ChipProg+

# Universal Programmer

# User's Guide

Phyton™

**Copyright Notice**



**Disclaimer**

The information in this document is subject to change without notice and does not represent a commitment on any part of Phyton. While the information contained herein is assumed to be accurate, Phyton assumes no responsibility for any errors and omissions.

In no event shall Phyton, its employees or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

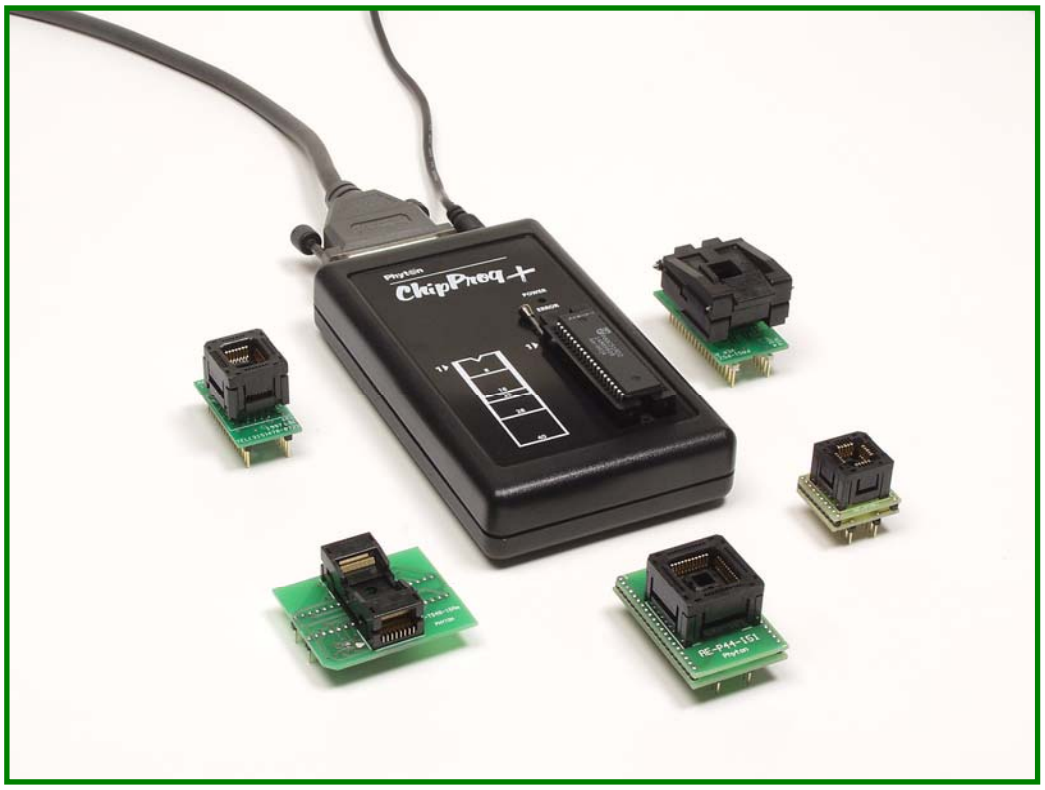
**Trademarks**

ChipProg+ is a trademark of Phyton, Windows and MS-DOS are trademarks of Microsoft Corp. All other product names are trademarks or registered trademarks of their respective owners.

# Table of Contents

# Chapter 2 Introduction

ChipProg+ is a universal programmer that supports thousands of programmable devices, including parallel and serial EPROMs and EEPROMs, embedded microcontrollers with reprogrammable code and data memories, low-density PALs and PLDs. ChipProg+ can handle many distinct device technologies. The picture below shows the ChipProg+ programmer surrounded by several types of programming adapters.




ChipProg+ has a 40-pin zero-insertion-force (ZIF) socket for direct programming of dual-in-line (DIP) package devices. The socket accepts any Phyton brand adapter, as well as many popular third party adapters, and thus works with widely used non-DIP packages (SOIC, SSOP, TSOP, PLCC, QFP, BGA and others.). A variety of devices programmable in-system can be programmed by using special cable adapters without removing from the equipment where the device is installed.

The programmer is small and intended for both engineering and low- to mid-volume manufacturing. The ChipProg+ software runs under Windows. It's easy to use and very intuitive. It includes all operations that are commonly needed by engineering programmers, as well as an embedded script language and some tools to help automate the programming.

## Package Content

A standard ChipProg+ kit includes, but is not limited to:

- A programmer unit enclosed in a plastic bag
- Power supply (14V – 15V/0.5A unregulated or 18V/1A regulated)
- A cable for connection to a PC parallel (printer) port (LPT)
- Software on a CD-R

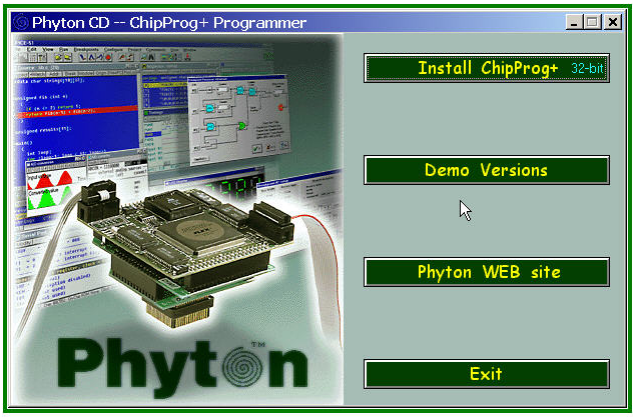
If the programmer has been ordered with additional adapters the kit may include them or they may be shipped in a separate shipping container.

## System Requirements

- A personal computer working under control of Windows 98/2000/NT/XP
- Pentium-II CPU or higher
- 256MB of RAM
- At least one parallel line printer (LPT) port that can be assigned for exclusive use by the programmer
- A hard drive with at least 100MB of free space

# Chapter 3 Software Installation

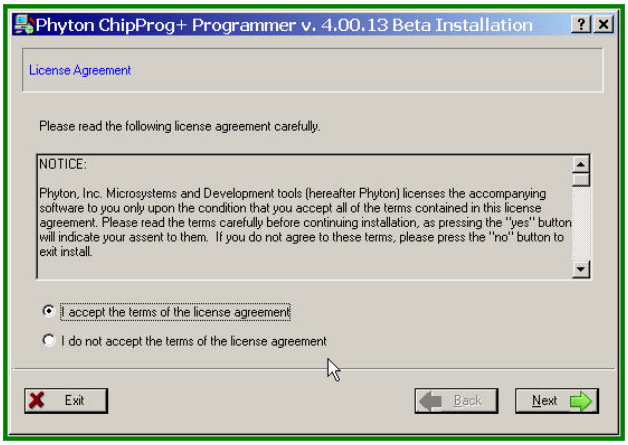
The included Phyton CD contains the ChipProg+ software package and associated user manuals. Follow the procedure below to install the software.




Installing from a CDROM

Insert the CD in a CD drive. The auto-run will launch the installation dialog. Click the Install_ChipProg+ button. If the disk auto-run function is disabled, start the *setup.exe* file from the CD.

