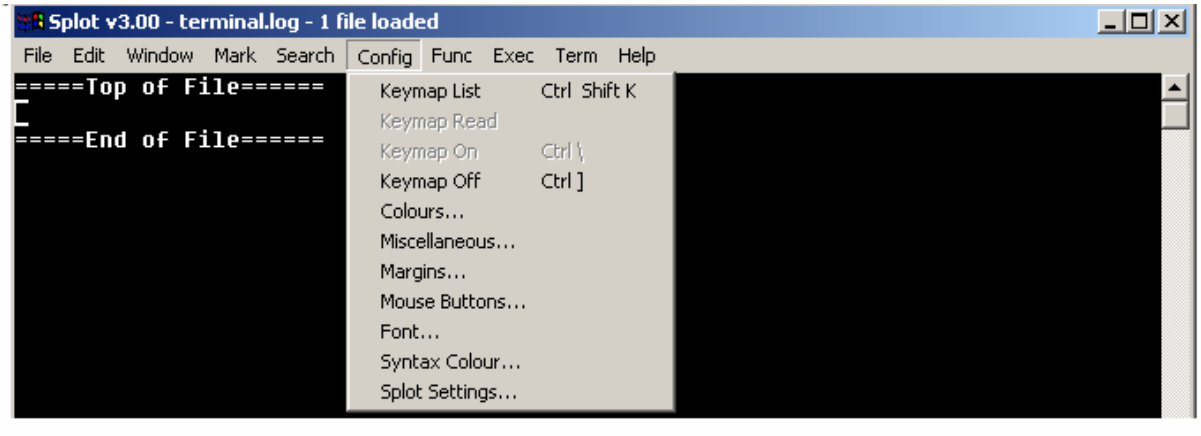


Figure 6 Config menu

You configure Splot using the menu commands on the Editor window.

1. From the **Edit** menu, select **Enable Edit**.

   The **Config**, **Func**, **Edit**, **Term** menus are added to Splot's menu bar.

2. From the **Config** menu, select menu commands to set your preferences for fonts, colors, margins, etc. (See Using the Config Menu to configure Splot for descriptions of the menu items. More information is also available in Splot's online help.)

   Splot automatically saves your configuration so that the next time you open the program it is set-up the same way you left it.

Note: When you install Splot, a `splot.cfg` file, containing a default configuration, is installed in the same folder as the Splot executable. The first time you run Splot, a new version of this file (also named `splot.cfg`), with the saved configuration changes, is stored in the root directory of your `c:\` drive. This file is the one Splot uses the next time, and subsequent times, you start the program. If you wish to return to Splot's default configuration, simply delete the `splot.cfg` copy from the root directory of your `c:\` drive. When you next run Splot, a new .cfg file with the default settings is copied to your root directory.

6.1 Keymapping Items (Keymap List, Keymap Read, Keymap On/Off)

Only use keymapping if you want to customize Splot's built-in editor. It is recommended that you do not use keymapping unless you are an advanced user of Splot.

To enable keymapping:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Key Map** check box.

   A checkmark displays to signal that key mapping has been enabled.

3. Click **OK**.

   The configuration file `splot.cfg` is read to look for redefined keys and definitions.

To disable keymapping:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Key Map** check box.

   The checkmark disappears to signal that key mapping has been disabled.

3. Click **OK**.

   Now, Splot uses the default key mappings even if some key translations are defined in the configuration file (splot.cfg).

To display a list of key reassignments:

1. From the **Config** menu, select **Keymap List**.

2. The `editkey.map` file displays in the Editor window, listing any key reassignments.

Note: Older versions has a Windows Standard mode which when enabled had some built-in key reassignments.

To edit key reassignments:

1. From the **Config** menu, select **Keymap List**.

2. The `editkey.map` file displays in the Editor window, listing any key reassignments.
3. Modify existing lines.

For example, if you add lines such as Ctrl u Ctrl z, Splot interprets this as the command to translate all Ctrl u mappings to Ctrl z. It is possible to have a sequence of key codes before the in order to map a key combination to a Splot default keystroke.

To activate key reassignments that you've changed or added:

- From the Config menu, select Keymap Read.

Splot reads and activates any key re-assignments listed in the Keymap List.

To turn on key reassignments:

- From the Config menu, select Keymap On.

Splot activates the key assignment.

To turn off key reassignments:

- From the Config menu, select Keymap Off.

Splot deactivates the key assignments

6.2 Colors

The **Colours** menu is used to select the text colors used for the Editor window and the command line.

Note: The **Colours** menu is not used to set the colors for syntax coloring. From the **Config** menu selecting **Syntax Colour** to set the colors for syntax coloring.

To select/change the text colors used in the Editor window and for the command line:

1. From the **Config** menu, select **Colours**.
2. The Colours dialog box displays.
3. Select the colors to be used in the Editor window:
   - Use the **Ordinary** box to select the color for ordinary text.
   - Use the **Background** box to select the background color for the Editor window.
   - Use the **Changed** box to select the color for text that has been changed.
4. Select the colors to be used for the command line:
   - Use the **Command Line** box to select the color for text displayed on the command line.
   - Use **the Cmd Line Background** box to elect the background colour for the command line.
5. Click **OK**.

6.3 Miscellaneous (Toggles)

The **Miscellaneous** menu is used to enable and disable **Overwrite Warning**, **Error Beep**, **Tab Compress**, **Key Mapping** and **Alt Menu Bar Access**, as well as set Splot's **Undo Limit**.

6.3.1 Undo Limit

Undo Limit is used to set the maximum number of undo records saved in Splot's memory. The default setting is 30000 (which is essentially unlimited for normal editing tasks).

To set the Undo Limit:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. In the **Undo Limit** box, type in a number entry box to set the maximum number of undo records saved in Splot's memory.

3. Click **OK**.

CAUTION: Do not change the Undo Limit **except if you are using the built-in editor to do column manipulation of very long columns, because, the editor relies on the undo records for some internal operations so.**

If you are using the editor to do column manipulation, the undo mechanism's memory usage gets very high. Entering a zero value for the **Undo Limit** attains a much better performance.

6.3.2 Overwrite Warning

If **Overwrite Warning** is enabled, you will be warned before a file of the same name is over written.

To enable Overwrite Warning:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Overwrite Warn** check box.

   A checkmark displays to signal that **Overwrite Warning** has been enabled.

3. Click **OK**.

Splot will now warn you before overwriting a file of the same name.

To disable Overwrite Warning:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Overwrite Warn** check box.

The checkmark disappears to signal that **Overwrite Warning** has been disabled.

3. Click **OK**.

When saving a filel Splot will no longer warn you before overwriting a file of the same name.

6.3.3 Error Beep

To enable Error Beep:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Error Beep** check box.

   A checkmark displays to signal that **Error Beep** has been enabled.

3. Click **OK**.

In addition to displaying the error dialog, Splot will now beep when there is an error with the built-in editor.

To disable Error Beep:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Error Beep** check box.

   The checkmark disappears to signal that key mapping has been disabled.

3. Click **OK**.

Splot will no longer beep when there is an error with the built-in editor, the program will only display the error dialog.

6.3.4 Tab Compress

If **Tab Compress** is enabled, when Splot's built-in editor writes a file to disk a sequence of n spaces is replaced with a tab character. This reduces the size of a saved file. When reading a file, tabs are always replaced with n spaces.