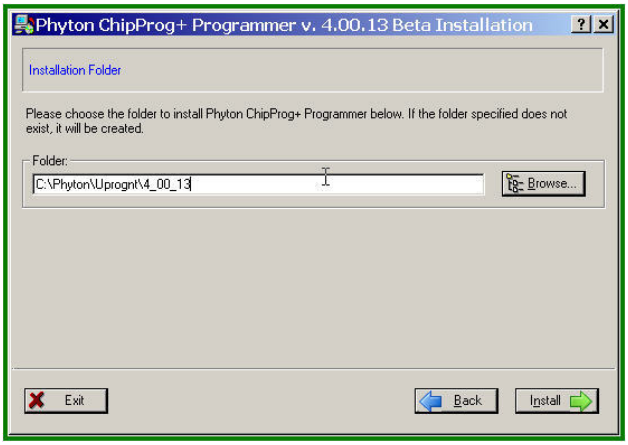
Clicking the Demo Versions button invokes an installation menu for demo versions of other Phyton development tools.




Accept the License Agreemenr Terms

Read the Phyton License Agreement. Use the Page Down button to scroll down the text.

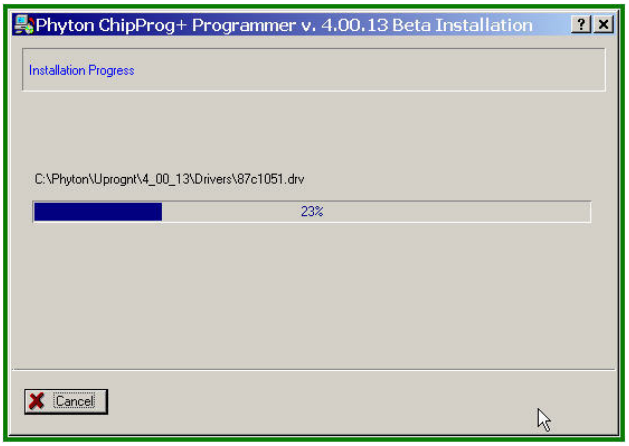
If you agree with the license terms, check the top radio button and click Next. If you do not accept the license terms, installation will terminate.




Select a Destination Folder

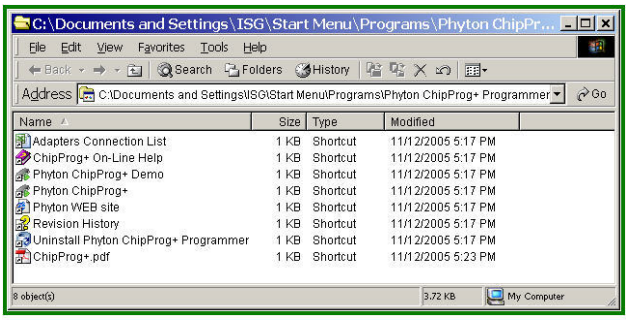
The installation program prompts the user to specify the folder for installing the programmer software. The installation process chooses a default folder on your C drive. If this folder is acceptable, click Install. If you wish to specify another existing folder, you can browse for it.

If you prefer, you can type in the entire path to the folder. In either case, click the Install button to continue or return one step back if necessary.




Installation in a Process

The computer displays the installation process progressing and the files being installed.




ChipProg+ Shortcuts Folder

When installation is complete, the computer displays a destination folder containing a set of shortcuts to the tools and manuals.

The following table lists the shortcuts that are accessible from the screen.

| Screen Selection | Program Launched or File Invoked |
|---|---|
| Phyton ChipProg+ | Launches programmer software |
| Phyton ChipProg+ Demo | Launches demo version of the programmer software, which can be evaluated without the ChipProg+ hardware |
| ChipProg+ On-Line Help | Invokes on-line Help manual |
| ChipProg+ pdf | Invokes the programmer manual in the PDF format |
| Adapters Connections List | Invokes a Microsoft Excel file with diagrams of all Phyton programming adapters for ChipProg+ |
| Revision History | Invokes a history of ChipProg+ software versions' updates |
| Phyton website | Connects a computer to the www.phyton.com website if this computer is connected to Internet |
| Uninstall | Launches the uninstaller to remove the ChipProg+ software from a computer |

# *Chapter 4 Hardware Installation and Start up*

## Controls and Insertion Devices and Adapters

The picture below shows the top of the programmer. A parallel cable connector for linking to a PC parallel (printer) port and a coaxial connector for connecting a power adapter cord are situated at the rear of the programmer unit. A zero-insertion-force (ZIF) 40-pin DIP socket on top of the unit allows insertion of DIP-packaged devices 150 mil to 300 mil wide. A pictogram at the left of the ZIF socket prompts correct positioning of the device to be programmed. This will be either a DIP-packaged chip or an adapter for non-DIP devices.

ChipProg+ unit

## LED Indicators

There are three LEDs on top of the programmer:

- *POWER* – green LED is always on when the programmer is powered
- *ERROR* – red LED is on when programming operation failed or an over-current condition was detected through the target device. When this LED is illuminated, the programmer blocks any signals coming to the socket pins.
- *BUSY* – yellow LED indicates that the programmer is executing an operation on the target device. While this LED is illuminated, the device should not be removed from the programming socket or disturbed in any way.

## Programmer Self-Testing

The programmer always starts by conducting a few tests, which can be divided in two groups:

- Communication tests
- Hardware tests

The first group checks the communication link to the PC.

The second group of tests runs a complete functionality check of all programmer hardware resources. During the testing process the programmer generates logical signals on the ZIF socket's contacts. These signals may damage any devices that are present in the socket. In most cases the programmer detects the device that's in the socket, stops the tests and turns on the red *ERROR* LED. However, to be safe, it's best to make sure that no devices are present in the programming socket when you start up the programmer.

If the programmer fails the test it issues an appropriate error message or warning on the PC screen.

## Starting the Programmer

Make sure the programming socket is empty; prior to startup you should remove any device or programming adapter that may have been inserted.

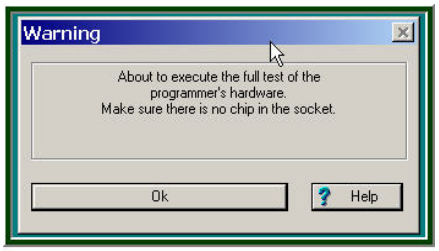
Connect the programmer to one of the PC printer ports: LPT1, LPT2, LPT3 or LPT4. Make sure that the computer's LPT port that drives the programmer works in either a standard mode (SPP) or extended mode (EPP) and that no other equipment is connected to the same LPT port. This includes printers, security dongles, etc. It is permissible to connect the LPT cable while the PC is running.

Check the input voltage marking on the power adapter to make sure it matches the actual voltage value (~110V in North America and ~110 – 240V almost everywhere else). Plug the power adapter into to the power outlet (mechanical converters for adaptation might be required). Then plug the coaxial connector into the power input plug on the programmer unit, and make sure the green LED is illuminated, indicating that power on.


Communication Setup

Start the ChipProg+ program by clicking an appropriate icon in the folder where the programmer software has been installed. You should get the communication setup screen on the left.
Select a vacant LPT port and make sure it is not blocked or used by any other application running on your PC. Then click OK to start self-tests.

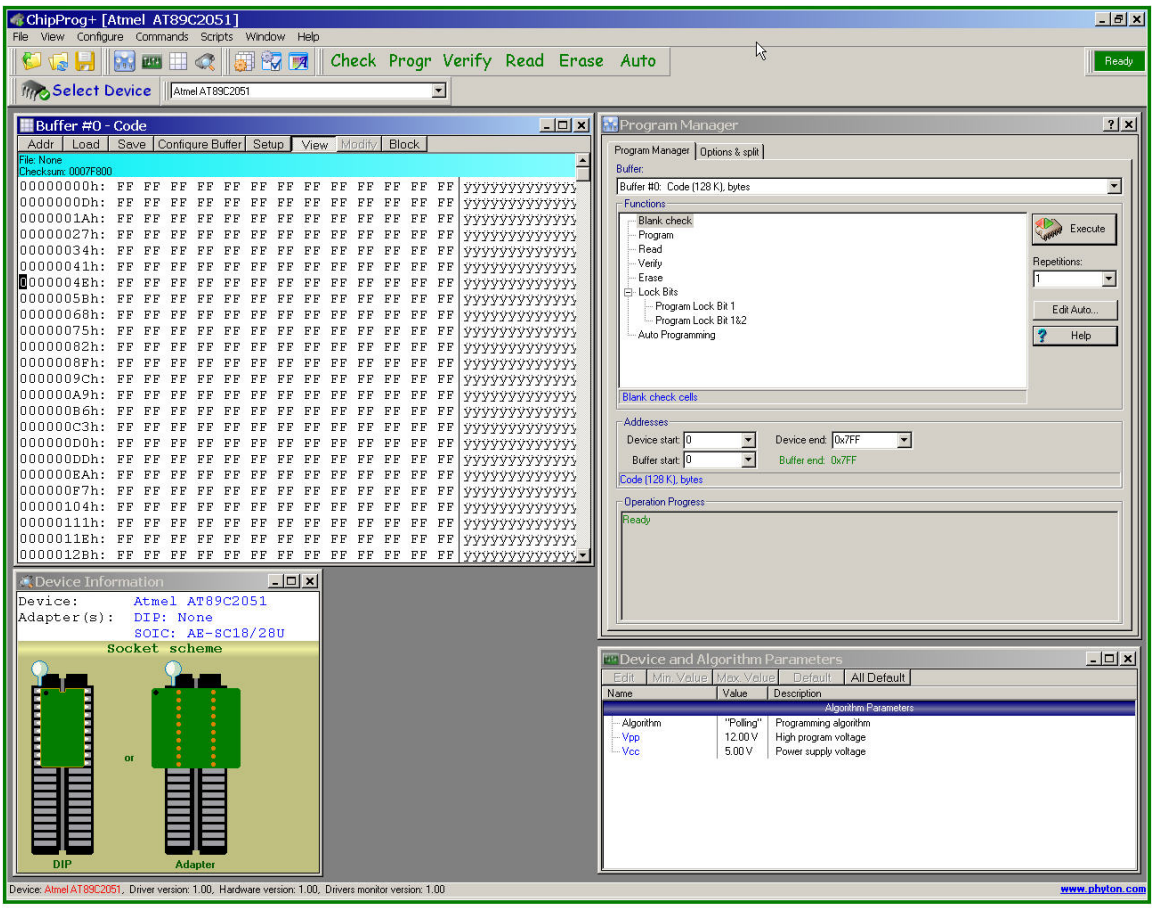
The Demo button in the left bottom corner of the dialog allows evaluation of the programmer's user interface without use of the programmer itself.




Startup Warning

The program confirms linking to the programmer hardware by issuing the warning shown here. Make sure the programmer socket is empty and click OK.

If the programmer has successfully passed the self-test, the program will open the main window. Otherwise you will get an appropriate error message and prompt. A typical main window is shown below.




ChipProg+ Main Window

# Chapter 5 Toolbars and Graphic User Interface Customization

The picture below shows the complete set of toolbars for the programmer. By default the ChipProg+ main window opens with toolbars that do not include all these controls.

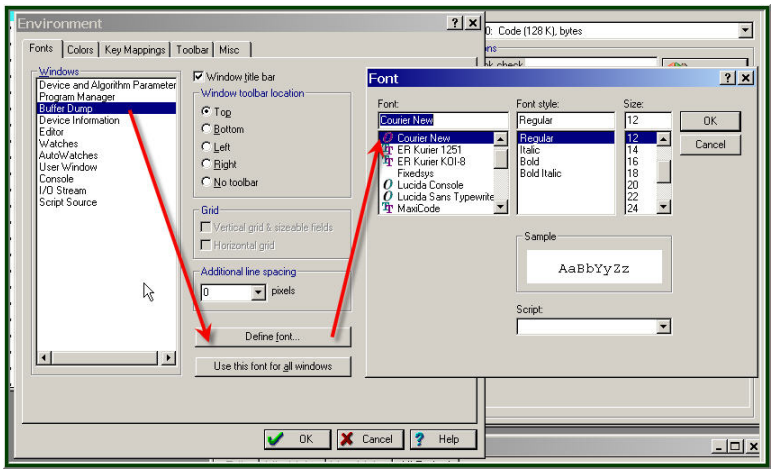
The top line, which is shown right under the ChipProg+ title line of the main window, is the ***Main*** menu. A second line under the ***Main*** menu line displays icons and buttons of the most-frequently used commands that deal with files and target devices. The third line displays a target device selector and the fourth line, which is not displayed by default, includes options for an embedded editor and commands for scripts.


## Customizing the Environment

The programmer working environment includes: toolbars, key mapping, fonts, colors and other minor parameters. There are two way to invoke the dialog for the environment settings: a) right-click the mouse when the cursor is within the toolbar space and select ***Customize..;*** b) select ***Main*** menu > ***Configure*** > ***Environment***. At the first start we suggest skipping this chapter and using the default settings of the programmer software. You can customize the environment any time you need.

## Customizing Fonts

To change the fonts for messages and data in the programmer windows click on the Fonts tab. It will open this dialog:




To change the font for a particular window, select the window and click the Define font button. Then pick the font type, style and size from a pop-up menu and click OK twice to

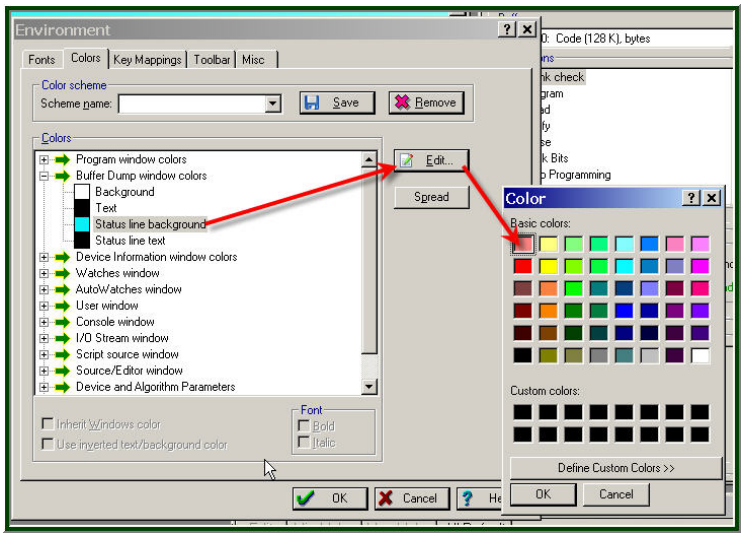
complete the font setting. After you set the font for one programmer window you can export this setting for all other windows by clicking the Use this font for all windows button.

In the same dialog you can specify a location for each window's local toolbar. By default these toolbars reside at the top of the window. You can move them to any of the four edges of the window or turn off the local toolbar by checking the appropriate radio button in the ***Window toolbar location*** submenu.

## Customizing Colors

You can set individual colors for backgrounds and texts in the programmer windows. Click the Colors tab in the ***Environment*** dialog to open a setting dialog, shown below.