CAUTION: Enabling Tab Compress is not recommended since the formatting of a file can be corrupted if multiple people are working on a file and using different values of n. However, Tab Compress is the only way to get an explicit tab character in a saved file in text mode.

To enable Tab Compress:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Tab Compress** check box.

   A checkmark displays to signal that **Tab Compress** has been enabled.

3. Click **OK**.

Splot's built-in editor will replace a sequence of n spaces with a tab character when writing your files to disk.

Note: The value of n is determined by a setting in the **Config** menu's **Margins** command, unless the **Default I/O Tab Size** check box (located directly below the **Tab Compress** checkbox in the Toggles dialog box) is also checked. If it is also checked, n is 8.

See Margins for information on setting a value for n to be used by Tab Compress.

To disable Tab Compress:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **Tab Compress** check box.

   A checkmark disappears to signal that **Tab Compress** has been disabled.

3. Click **OK**.

Splot's built-in editor will no longer replace a sequence of n spaces with a tab character when writing your files to disk.

6.3.5 Alt Menu Bar Access

If **Alt Menu Bar Access** has been enabled, Splot's main menu bar (located at the top of the Editor window) can be accessed by pressing the Alt key.

Note: The only reason for disabling this feature is that sometimes the Alt key is pressed accidentally. When this happens, it is not always immediately obvious why Splot's built-in editor is not responding as expected. So, if you only access Splot's menus using the mouse, you may wish turn off **Alt Menu Bar Access**.

To enable Alt Menu Bar Access:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **ALT menu bar access** check box.

   A checkmark displays to signal that key mapping has been enabled.

3. Click **OK**.

   You can now access Splot's main menu bar simply by pressing the Alt key.

To disable Alt Menu Bar Access:

1. From the **Config** menu, select **Miscellaneous**.

   The Toggles dialog box displays.

2. Click the **ALT menu bar access** checkbox.

   The checkmark disappears to signal that key mapping has been disabled.

3. Click **OK**.

Now, the only way to access Splot's main menu bar is by using the mouse.

6.3.6 Margins

Text margins are useful when editing text files that are not 'C' code. Setting margins gives Splot parameters so that when you select the **Reflow** command from the **Edit** menu, the program can reformat paragraphs with a suitable line length.

Using the **Config** menu's **Margins** command, you can set text margins, auto indent, and set the n value for **Tab Compress**.

To set the text margins for plain text:

1. From the **Config** menu, select **Margins**.

   The **Margins** dialog box displays.

2. Set the margins:

3. In the **Left Margin** box, enter a value for the left margin

4. In the **Right Margin** box, enter a value for the right margin

5. In the **Indent** box, enter an indent value

6. Click **OK**.

   If Splot's built-in editor is in text mode, the left, right, and indent values will be used by the **Edit** menu's **Reflow** command to reformat a paragraph

Note: Text mode means that syntax coloring is off. See Syntax Colour for more information.

6.3.7 Auto Wrap and Auto Indent

Auto Wrap and **Auto Indent** are used to control line lengths of scripts being written using Splot's built-in editor.

If **Auto Wrap** is enabled, then once the cursor goes past the right margin setting, a new line is started. If **Auto Indent** is enabled, then the cursor is positioned so that the left edge of each new line lines up with the previous line.

Note: For programming it is best to have **Auto Wrap** disabled and **Auto Indent** enabled. When working with plain text, **Auto Wrap** should be enabled and **Auto Indent** disabled.

To enable Auto Wrap:

1. From the **Config** menu, select **Margins**.

   The Margins dialog box displays.

2. Click the **Auto Wrap** check box.

   A checkmark appears to signal that **Auto Wrap** has been enabled.

3. Click **OK**.

Now in Splot's text editor, once the cursor goes past the right margin setting, a new line is started.

To disable Auto Wrap:

1. From the **Config** menu, select **Margins**.

   The Margins dialog box displays.

2. Click the **Auto Wrap** check box.

   The checkmark disappears to signal that **Auto Wrap** has been disabled.

3. Click **OK**.

Now in the text editor, once the cursor goes past the right margin setting, a new line is not started, the same line continues.

To enable Auto Indent:

1. From the **Config** menu, select **Margins**.

   The Margins dialog box displays.

2. Click the **Auto Indent** check box.

   A checkmark appears to signal that **Auto Wrap** has been enabled.

3. Click **OK**.

Now in Splot's text editor, the cursor is positioned so that the left edge of each new line lines up with the previous line.

To disable Auto Indent:

1. From the **Config** menu, select **Margins**.

   The Margins dialog box displays.

2. Click the **Auto Indent** check box

   The checkmark disappears to signal that **Auto Indent** has been disabled.

3. Click **OK**.

Now in Splot's text editor, the cursor is no longer positioned so that the left edge of each new line lines up with the previous line.

6.3.8 Tab Compress

To set the value of n for Tab Compress:

1. From the **Config** menu, select **Margins**.

   The Margins dialog box displays.

2. Use the **Tab Size** box to type in the number of spaces that will be replaced with a tab character when Splot's built-in editor saves a file to disk, when **Tab Compress** is enabled.

3. Click **OK**.

   Every time you press the Tab key, Splot will now insert a number of spaces in the text file so that the cursor position is a multiple of the Tab Size.

When **Tab Compress** is enabled, the value typed in the **Tab Size** box is used as the number of spaces that equal one tab space when Splot is reading or writing a file.

Note: If **Default I/O Tab Size** check box (located directly below the **Tab Compress** check box in the Toggles dialog box accepted through the **Misc** command) is also checked, then the value used will be 8, regardless of what has been entered in the **Tab Size** entry box.

See Miscellaneous for information on enabling and disabling **Tab Compress**. Also see Miscellaneous for information on enabling and disabling the use of the **Default I/O Tab Size**.

6.3.9 Mouse Buttons

The mouse can be used in Splot's text editor to mark regions of text. Each mouse button can be assigned one of three different marking modes:

- Normal Text
- Lines
- Rectangular

To assign a marking mode to a mouse button:

1. From the Config menu, select Mouse Buttons.

   The Regional Marking Assignments dialog box displays.

2. From the pull-down menu, below the name of the mouse button to which you wish to assign a marking mode, select one of the three marking modes used for marking blocks of text in a script.

3. Click **OK**.

Now when you use the mouse button(s) to which you assigned a marking mode, you can mark regions of text displayed in the Editor window using that marking mode.

Note: The default marking mode for the Left mouse button is normal text and the default for the Right mouse button is lines.

6.3.10 Font

The font, font style, and size of the text displayed in the Editor window can all be configured to your preferences.

Note: The Editor window can only use monospace fonts. These fonts appear in the Font selection dialog box.

To set the font characteristics of the text displayed in the Editor window:

1. From the **Config** menu, select **Font**.

   The Font dialog box displays.

2. Select the font options you desire.

3. In the **Sample** display box, a sample of what the text in the Editor window will look like displays.

4. Click **OK**.

The text displayed in the Editor window will now conform to what you just set.

6.3.11 Syntax Colour

In Splot, syntax colouring is enabled using the **Config** menu's **Syntax Colour** command. The colours for **Comment**, **Quote, Changed, Unbalanced** and **Preprocessor** script elements can all be set individually.