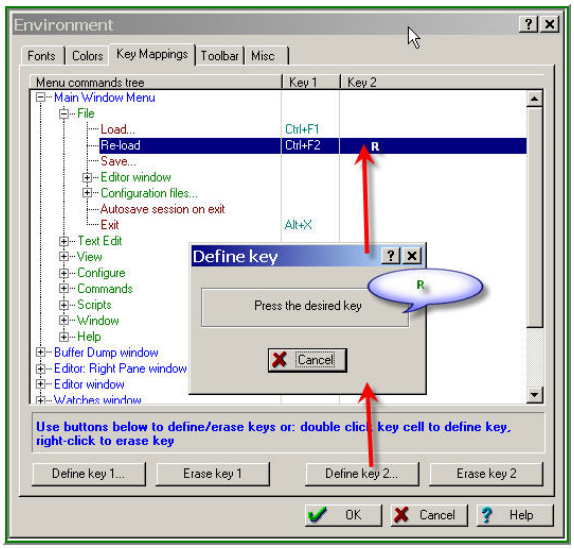
To change a color of a particular window element, highlight it in the left pane. Then click the Edit button to open a color palette, pick the color you want and click OK in the palette box. When you complete the color configuration process, the program will prompt you to




assign a unique name to the color scheme. Type the name into the ***Scheme name*** box and click the Save button. You may create several color schemes and invoke them at a later time.

## Key mapping

By default the ChipProg+ program includes a number of preprogrammed hotkeys that are used to simplify operations with the programmer. You can use these hotkeys, reassign them and add your own hotkeys. You can assign up to two hotkeys to any command from the ***Main*** menu tree. Click the Key Mappings tab of the ***Environment*** dialog to reassign hotkeys for your convenience. See the dialog below.




Adding a Second Hotkey

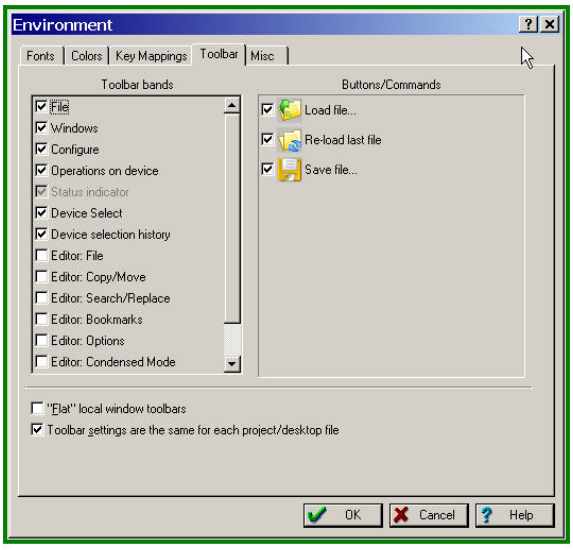
To re-define existing hotkey #1 click the Define key 1 button. The program will prompt you to press a new key or key combination.

To add an additional hotkey click the Define key 2 button. The program will prompt you to press a new key or key combination. See the picture.

Do not forget to click the OK button to save all changes made in the dialog.

## Customizing Toolbars

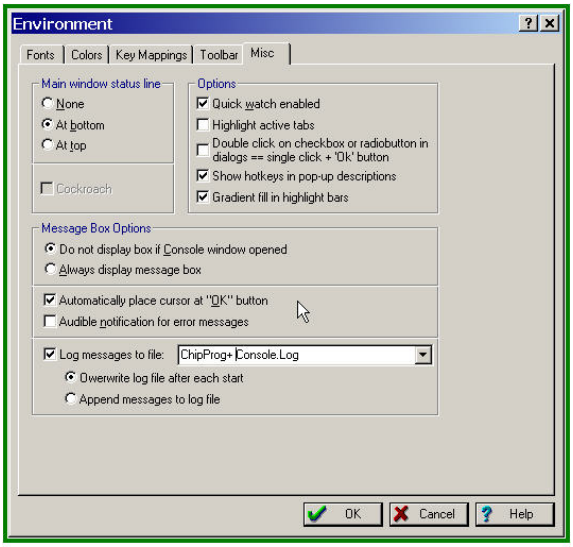
Click on the Toolbar tab of the ***Environment*** dialog to customize buttons on the programmer toolbars. Here you can add or remove any icons belonging to the toolbar bands at left. Checking a box at left brings to the right pane a list of the commands that are available for the selected band. Checking a particular command places it on the toolbar; un-checking - removes it from the programmer toolbar.




Customizing Main Toolbar

## Miscellaneous Settings

Several minor settings are gathered in the dialog that can be opened by clicking the Misc tab in the ***Environment*** dialog (see it on the picture below). Most of these settings are for advanced users.

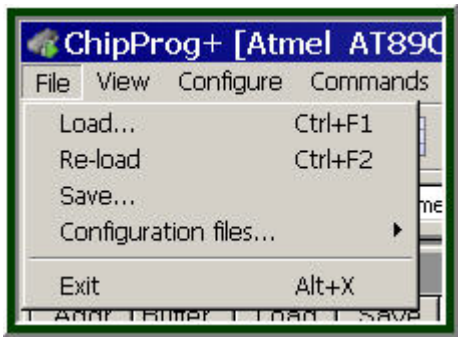



Miscellaneous Settings

# Chapter 6 The 'File' Menu

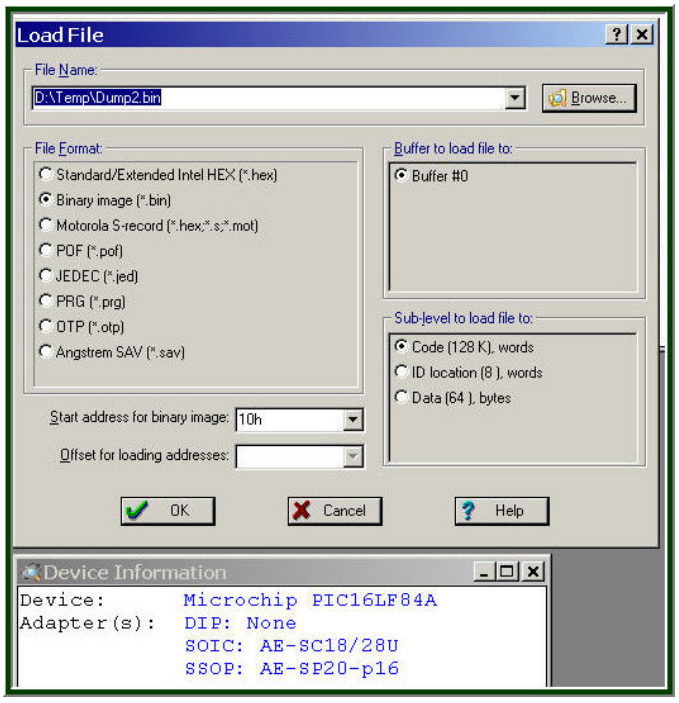
The programmer ***Main*** menu includes a few submenus displayed on the most top toolbar. The following chapters describe commands (functions) of each these submenus.

The ***File*** menu includes several commands operating on files of different types (see the picture below). The three most frequently used commands: ***Load...***, ***Re-load*** and ***Save...*** appear also as shortcut icons on the toolbar situated below the Main menu line (yellow folders and a diskette).




Load Reload Save

## Loading and Re-loading Files to Buffers




Click the ***Load*** icon or press Ctrl+F1 to open the loading dialog.

Here you can load a file to the buffer. You can write this file into the target chip either after editing or "as is." The dialog enables you to browse or to type in file paths directly. You can specify the file format, the destination buffer (if more than one has been opened), sub-level of the file memory (if more than one exists for the target device), an offset for the loading address, and a start address for loading a binary file if the ***Binary image*** format was chosen.

## *Loadable file formats*

The files to be processed by the PC for the ChipProg+ contain the data to be written into the target chip, as well as addresses in the target device and some other helpful information. The ChipProg+ loader is able to load files in all popular formats generated by compilers and other programs that prepare data for burning into programmable memory devices.

***Binary*** image files do not include information about addresses. Any file, whether or not it specifies addresses, can be loaded into the programmer as a binary file. In order to handle such a file, you need only specify the starting address of the binary image when loading binary files from the PC's disk. This format is commonly used when one reads or duplicates an already programmed device. All other loadable formats carry addresses and other references that enable ChipProg+ to write files accurately to specified addresses of the target device.

The most commonly used data file format is standard ***Intel HEX***. The "HEX" designation is somewhat misleading; perhaps it should be called "ASCII-encoded HEX." It contains load addresses along with program codes. This type of file can be recognized by a colon (:) at the beginning of every text line.

***Motorola S*** format is also very popular. As with ***Intel HEX, Motorola S*** format is an ASCII file that contains load addresses along with program codes and other information. ChipProg+ supports all types of ***Motorola S*** files.

***POF*** is a registered Altera-type format used for programming Altera PLDs and some other devices. A ***POF*** file contains load addresses along with program codes, checksums and some other information.

***PRG*** format is a variation of the ***Intel HEX*** format used for programming Xilinx PLDs.

## *Loading files to sub-levels*

For some programmable devices (especially microcontrollers), the fully addressable memory range may comprise a number of functionally different areas (sub-levels), and a compiler may generate distinct files for each sub-level. Each sub-level is associated with a certain type of target device's address space. For example, for the Microchip PIC16F84 microcontroller (see the picture above) every buffer has three sub-levels: a) code memory; b) EEPROM data memory; c) user's identification; for the Intel 87C51FA microcontroller every buffer has two sub-levels: a) code memory; b) encryption table. ChipProg+ recognizes the specific sub-levels in each supported device. Select each sub-level to load a chosen file to an appropriate part of memory.

## *Offset for loading addresses*

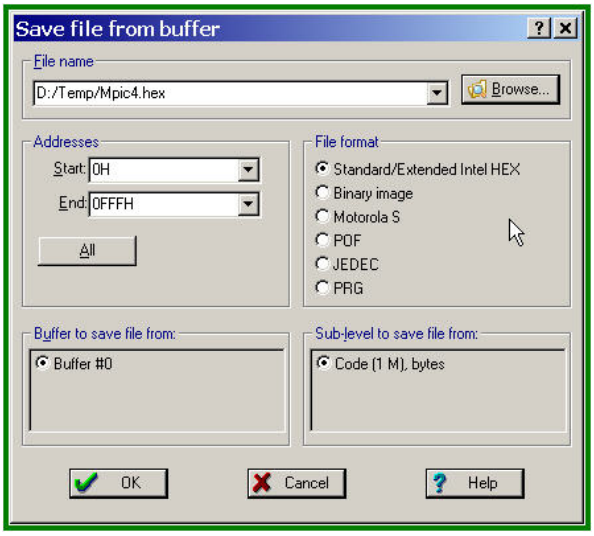
When loading a file to specific addresses in ***Intel HEX, Motorola S*** and ***PRG*** formats you can offset the placement of the load file in the buffer. Just type the offset value into the ***Offset for loading addresses*** box. For example, if data in a HEX file are located at

addresses 100h…0FFFFh and the offset is 0F0000h, then data from the file will be loaded to the buffer at addresses 0F0100h…0FFFFFh.

To re-load the most recent file loaded to the programmer's buffer, just click the ***Re-load*** icon or press Ctrl+F2. This will allow you to avoid repetitive settings while you are working with the same file.

## Saving Files from Buffers

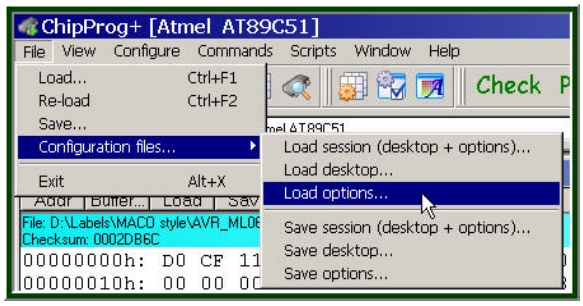
When you read from a pre-programmed memory device you may want to save the information on a disk of your computer. Or you may want to save a file after it was modified in the buffer.




Click the Save icon or press Ctrl+F1 to open the saving dialog. Here you can type in or browse for the destination file's name and path, and you can select the file format, the buffer and any sub-levels to be saved. You can also specify the start and end addresses of the file to be saved. By default the program offers to save a full space for each existing sub-level, but you can narrow down the space to be saved by setting the start and end addresses as needed.

## Saving and Loading Configuration Files

To minimize the scope of configuration operations after starting, the ChipProg+ software creates and stores several configuration files that represent the programmer's status.

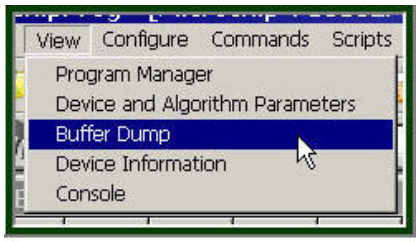



Configuration files can also be saved at any time through the commands appearing when you move the mouse cursor over the ***Configuration files*** line of the ***File*** menu (see the picture at the left). It is possible to keep several sets of configuration files for different purposes and then load them "on the fly." There are two types of configuration files that can be manipulated independently:

- A desktop file that keeps information on display options and screen configuration; it keeps all information on the position, size, colors, and fonts of all debugger windows.
- An options file that remembers the target device's type and other options.

Before completion of its session, the ChipProg+ program writes the session file to the current directory in order to know what desktop and options file should be loaded the next time it starts. Desktop and options files are saved in the same folder from which they were last loaded. Before the ChipProg+ program exits, the history file is written to the current folder. The history file keeps a history for all input lines of all the ChipProg+ dialogues. You can save and load these configurations separately or in combination.

# Chapter 7 The 'View' Menu and Programmer Windows

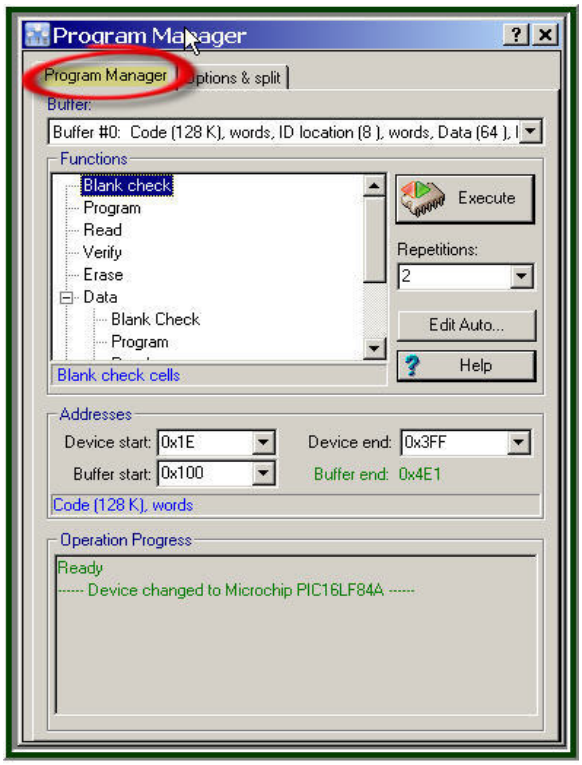



Menu 'View'