Note: Syntax Colouring, especially the detection of unbalanced brackets,requires significantly more computations from the processor and hence may cause the editor to be unacceptably slow on some systems. Disable **Syntax Colouring**, or at least unbalanced bracket matching, if this happens.

To enable syntax coloring:

1. From the **Config** menu, select **Syntax Colour**.

   The Syntax dialog box displays.

2. Select the **Colouring On** check box.

   A checkmark appears to signal that syntax colouring has been enabled.

3. Click **OK**.

   In the Editor window, the **Comment**, **Quote**, **Changed**, **Unbalanced** and **Preprocessor** script elements will all appear in the colors that were listed in the Syntax dialog box.

To set the colors of script elements:

1. From the **Config** menu, select **Syntax Colour**.

The Syntax dialog box displays.

2. Select the script option's colors.
3. Click **OK**.

   If syntax coloring is enabled, the **Comment**, **Quote**, **Changed**, **Unbalanced** and **Preprocessor** script elements will now appear in the Editor window in the colors you set.

Note: Checking for unbalanced brackets is only completed if the box next to the unbalanced test is selected.

6.4 Splot Settings

6.4.1 Graphics page size

The width and height of the Graphics page can be set using **Splot Settings**.

1. From the **Config** menu, select **Splot Settings**.

   The Splot Settings dialog box displays.

2. In the **page width in cm** and **page height in cm** boxes, type in the size in centimeters to be used for the width and height of the Graphics page.
3. Click **OK**.

   The width and height you chose will be used as the dimensions of the Graphics window.

These settings are added to the top of any generated postscript output files and are used to select the right paper size on printers that have multiple trays.

Note: The page size values must be accurate otherwise it is possible that the printer will not print the graphic and may return an error message.

6.4.2 Include files

If there are any include files needed by Splot's built-in script interpreter, enter the paths to these files using the **Splot Settings** menu command**.**

1. From the **Config** menu, select **Splot Settings**.
2. The Splot Settings dialog box displays.
3. In the **Include Path** entry box, enter the paths of any include files needed by Splot's built-in script interpreter.

Note: Use a semi-colon to separate multiple paths.

4. Click **OK**.

Note: See Splot's Online Help for more information on entering paths.

7. USING SPLOT

The standard process for using Splot to communicate with a device is to:

1. Start Splot.
2. Connect to the device:
   - locally (directly)

     Or:
   - remotely
3. Communicate with the device by either:
   - running scripts

     Or:
   - typing in commands
4. Receive data back from the device.

   Splot will parse the data, do calculations with the data, and plot the data.
5. Interpret the results.

7.1 Starting Splot

To start Splot:

1. Double-click the Splot icon located on your desktop, or select Splot from your **Start** menu.

   Splot opens. By default, three windows display:

   - the Graphics window
   - the Editor/Terminal window
   - the Macro Buttons window

7.2 Exiting Splot

To exit Splot:

- From the **File** menu, select **Exit,** or press Alt + F4, or click on the  icon located on the top right-hand corner of the Editor/Terminal window.

Note: When you exit Splot, any changes you made to the program's configuration are automatically saved in the splot.cfg file in your root directory.

7.3 Connecting to a Local Device

In order to communicate with a device, you must first connect to the device. In Splot, there are two ways of connecting to a device: locally and remotely.

7.3.1 Making a Local (Direct) Connection

1. Hook up a serial cable between your computer's serial port and the device's serial diagnostic port.
2. From the **Term** menu, select **Port**.

   The Pick List dialog box displays.
3. In the **Port** entry box, type in or select the port you wish to use.

Note: Make sure that no other program is using the serial port as only one program can use the port at a time.

4. Click **OK**.
5. From the **Term** menu, select **Port Settings**.

   The Port Setting for port <#> dialog box appears.
6. In the **Baud Rate** entry box, type in the baud rate you wish to use to communicate with the device.
7. Click **OK**.
8. Press **Enter** to see if the expected prompt, for example, `>>,` appears in the Editor window.

Note: The prompt is dependent on the device Splot is connected to.

7.3.2 Troubleshooting local (direct) connections

If the expected prompt does not appear, make sure that the serial port and baud rate selected for **Port** and **Port Settings** under the **Term** menu match the actual port and baud rate being used.

If you still do not get the expected prompt, make sure that no other program is using the serial port on your computer, as only one program at a time can use the port.

Note: You can exit other open programs by typing the commands `pl` and `kill` in the Editor/Terminal window command line. See Using the command line to communicate with a device for more information on using the command line to communicate with a device.

If there is still no response from the machine to which you are trying to connect, then check or replace the cable you are using. As a last resort, try rebooting both your computer and the Kodak device.

7.4 Connecting to a Remote Device

Using Splot, it is possible to communicate with Kodak devices in remote locations through a remote computer that is also running Splot. Provided the remote computer is running Splot, configured for a remote server, and a TCP/IP connection can be established using the name of the remote machine, you can run scripts remotely and communicate with any devices connected to the remote computer.

To use Splot to communicate with a device remotely, you must first configure Splot on the remote computer, to which the Kodak device is connected, as a remote server and then use the version of Splot on your computer (the client) to connect to the remote computer, which should already be running Splot.

7.4.1 Configuring Splot to run as a remote server

- On the remote server, from the **Term** menu, select **Remote Server**.

A checkmark appears next to the command. This means that remote server mode has been enabled, allowing other computers to connect to the computer as clients.

CAUTION: Clients can only connect to a remote server if server mode is enabled on the computer acting as the server.

Note: In order for Splot to be configured as a remote server, Splot must be able to find the remote connect DLL `splotic.dll`.

7.4.2 Connecting to a Remote Server

1. Ensure that the remote computer to which you wish to connect is running Splot.
2. On your computer (the client), open Windows Explorer.
3. In the Splot folder (by default, `\ProgramFiles\Splot`), ensure that the DLL `splotic.dll` and the Splot executable `splotw95.exe` are there.

   `Splotic.dll` is distributed as part of the zipped collection of Splot files.

Note: Note: There is an Intel and an Alpha version of the `splotic.dll`, so make sure that you have the one that matches your hardware.

4. From the **Term** menu, select **Remote Connect**.

   The Enter Remote Computer Name dialog box appears.

5. In the **Remote Computer Name** box, type in the remote server name or select the server's name from among the servers listed.
6. Click **OK**.

   When the connection has been made, the title bar will begin flashing the remote server name. (The machine acting as the remote server will beep three times and display the client name on its title bar.)

Note: If you cannot establish a client connection with a remote machine, check that server mode is enabled on the remote machine to which you are trying to connect and that the machine has Splot running.

7.4.3 Splot and Service Shell Software

The Service Shell software is used for remote and local diagnostics, as well as the operation of a variety of Kodak products. Splot runs as a sub-utility of Service Shell, operating as the scripting engine for the program and allowing you to open and run scripts.

The Rev2 Service Shell installer includes a bundled version of Splot and the necessary DLLs for connecting to a Service Shell host, which are installed at the same time as the DOP software.

To make a connection to a Service Shell host from outside of the Service Shell:

1. From the **Term** menu, select **Remote Connect**.

   The Enter Remote Computer Name dialog box appears.

2. In the dialog box, type in the remote server name or select the server's name from among the servers listed. Prefix the name with `SVSH:<servername>` where `<servername>` is the name of the remote computer.

   > **Note:** To connect to a local Service Shell, you do not need to enter a server name, simply type `SVSH`.

   Currently, with a Service Shell connection, as a client you can only run scripts. Also, with this kind of connection, the **Redo Cmd** and **Remote** commands on the **Term** menu cannot be used.

7.5 Loading and Unloading Custom DLL Language Extensions

7.5.1 Communicating with a Device

There are two ways of using Splot to communicate with a device:

- typing single commands on the command line of the Editor/Terminal window
- running scripts that communicate with the device

7.5.2 Using the command line to communicate with a device

When you only need to send single commands to a device, it is easiest to use the command line at the bottom of the Editor/Terminal window. For example, the command line feature is an easy way to send a command to a device to see if it is connected, whether a device's laser is on or off, or to troubleshoot problems with an existing script you have run.

To use the Command Line to communicate with a device:

1. From the **Term** menu, select **Terminal File**.

   On the white command line located at the bottom of the Editor window, the name `terminal.log` should be visible.

This means the terminal mode is enabled.

If the terminal log file does not display, from the **Window** menu, select **Terminal** to display the terminal log file.

CAUTION: You can only communicate directly with devices connected to the serial port if the Editor window displays the `terminal.log` file.

2. From the **Term** menu, select **Input Mode**.

   This enables input mode, which means you can communicate directly with a device.

3. On the command line, type in a command.

   The cursor jumps to the white editor command line, which displays what you type in.

4. Press **Enter**.

   The text is sent out through the serial port. Any responses received from the device through the serial port are displayed in the Editor window.

To use a macro to send a command when using the Command Line to communicate with a device:

1. From the **Term** menu, select **Terminal File**.

   On the white command line located at the bottom of the Editor window, the name "terminal.log" should be visible.

   This means the terminal mode is enabled.

   If the terminal log file does not display, from the **Window** menu, select **Terminal** to display the terminal log file.

CAUTION: You can only communicate directly with devices connected to the serial port if the Editor window displays the "terminal.log" file.

2. From the **Term** menu, select **Input Mode**.

   This enables input mode, which means you can communicate directly with a device.

3. Click the macro button to which the command you wish to send is attached.

   The cursor jumps to the editor command line, which displays the command attached to the macro.

   The text is sent out through the serial port. Any responses received from the device through the serial port are displayed in the Editor window.

Note: If the command is not sent automatically, it means that there is no enter keystroke attached to the end of the macro. In this case, press **Enter** to send that command.

7.5.3 Using Splot's command history to resend commands

Splot maintains a history of previously sent commands so that there is a history of sent commands and their corresponding responses. Using the command history can save a significant amount of typing when doing repetitive tasks.

To resend a previously sent command using the command history:

1. Ensure that Splot is in Terminal mode.

Note: If Splot is in Terminal mode, the command line displays `terminal.log`. If it isn't, then enable Terminal mode by selecting **Terminal File** from the **Term** menu to display it. If the terminal log file still does not display, from the **Window** menu, select **Terminal** to display the terminal log file.

2. Ensure that Splot is in Input Mode. (Under the **Term** menu, there will be a checkmark next to **Input Mode**.)
3. Activate the command line by pressing **Escape** or clicking with the mouse.

   The cursor displays at the beginning of the command line.
4. Use the up and down arrow keys to scroll through previously sent commands until the command you'd like to resend displays.
5. Press **Enter**.

   The command is resent out through the serial port. Any responses received from the device through the serial port are displayed in the Editor window.

Note: The command may be edited before you send it by using the right and left arrows to position the cursor.

7.5.4 Using scripts to communicate with a device

The most common usage of Splot is to run scripts that communicate with devices to control them and gather information from them.

The following are types of scripts that can be run using Splot:

- scripts that communicate with a device, return data, parse the data and then produce graphs/drawings based on the parsed data
- scripts that produce drawings

Depending on your needs, you can either write new scripts using Splot's built-in editor or run existing scripts. You can also use the editor to edit existing scripts before running them.

Several sample scripts are included with Splot. These scripts are installed automatically when you install Splot and can be found in \ProgramFiles\Splot\Demo.

See Related Documentation for information on where to find other existing scripts.

To run a script:

Note: To load and execute a script in one step, select **Run** from the **File** menu.

1. From the **Term** menu, select **Echo to Log**.

Note: It is recommended that you enable the **Echo to log** setting, which means the serial port communication is logged to the `terminal.log` file. If **Echo to log** is not selected then port traffic is logged to the `Output` file. The advantage of this setting is that the serial port traffic log will be logged to a disk file in the **Temp** folder and will not be erased once another script is run. This is important as the `Output` file only contains the output from the last executed script and is always deleted when a new script is run.

2. From the **File** menu, select **Open** or select a recently used script from the list of at the bottom of the **File** menu or recently run scripts found at the bottom of the **Exec** menu.

   Splot loads the script and displays it in the Editor/Terminal window.

3. From the **Exec** menu, select **Execute** to execute the script currently visible in the Editor/Terminal window.

   Commands are sent out through the serial port.

   Any responses received through the serial port are displayed in the Editor window.

7.5.5 Running already loaded scripts from a ring of files

The Splot text editor can contain a ring of loaded files. Once a script has been run it stays loaded in the editor and may be rerun without reloading. There is no maximum to the number of scripts that can be loaded at a time but only nine are listed in the list of scripts displayed at the bottom of the **File**, **Window** and **Exec** menus.

To run an already loaded script from a ring of files:

1. If it is not already displayed in the Editor window, display the desired script by one of the following methods:
   - scroll through the list of loaded files by using the **Next** and **Prev** items of the **Window** menu
   - directly select the desired file, by selecting the name of the file from the list of (up to nine) currently loaded scripts found at the bottom of the **Window** menu.
   - select and run a script in a single step by selecting the name of the file from the loaded scripts listed at the bottom of the **Exec** menu.
2. From the **Exec** menu, select **Execute**.

   If the script produces any text output, (for example, from a call to puts()), then a special read-only file called `Output` will be added to the ring of files and brought to the foreground while the script is executing so that the text output from the script is immediately visible.

Any communication that the script does on the serial port will be logged either to the `terminal.log` or the `Output` file depending on the **Echo to log** setting under the **Term** menu.

7.5.6 Troubleshooting Script Errors

This section describes some of the more common problems encountered while running scripts and to how to fix or work around them when using Splot to communicate with a device.

Errors while running scripts

If an error occurs during the execution of a script, then an additional read-only file called `ErrList` is added to the ring of files, with the error details as well as the preprocessed script appended. The `ErrList` file may help in debugging the problem.

Note: The `ErrList` file is not brought to the foreground by an error. Instead, the current script is brought to the foreground with the cursor positioned at the point where the error occurred.

A dialog will also appear with the error message. The additional error information in the `ErrList` file may be viewed by manually bringing it to the foreground by selecting **ErrList** from the **Window** menu.

Inability to communicate with a device

The most commonly encountered problem when running an existing script which has been adequately debugged is an inability to communicate with the device connected to the serial port. It is recommended that you verify communications with the device first by interactively sending a command to the device from the `terminal.log` file as discussed in Communicating with a Device and, if necessary, adjust the port settings or cabling.

See Configuring Splot for information on setting port settings. See your device Operator Manual for information on cabling.