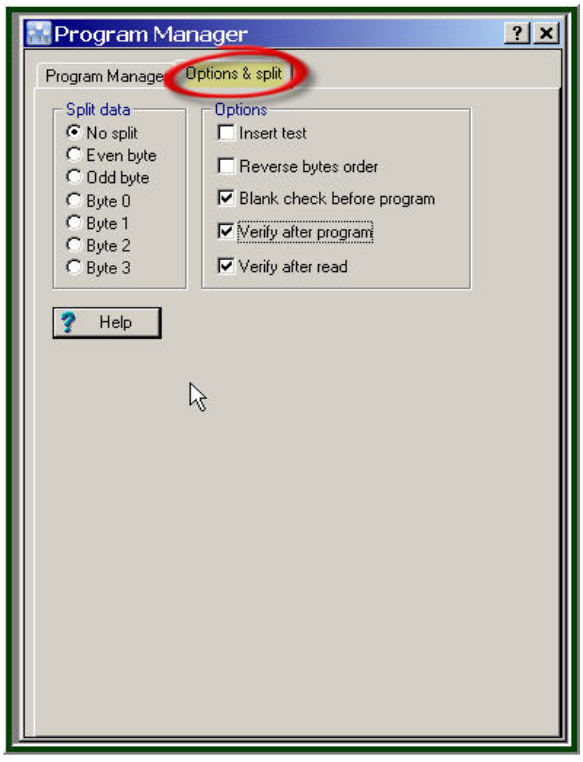
This menu includes the commands for opening particular windows within the programmer's main window. The picture here displays what windows can be open.

## 'Program Manager' Window

The ***Program Manager*** window is the main window of the ChipProg+ software shell. It serves for launching all operations with a target device: reading, writing, etc. and for




Program Manager Controls

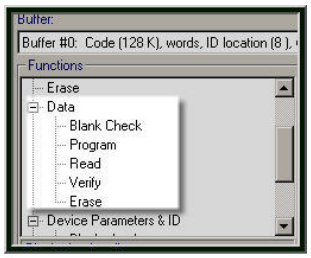



Program Manager Options

setting major parameters to control these operations. The window has two tabs – Program_Manager and Options & Split – that open two different dialogs (see below).

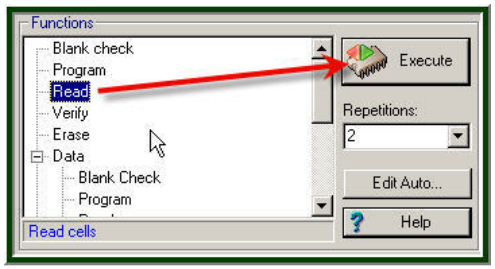
### *Execution of Programming Functions*

The ***Buffer*** box below specifies the name of the buffer that will interact with the target device under control of the commands shown in the ***Functions*** box. To enter the buffer name just type it in or pick one from the history list.




Expanded Function Tree

The ***Functions*** box displays all the commands available for a selected target device. In addition to the commands that are common for all target devices (***Blank Check***, ***Read***, ***Program***, ***Verify***, ***Erase***), the ***Functions*** tree may include the commands specific for a particular selected device only. Some commands in the tree can be grouped; such groups by default are shown collapsed. The picture here displays all the commands for the ***Data*** group belonging to the ***Function*** tree for a selected device (here it is a Microchip PIC16LF84A).




Executing of Programming Functions

To execute a command you can either double click on its line in the ***Functions*** window or highlight it and click the Execute button. To execute a command from a group, expand the group, highlight the command and double click on it or click the Execute button.


Operation in progress

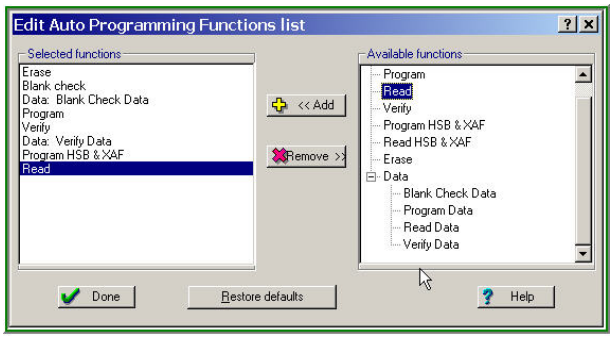
During command execution you will see a blue running bar in the ***Operation Progress*** field.

When ChipProg+ executes a command it replaces the Execute button by the Stop button. You can interrupt an operation during execution by clicking the Stop button.

By default the programmer executes each operation once. However you can set a number of repetitions for each current command by entering the number of consecutively executed commands into the ***Repetition*** box.

## *Auto Programming*

***Auto Programming*** is the last function in the command tree. This term is used for calling a batch of elementary functions (commands) to be executed in a certain order, usually to automate a complex programming procedure. The most typical set includes: checking to make sure that the target device is blank and can be programmed; programming the device from an active buffer; and verifying that the device was programmed properly. However, you may want to customize the batch of functions. To program the chain of commands click the Edit Auto button and the dialog shown here will pop-up. Then you can edit the batch by adding/removing any command to/from the command list.




Adding a Command to the Auto Programming Function List

A selected function can be inserted precisely where you want it. To insert a function, first highlight the function in the left pane (***Selected functions***) after which you want a new command to be inserted. Then highlight the command to be added in the right pane (***Available functions***) and click the Add button. The picture above displays the addition of a ***Read*** command to the tail of the auto programming function list. The result is that the device will be read to the buffer after programming.

To remove a command from the batch, highlight it in the pane ***Selected functions*** and click the Remove button.

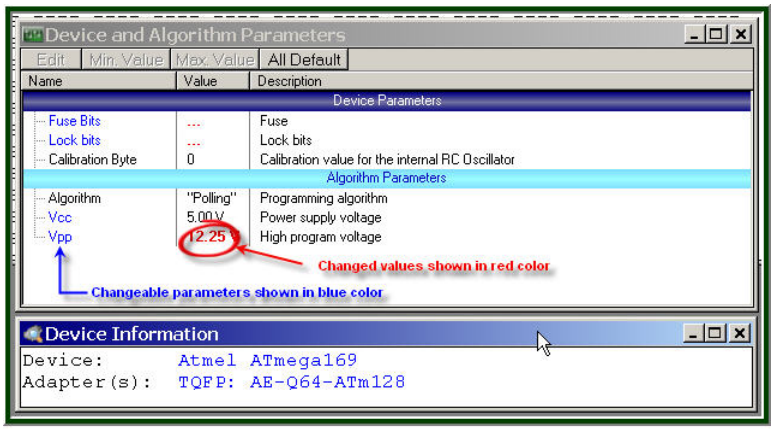
## *Controlling Source and Destination Addresses*

Usually the process of writing into a target device is a matter of moving an elementary piece of information (usually a byte) from buffer cell #0 to the device cell with physical address #0, and then incrementing both addresses repeatedly until the full chip is programmed. Similarly, when reading from a device to the buffer, the information flows in the opposite direction, but the iteration process proceeds in the same way, until the complete chip is read out. In addition to transferring the entire file, you can write or read the device partially. In the ***Addresses*** field you can set any start and end addresses of the target device and the buffer's start address. The screen shot here shows the setting for reading a full chip from the address 0h to the address FFFFh to the buffer with the address offset of 80h; i.e., the device image in the buffer will be placed in the buffer range between addresses 80h and 1007Fh.


Buffer and Target Chip Addresses

## 'Device and Algorithm Parameters' Window

The ***Device and Algorithm Parameters*** window is intended to display and prepare (where possible) the device's internal parameters and settings, which can then be programmed into a target device by executing the ***Program*** command in the ***Program Manager*** window.




Device and Algorithm parameters

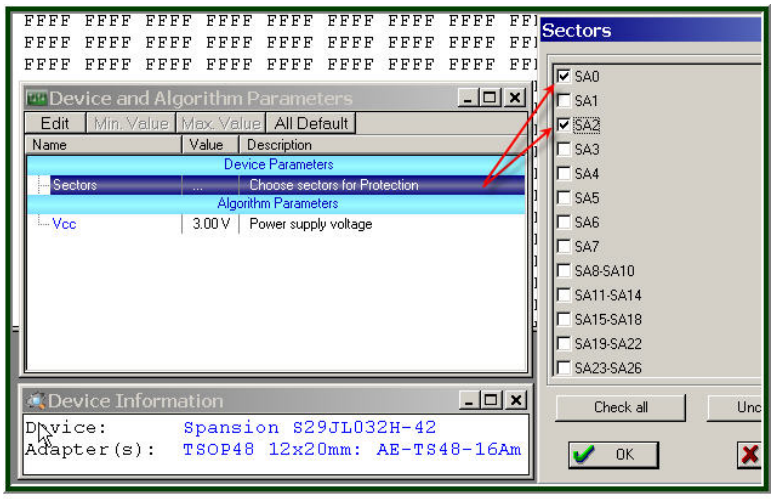
The parameters displayed in this window are split in two groups: ***Device Parameters*** and ***Algorithm Parameters***. The groups are separated by a light blue stripe.

The top group includes parameters that are specific for each selected device: sectors for memory devices, lock and fuse bits, configuration bits, boot blocks and other controls for microcontrollers. Not all of these are listed in the screen shot above. It is impossible to specify absolutely all features that may appear in future devices, and, therefore, new parameters for these new devices. Usually these parameters are combinations of certain bits in a microcontroller's Special Function Registers (SFRs). Some of these SFRs can be set in ChipProg+ buffers in accordance with device manufacturers' data sheets. But setting the parameters via the ***Device and Algorithms Parameters*** window is much easier and more intuitive.

The bottom group includes parameters of the programming algorithm for a selected device – including the algorithm type and the programming voltages.

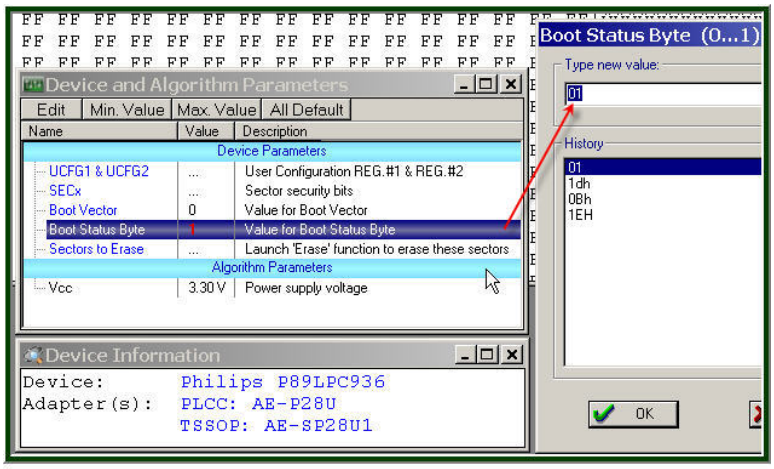
The window is separated into three columns: 1) name of the parameter, 2) its value or setting and 3) a short description. Names of the editable parameters are shown in blue; other names are shown in black. Default values in the ***Value*** column are shown in black, but if the user has changed the parameter, then the new value is shown in red. If the value is too long to display in a limited space it is represented as three dot signs ('…'). If these dots are red it means that the parameter has been edited.

In order to edit a parameter, double click its name with the left mouse button. An appropriate prompt box will pop up. See some examples below.



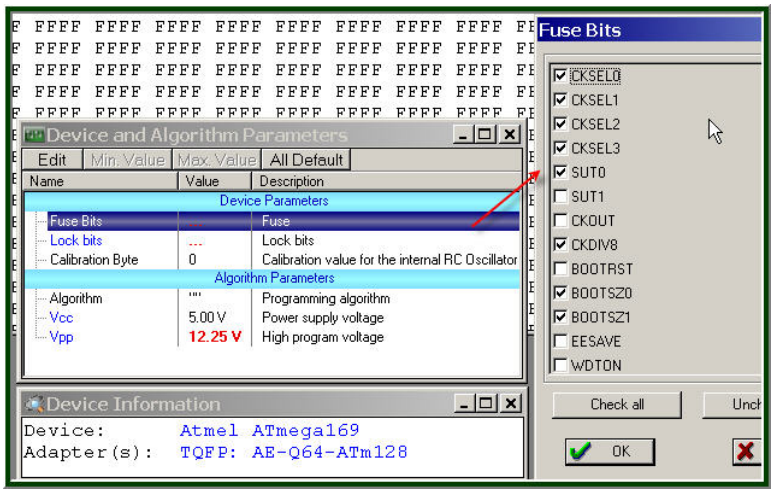

Sector Protection for Flash Memory Device

The picture at left shows how to protect two sectors, SA0 and SA2, of the Spansion flash memory device, which supports such a feature. Just check the boxes to protect certain sectors of the device against overwriting.

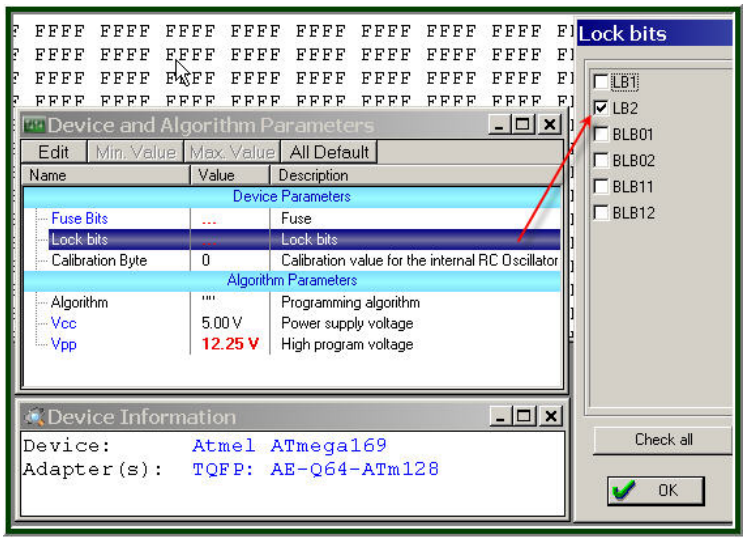



Setting Boot Status Byte

This picture shows how to program the ***Boot Status Byte*** of the Philips P89LPC936 microcontroller. Just type the value into the prompt box or pick it from a history list (some values in this list shown here are inappropriate for P89LPC936).