Timeout while waiting for a given response from a device

The second most common problem is a timeout while Splot waits for a given response from the machine. There are in general three possible reasons for this:

- there is no communication
- there is a problem with the machine connected to the serial port
- the script does not match the machine

In all three cases, there is some trouble with the computer equipment connected to the serial port, or the script, that needs to be resolved before the script can continue.

Timeouts can occur because a script may have been developed for a particular machine only. Different types of machines have slightly different command sets and, unfortunately, sometimes even different responses to the same command. In general, scripts can be written to accommodate for these differences, provided the scriptwriter is aware of them.

When a timeout occurs, first verify that you have not lost communication with the device (see Inability to communicate with device above). If communication has not been lost, check that there are no problems with the machine (such as an open door). If communication has been lost:

Consult the error message displayed in the error message dialog box, which appears automatically when there is an error using the built-in editor.

The error message will list the script line where the error occurred.

1. From the **Window** menu, select **Errlist**.

   The `Errlist` file displays in the Editor window, allowing you to read more detailed information about the error.

2. In the Editor window, click on the script line where the error occurred.

3. From the **Edit** menu, select **Enable Edit**.

4. Display the script in which the error occurred by:

   - scrolling through the list of loaded files using the **Next** and **Prev** items of the **Window** menu until the file displays

     Or:

   - selecting the name of the file from the list currently loaded scripts found at the bottom of the **Window** menu.

5. In the Editor window, edit the script with the necessary changes and adjustments.

   The error message tells you on which line of the script the problem occurred; the problem will generally be at one of the `wait…()` type of function calls. By consulting the `terminal.log` to see what the actual command response was, you can make the necessary adjustment to the script after switching to script-editing mode.

In the case of a `waitreg()` error, it may be somewhat more difficult to interpret what response the script is waiting for unless you are familiar with regular expressions. In this case, you may want to replace the `waitreg()` function with a simpler `waitfor()` or else remove the wait line all together.

Removing the line is often acceptable if the script has a `wait()` function after every command and the machine can generally tolerate the queuing of multiple commands, provided it is not too many and the next step does not depend on the results of the previous command.

Note: If you need to edit a script that you did not write yourself in order to make it work on a given machine, send an e-mail to the script author or web site owner so that the master copy can be updated and thereby save others from encountering the same problem. It is common practice among script writers to put a comment at the beginning of their scripts which lists the date the script was created, the author's name, and a brief description of what the script does.

7.6 Using Command Macros

A simple way to send a command repetitively as many times as you need to send it, is to assign it to one of the macro buttons visible in the Macro Buttons Window.

Note: From the **Window** menu, select **Macro Dialog** to display the Macro Buttons window if it is not already visible.

Macros are user-definable sequences of keystrokes that once attached to a button can be reused by clicking that button. They are used in Splot mainly as timesavers.

Macros can be used:

- to communicate directly with a device (ie. the macro instructs the device to turn on a laser)
- as accelerators (like keyboard shortcuts)
- to insert a command / function into a script

The distributed splot.cfg file contains a number of sample macro assignments that are a set of predefined commands that may or may not correspond to your needs. These macros may be edited to suit your needs. In addition to just sending text strings on the serial port, macros can also contain editor keystrokes. This means that the definition of macros is somewhat more complicated than just entering the text to be sent.

To execute a macro when you are communicating directly with a device:

- Left-click on the desired macro button or press the corresponding shortcut keys.

Note: Not all macro buttons will work with their corresponding shortcut keys because Windows uses some combinations for its own purposes. (For example the combination Alt + F4 terminates the application). In these cases the keystroke never reaches Splot. To get around this, when assigning keystrokes to a macro, make sure you don't assign the macro a series of keystrokes used already as a Windows default shortcut. Alternatively, only execute a macro button with the mouse.

To insert a macro into a script you are writing or editing:

1. In the Editor window, position the cursor where you would like the macro inserted.
2. Left-click on the desired macro button, or press the corresponding keystroke for the macro, to insert the macro.

The macro is inserted into the script

To edit a macro:

1. Right-click on the button to which the macro you wish to edit is attached.

   The Macro Edit dialog box appears.
2. In the **Macro label** box, type in a new macro definition.
3. Type in a label (name) for the macro. This will be what appears on the button in the Macro Buttons window.

Note: If you leave the **Macro label** box blank, Splot automatically produces a label for the button by using the contents of the macro definition after stripping out all keystrokes other

than plain text. If the Splot-generated label is not suitable for your needs, then an alternate label can be entered in this box.

4. Click **OK** to save your changes.

5. The button you selected will now display the new label.

Note: To attach a macro to one of the macro buttons that is intended to run on a remote server machine, prefix the usual macro definition with `Alt C`. Text that is to be sent on the serial port is prefixed with `Alt N`.

To attach the last sent command to a macro button so that it can be reused:

1. Right-click on the button on which that the command is to be assigned.

   The Macro Edit dialog box appears.

2. Click the **Last Cmd** button.

   The last command sent out on the serial port will automatically be entered in the macro definition edit box. This will include the necessary command prefixes to ensure that the editor switches to the `terminal.log` file before attempting to send the command.

3. Assign a label (name) to the macro. This will be what appears on the button in the Macro Buttons window.

Note: If you leave the **Macro label** box blank, Splot automatically produces a label for the button by using the contents of the macro definition after stripping out all keystrokes other than plain text. If the Splot-generated label is not suitable for your needs, then an alternate label can be entered in this box.

4. Click **OK**.

   The button you selected will now display the new label.

For some macros it may be desirable to only define a prefix to which some text will be added after selecting the macro.

To define a prefix for a macro:

1. Right-click on the button to which the macro you wish to edit is attached.

   The Macro Edit dialog box appears.

2. Edit the macro in the **Macro Edit** dialog box by deleting the macro ending **`Enter`**.

Note: The **Enter** keystroke sends the command attached to the macro button; therefore, removing it from the macro definition will leave the macro text on the editor command line and the cursor positioned to enter a command suffix when the macro is selected. This may be useful for commands that need some numeric input that is variable.

Consult Splot's online help for information on how to make more complicated macros by manually editing the macro text.

8. WRITING AND EDITING SCRIPTS

The scripting language that Splot uses is interpreted ‘C’, which does not require any external compilers. It is a “safe” ‘C’ since the interpreter does range and pointer checking to ensure that there is never an out-of-bounds access, which would otherwise crash Splot. It is not yet a complete ‘C’ interpreter, but a very substantial sub set of the language, which comprises the most important parts.

8.1 Splot Default Mode

By default, Splot is configured to startup in the run script only mode. In this mode scripts cannot be edited and therefore cannot be changed accidentally. In run script only mode, Splot only presents a simplified menu of the items necessary to load and run scripts, as well as configuring serial ports; it does not show the items used to create and debug scripts. Run script only mode is thus recommended for those that are using Splot only to communicate interactively with a machine and to run existing scripts.

Note: To use Splot to write and edit scripts, from the **Edit** menu, select **Enable Edit**.

8.2 Writing Scripts

Splot’s built-in editor is used to write, edit and save new scripts.

To write a new script:

1. From the **Edit** menu, select **Enable Edit** if not already selected.

   A checkmark appears next to the menu item to show that it is enabled.

   The **Mark**, **Search**, **Config**, and **Func** menus are added to Splot’s menu bar.

2. From the **Edit** menu, select **New** to add a file to the ring of files.

   A new Splot text file is created and displayed in the Editor window, with the function name "main()"automatically inserted into the script text.

3. Insert commands or functions by typing them in.

4. selecting pre-set functions from among those listed under the **Func** menu

Note: When a selection is made from the **Func** menu, the text inserted is controlled by a macro and can be customized by selecting **Start Macro Learn** from the **Macro** submenu of the **Edit** menu. Although it’s faster to type in commands directly, if you do not know which function to use, the **Func** menu provides a complete list from which you can choose. If you right-click on any function command under the **Func** menu, the online help for that function displays. Double-clicking on the name of a function that has already been inserted opens up the online help for that function.

5. From the **File** menu, select **Save As**.

6. Save your file as <name>.spt.

8.3 Editing an Existing Script

To edit a script:

1. From the **Edit** menu, select **Enable Edit** if not already selected.

   A checkmark appears next to the menu item to show that it is enabled.

   The **Mark**, **Search**, **Config**, and **Func** menus are added to the Editor/Terminal window menu bar.

2. From the **File** menu, select **Open** or select one of the recently run scripts from the list of recently run scripts listed at the bottom of the **File** menu.

   Splot loads the script and displays it in the Editor/Terminal window.

3. Edit the script by adding, removing or editing commands.

4. Insert commands / functions by typing them in.

Note: When a selection is made from the **Func** menu, the text inserted is controlled by a macro and can be customized by selecting **Start Macro Learn** from the **Macro** submenu of the **Edit** menu. Although it's faster to type in commands directly, if you do not know which function to use, the **Func** menu provides a complete list from which you can choose. If you right-click on any function command under the **Func** menu, the online help for that function displays. Double-clicking on the name of a function that has already been inserted opens up the online help for that function.

5. Remove commands functions by selecting the text you wish to remove and pressing **Delete**.

6. Save your changes one of two ways:
   - From the **File** menu, select **Save** to save your changes, overwriting the existing script.
   - From the **File** menu, select **Save As** to save the edited script as a new file.

8.4 Built-in Functions Specific to Splot

Splot has a number of built-in functions that are not standard 'C' but have been written to facilitate the easy production of robust scripts that communicate with a machine connected to a serial port.

The usual method is to send a command with either `transmit()` or `transmitf()` and then to wait for an appropriate, unique response using one of `waitfor(), waitscan()` or `waitreg()` commands.

Note: Note: For a very simple script where the response is not important, you can use the `sleep()` command to wait for a command to complete.

What follows is information about, and descriptions of, some of Splot's many built-in functions. Consult Splot's online help for additional information on the functions listed below and Splot's other built-in functions.

8.4.1 transmit()

The `transmit()` function can be used to send a command out on the serial port either directly as in `transmit("help")`; or more indirectly, as in `transmit(cmd)` where `cmd` is a 'C' character array containing a text command terminated with the null character `\0`. This function automatically adds a carriage return character `\r` to the end of the command before sending it.

8.4.2 transmitf()

The `transmitf()` command is very similar to `transmit()` except that a string can be formatted before sending it out on the port. For example, `transmitf(carriage moveto %d, pos)`; will fill in the `%d` with the value of the integer variable `pos` before sending it. In other words, `transmitf()` is to `transmit()`, as is `printf()` to `puts()`.

8.4.3 wait()

It is recommended that you always wait for a specific response to a command before sending the next one. To do this you need to use one of the `wait...()` functions. `Wait…()` functions have the general format `wait...<response>, <timeout>` where `<response>` is the expected command response and the integer variable `<timeout>` is a timeout value in seconds.

8.4.4 waitfor()

If the expected response is not received within the timeout period, the wait function will return with a value other than 0. The trick to writing robust scripts is to wait for a response that is sufficiently unique so that the script cannot be fooled by additional text that may be received on the serial port between the time the command was sent and the time that the command response is received. This is often not a problem when waiting for a specific string such as, for example, `laser on`. In this example the function to use would be `waitfor(laser on, 60)`,which will wait for up to one minute for the expected simple text response.

8.4.5 waitscan()

In other cases, especially when the response contains a number that the script wants to extract, the simple `waitfor()` function may not be sufficient.

In these cases, use the `waitscan()` function. This function is very similar to the `waitfor()` function except that it interprets the first parameter as does the standard 'C' function `scanf()`, meaning that it can match strings containing variable content, but it also matches `%` format specifiers.

For example, to extract the dpi setting for the current mode use `waitscan("%d DPI", &dpi, 10)` to wait for the response from a `mode` command. The `&dpi` is then a pointer to an integer variable named `dpi`, which is ready to receive the integer value embedded in the actual response. This is much better than trying to receive the next incoming characters with the `receive()` function and then converting to a number with the `atoi()` function since with the `receive()` function there is no simple way to ensure that the next received number is indeed the dpi value.

8.4.6 waitreg()

The most powerful, but also the most complicated of the wait family of functions is the `waitreg()` function. With the `waitreg()` function, the first parameter string is interpreted as a regular expression. Regular expressions use several characters as special characters and so the wait strings are often mysterious looking to those not familiar to regular expressions. They are, however, a very powerful pattern matching tool and well worth learning for serious scriptwriters. Consult the online help for more information on regular expressions.

This function also differs in the sense that the actual string that matches the regular expression is returned in a buffer so that some decision may be made or data extracted after matching by looking at the matched content. This function can also accept up to 10 different regular expressions to wait for and hence can be set up to wait for one of a number of possible responses and then allow the script to act differently depending on the actual response received.

Kodak's widely distributed scripts make heavy use of `waitreg()` when in fact `waitscan()` would have been equally effective and simpler but this is for historical reasons since `waitscan()` has only been recently available as a function in Splot and it is not worth the trouble to rewrite the scripts.

8.4.7 catch()

One other function that is very important when writing a well-formed script is the special function `catch()`. This function, if defined in the script, is automatically called and executed when the execution of a script is unexpectedly terminated due to an error in the script, due to the selection of **Stop Execution** from the **Exec** menu by the user, or by a call to the `exit()` function with a non-zero argument. It is good practice to use this function to restore any machine parameters that the script has changed.

In this way it is possible to ensure that the running of the script does not change the state of the machine even if it has been interrupted in the middle. Unfortunately, `catch()` does not help in the event of a sudden loss of communication with the machine.

`Catch()` can also be explicitly called so the clean up code need not be duplicated. To do this, simply call `catch(0)` at the end of the script to restore the original settings.

Note: `Catch()` does not know what parameters have been changed and does not do the clean up automatically. Instead the catch routine must contain the explicit `transmit()` and `wait...()` functions necessary to restore the parameters.

9. SPLOT GRAPHICS

Splot was originally designed for producing publication quality graphs. Thus it has a rich set of built in functions that allow the production of just about any conceivable two-dimensional graph.

9.1 Producing Graphs

To produce a graph that has been stored in a two-dimensional array of doubles:

The first column of values should be the x values and the second column should be the corresponding y values. Once this array has been filled in (perhaps with results received from the machine connected to the serial port) it may be plotted.

Call the function `plotdata(array, 0, 1)` by adding this line to the script.

Array is the name of the array of doubles.

This will produce a graph in the Graphics window using all the built-in defaults.

When the script is excecuted next, the axes scale will be automatically chosen and tick marks and tick labels produced. The meaning of the values in the form of an axes label may be added using the label() function.

Note: If the default placement, scaling, tick marks etc. are not acceptable they can always be changed but this will involve additional function calls. For more information about using Splot to produce graphs, consult Splot Manual (splotdoc.pdf), which is installed when you install Splot and can be found in c:\Program Files\Splot\Doc.

For example a different size axis box may be selected by inserting a call to the `abox()` function before `plotdata()`

9.2 Making Drawings

Splot generates drawings from scripts, so in order to make a new drawing you must first write a script. A script that will generate a drawing can be created two different ways in Splot, either by:

- typing in drawing commands and functions using Splot's text editor (Editor window)

  Or:

- by using the mouse and Graphics window menus to insert commands into the text file

Note: Command macros can be used when customizing the text inserted when a function is selected; the macro keys can also be used to save on typing.

Use the mouse to make a drawing:

1. From the **Func** menu displayed on the Editor/Terminal window menu bar, select the desired drawing function. (There are menu commands for the C keywords, drawing functions, miscellaneous functions, and the defined constants.)

   The item is inserted into the text at the current text cursor position in the Editor window.

2. If required by the function chosen, type in coordinate points for parameters. Enter these by moving the mouse to the desired location on the Graphics window and double-click the left mouse button.

If the text cursor is already located at the start of a coordinate pair, that coordinate pair is erased and replaced with the new coordinates.

Note: Parameter errors are not detected until the file is executed.

3. From the **Exec** menu, select **Execute** to execute the text file corresponding to the drawing.

   The drawing is generated and displays in the Graphics window.

Note: Graphs and drawings are *only* generated upon execution of the script file.

Modify an existing drawing:

1. In the Editor window, from the **File** menu, select **Open**.

   The **Open** dialog box displays.

2. From the **Open** dialog box, select the script that generated the drawing.

3. Click **OK**.

   The script loads and displays in the Editor window.

4. From the **Exec** menu, select **Execute**.

   The script is executed. Splot generates the drawing and displays it in the Graphics window.

Note: Graphs and drawings are *only* generated upon execution of the script file.

5. In the Graphics window, select a previously drawn item with the right mouse button.

   In the Editor window, the text cursor jumps to the corresponding line and closest coordinate in the text file.

6. Modify the coordinates on this line by typing or by double-clicking the left mouse button, on the new location.

7. From the **File** menu, select **Save** when you've made modifications to the drawing in the above fashion.

   Splot saves the changes you've made to the drawing.

Note: The utility of this method of modifying an existing drawing strongly depends on how the drawing was made. If the drawing was created by inserting absolute coordinates, the above method will work well. However, if the coordinates used are relative coordinates or calculated coordinates, then it makes no sense to replace them with absolute coordinates.

9.3 Remote Graphical Output

If a script running on the remote server produces graphical output (a drawing or graph), it will be transferred across the network and appear in your Graphics window. If the remote script uses some graphical user interface items in the script, such as dialogs, to query for user

selections, the same items will also appear on your computer (client computer) so that you can input text or make a selection.

10. APPENDIX A: HOW TO CREATE CUSTOM DLLS

Load in a DLL containing a set of custom functions to be added to the interpreter. These will be added to the olib menu and may be used just as any other built in library function. Note that only one DLL at at time can be loaded and thus loading a new DLL will cause previous custom functions to be removed. To build a DLL you will need a C/C++ compiler. Compiled custom functions have the advantage of running much faster than interpreted functions. A DLL once explicitly loaded will reload itself every time the interpreter is started.

To build a DLL you will need a C/C++ compiler, the `basicdefs.h` and `sware.lib` files, installed with Splot, which export the necessary hooks so that your DLL can get the parameter values etc. Configure the compiler to generate a multi-threaded Microsoft Visual C compatible DLL. To illustrate a custom DLL consider the following simple example:

Example DLL code...

```
#include <windows.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
// The next file contains the definitions
// of the interpreter exported functions
// available for use such as get_param_val,
// sntx_err etc. as well as enumerated
// constants and struct definitions.
#include "basicdef.h"

// Be sure to link this file with the
// splotw95.lib library that
// contains the exported functions.
// Do this by adding the lib file to
// the Visual C project.
// Note that the distributed splotw95.lib
// is for use with Visual C version 6 and
// will not work with other versions.

// prototypes
DLLEXPORT union data_val CALLBACK cosdll(void);
DLLEXPORT union data_val CALLBACK strncpydll(void);
DLLEXPORT union data_val CALLBACK printdll(void);
DLLEXPORT union data_val CALLBACK mallocdll(void);

// globals
HINSTANCE hInstance;
char msg[80];

// Put all function names in here so that they
// will be put into the olib menu and registered
// with the interpreter.
int WINAPI DllMain(HINSTANCE hInst, DWORD
                   fdwReason, PVOID pvReserved)
   {
   hInstance = hInst;
   switch(fdwReason)
      {
```

```
      case DLL_PROCESS_ATTACH:
         // add_custom_function is exported by
         // the interpreter for registering
         // functions when the DLL is loaded.
         // The second parameter is the return
         // type. Types are enumerated in
         // basicdef.h
         add_custom_function("cosdll",DUB,
                        (void *) cosdll);
         add_custom_function("strncpydll",ING,
                        (void *) strncpydll);
         add_custom_function("printdll",ING,
                        (void *) printdll);
         add_custom_function("mallocdll",PTRT,
                        (void *) mallocdll);
         break;
      }
   return TRUE;
   }

// demo of a very simple function
DLLEXPORT union data_val CALLBACK cosdll(void)
   {
   union data_val value;

   get_param_val(&value,DUB,ONLY);
   value.d = cos(value.d);
   return value;
   }

// demo of a function with multiple parameters
// and possible error conditions. Illustrates
// the use of pointer parameters.
DLLEXPORT union data_val CALLBACK strncpydll(void)
   {
   union data_val value1,len;
   char *dest,*src;

   get_param_val(&value1,PTRT,FIRST);
   if (value1.pv.p == NULL) sntx_err(NULL_PTR);
   if (value1.pv.pinfo->btype != CHR)
      sntx_err(QUOTE_EXPECTED);
   dest = (char *) value1.pv.p;
   get_param_val(&value1,PTRT,MID);
   if (value1.pv.p == NULL) sntx_err(NULL_PTR);
   if (value1.pv.pinfo->btype != CHR)
      sntx_err(QUOTE_EXPECTED);
   src = (char *) value1.pv.p;
   get_param_val(&len,ING,LAST);
   value1.pv.p = (union data_val *)
                 strncpy(dest,src,len.i);
   return value1;
   }

// more complicated fucntions with a variable
// number of arguments and lower level parameter
// extraction using gettoken() and evalexp().
// Also illustrates writting to the interpreter
// output file using the exported function oputs().
```

```
DLLEXPORT union data_val CALLBACK printdll(void)
   {
   union data_val i,r;
   char typ,tmp[80],tmp2[256];
   int count = 0;
   char token[80];

   tmp2[0] = '\0';
   gettoken(token);
   if (*token != '(') sntx_err(PAREN_EXPECTED);
   do
      {
      if (count != 0)
         {
         if (strlen(tmp2) + strlen(" , ") < sizeof(tmp2))
            strcat(tmp2," , ");
         else sntx_err(LINE_TOO_LONG);
         }
      count++;
      typ = evalexp(&i);
      switch (typ)
         {
         case UCHR:sprintf(tmp,"%d",i.uc); break;
         case CHR: sprintf(tmp,"%d",i.c); break;
         case SHRT: sprintf(tmp,"%d",i.s); break;
         case USHRT: sprintf(tmp,"%u",i.us); break;
         case ING: sprintf(tmp,"%d",i.i); break;
         case UING: sprintf(tmp,"%u",i.ui); break;
         case SFLT: sprintf(tmp,"%g",i.f); break;
         case DUB: sprintf(tmp,"%g",i.d); break;
         case PTRT:
         default:
                 sprintf(tmp,"%x",(unsigned int) i.pv.p);
                 break;
         }
      if (strlen(tmp) + strlen(tmp2) < sizeof(tmp2))
         strcat(tmp2,tmp);
      else sntx_err(LINE_TOO_LONG);
      gettoken(token);
      }
   while (*token == ',');
   if (*token != ')') sntx_err(PAREN_EXPECTED);
   /* add to output file */
   oputs(tmp2,FALSE,8);   // 2nd parameter false unless append to prev
line, 8 is white.
   r.i = 0;
   return r;
   }

DLLEXPORT union data_val CALLBACK mallocdll(void)
   {
   union data_val dsize,dim,value,ret;
   int i,num,numdim = 0;
   static struct var_type newm;     // must be static, global or
dynamically allocated
   struct ptrinfo *pi;
   char *ptr;

   // get the size of the elements
```

```
   get_param_val(&dsize,ING,FIRST);
   // get the number of dimensions
   get_param_val(&dim,ING,MID);
   numdim = dim.i;

   // create and fill in internal data structures
   newm.pinfo = (struct ptrinfo *) malloc(sizeof(struct ptrinfo));
   if (newm.pinfo == NULL) sntx_err(OUT_OF_MEM);
   memset(newm.pinfo,'\0',sizeof(struct ptrinfo));
   pi = newm.pinfo;
   pi->dimen = (short) numdim;
   if (dsize.i == sizeof(double))
      pi->btype = DUB;
   else if (dsize.i == sizeof(int))
      pi->btype = ING;
   else if (dsize.i == sizeof(short))
      pi->btype = SHRT;
   else if (dsize.i == sizeof(char))
      pi->btype = CHR;
   newm.type = pi->btype;

   // find the number of elements in each dimension
   for (i = 0;i < numdim-1;i++)
      {
      get_param_val(&value,ING,MID);
      pi->dim[i] = value.i;
      }
   // get last one
   get_param_val(&value,ING,LAST);
   pi->dim[i] = value.i;
   pi->dim[numdim] = 0;
   pi->star = 1;  // needs to be 1 for regular C arrays no matter how many
dimensions
   pi->avp = &newm;

   // calc number of bytes
   num = dsize.i;
   for (i = 0;i < numdim;i++)
      num *= pi->dim[i];

   // malloc memory for array.
   // There is no way for splot to free this automatically
   // so it will form a memory leak and should be dealt with
   // in this DLL
   ptr = (char *) malloc(num);
   if (ptr == NULL) sntx_err(OUT_OF_MEM);
   memset(ptr,'\0',num);   // blank

   // fill in pointer information
   pi->dp = newm.dp = (union data_val *) ptr;
   pi->num = newm.num = num;
   pi->bsize = newm.dsize = dsize.i;

   /* return pointer to allocated memory */
   ret.pv.p = ptr;
   ret.pv.pinfo = pi;
   return ret;
   }
```

APPENDIX B: TEXT EDITOR KEYBOARD SHORTCUTS

Splot allows you to create keyboard shortcut keys for many of its built-in commands and functions. This section lists the default shortcut keys which can be used in the Editor window.

Note: Keyboard shortcuts are case-sensitive.

10.1 Unmapped Cursor Motion Keys:

Keystroke	Cursor Motion
Left arrow	Left one space
Right arrow	Right one space
Up arrow	Up one line
Down arrow	Down one line
End	End of line
Home	Beginning of line
PgDn	Scroll Down one screen
PgUp	Scroll Up one screen
Ctrl + left arrow	Left one word
Ctrl + right arrow	Right one word
Ctrl + up arrow	Scroll Up five lines
Ctrl + down arrow	Scroll Down five lines
Ctrl + end	End of file
Ctrl + PgDn	End of file
Ctrl + home	Beginning of file
Ctrl + PgUp	Beginning of file
Alt + home	Top of page
Alt + end	Bottom of page
Alt + PgUp	Scroll Up half a page
Alt + PgDn	Scroll Down half a page

10.2 Unmapped Control Keys:

Keystroke	Function
Ctrl + a	Mark entire file
Ctrl + A	Save all changed files
Ctrl + b	Bracket match {, (, [, <, or /*
Ctrl + B	Bitmap graphic output
Ctrl + c	Copy to clipboard
Ctrl + C	Change case of character under cursor
Ctrl + d	Delete word
Ctrl + D	Read in the displayed Macro list
DEL	Delete character

Keystroke	Function
Ctrl + e	Delete to end of line
Ctrl + E	EPS graphic output
Ctrl + f	Open Find dialog
Ctrl + F	Find next
Ctrl + g	Execute file
Ctrl + G	Execute selected region
Ctrl + h	Open Replace dialog
Ctrl + H	Open select other loaded file dialog
Ctrl + j	Join next line to current line
Ctrl + k	Cut current line at cursor
Ctrl + K	List the currently defined key mappings
Ctrl + l	Return to previous set place mark
Ctrl + L	List the currently defined macros
Ctrl + m	Set place mark
Ctrl + M	Open command line input dialog
Ctrl + n	Create new file in ring
Ctrl + o	Open file select dialog
Ctrl + O	Switch to other window if screen is split
Ctrl + p	Print graph to printer (Splot version)
Ctrl + q	Quit without saving file
Ctrl + r	Open replace dialog
Ctrl + s	Save current file
Ctrl + t	Print text
Ctrl + T	Metafile graphic output
Ctrl + W	Write file to disk with different name
Ctrl + x	Copy to clipboard and delete
Ctrl + y	Split edit window horizontally in two
Ctrl + z	Undo changes to last modified line
Ctrl + Z	Zap white space
Ctrl + BACK	Delete line
Alt + A	Alternate binary representation
Alt + B	Set breakpoint
Alt + D	Defines
Alt + e	Execute last macro
Alt + F	PDF graphic output
Alt + G	Opens the Global vars dialog box
Alt + h	View only lines with place mark
Alt + i	Clear all place marks
Alt + I	Step into
Alt + k	Enter next char literally
Alt + L	Opens the Local vars dialog box
Alt + O	Step over

Keystroke	Function
Alt + p	Position cursor at previous location
Alt + P	Postscript direct to Lpt1:
Alt + q	Restore a line from undo record
Alt + r	Re-flow paragraph, stops at blank line
Alt + s	View lines containing search string
Alt + t	Teach (end) new macro
Alt + T	Postscript graphic output
Alt + v	View only changed lines
Alt + w	Swap current character and next character
Alt + x	Enter control character
Alt + z	Zero command multiplier
Alt + 0-9	Enter command multiplier
Alt + .	Edit next file in ring
Alt + ;	Edit previous file in ring
Alt + F4	Remove all loaded files, terminate editor and close application