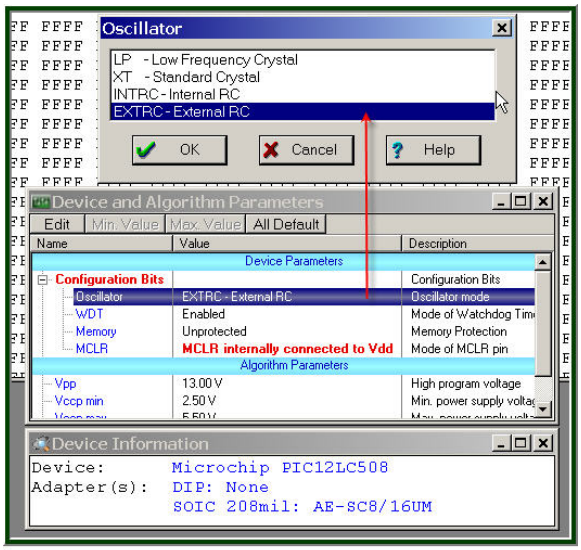



This picture shows how to choose fuses for the Atmel Atmega169 microcontroller. Check the boxes for those fuses you wish to blow when the device is programmed.



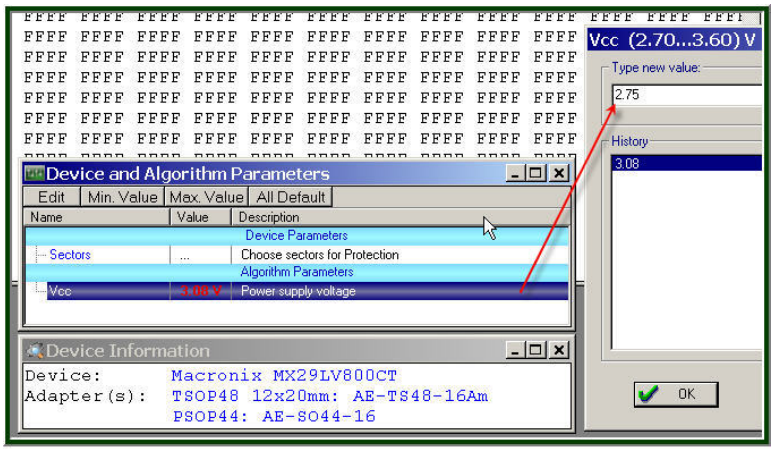

Programming Lock Bits

This picture shows how to preset lock bits for the Atmel Atmega169 microcontroller. Check the boxes for those lock bits you wish to set when the device is programmed.




Programming Oscillator Mode for PICmicro

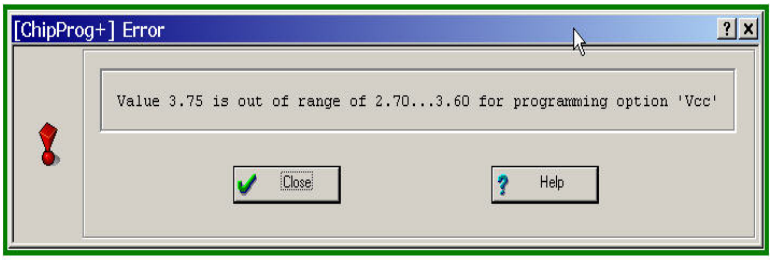
This picture illustrates how to choose an oscillator mode for the Microchip PIC12LC508 microcontroller. Click on the parameter name, select one of four available modes and click OK to fix the choice. The device will be programmed to support the oscillator mode you have set here.



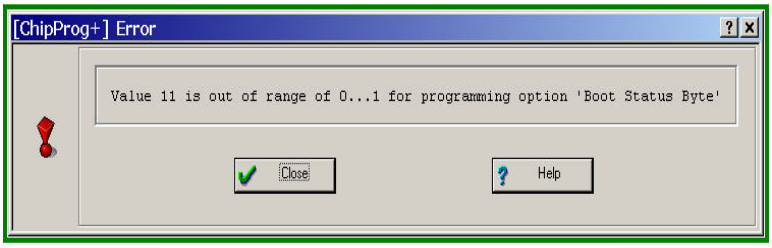

Editing Vcc Voltage for Flash Memory

This picture shows how to edit the Vcc voltage for programming a Macronix flash memory device. The manufacturer allows Vcc in the range 2.7 to 3.6V. By default the value is 3.0V. Type in any value in the permitted range or pick it from a history list. Since device manufacturers guarantee

that their devices operate properly in limited voltage ranges, the programmer software prevents the setting of an out-of-range value and issues appropriate warnings. When possible, the programmer issues similar warnings whenever a user attempts to set an incorrect value for any parameters. See the examples of such warnings below.




Error: Incorrect Vcc setting



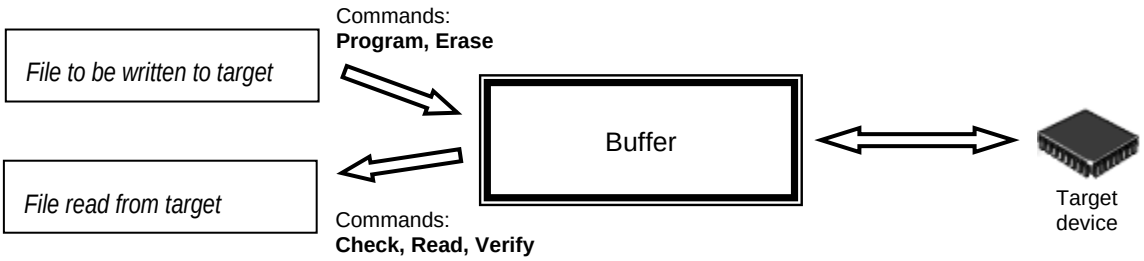

Error: Attempt to set incorrect boot status byte

***IMPORTANT NOTE!***

Any changes in the 'Device and Algorithm Parameters' window do not immediately cause corresponding changes in the target device. Parameter settings made within this window just prepare a configuration of the device to be programmed. Physically, the programmer makes all these changes only upon executing an appropriate command from the 'Program Manager' window.

## 'Buffer Dump' Window

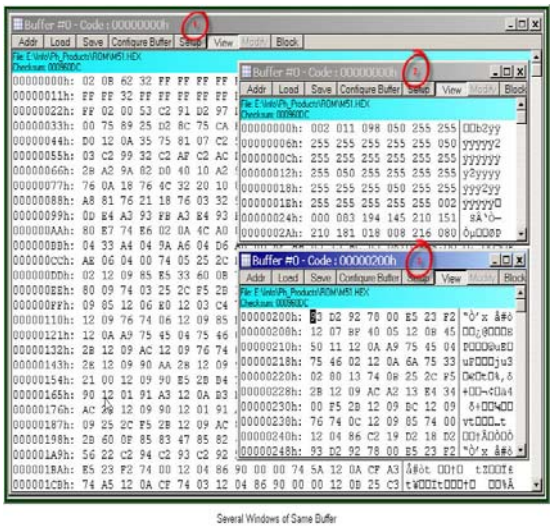
The memory buffer is the part of the computer memory that represents an intermediate link layer between a file, to be written to a target device or read from it, and the device itself.




ChipProg+ supports a flexible buffer structure:

- You can create an unlimited number of buffers. The number of buffers that you can open is limited only by the available computer RAM.
- Every buffer has a certain number of sub-levels depending on the type of target device. Each sub-level is associated with a specific section of a target device's address space. For example, for the Microchip PIC16F84 microcontroller every buffer has three sub-levels: 1) code memory; 2) EEPROM data memory; 3) user's identification sub-level.

This flexible structure allows for easy manipulation of several data arrays that are mapped to different buffers. In order to open a buffer window, click on the ***Buffer dump*** line in the ***View*** menu.

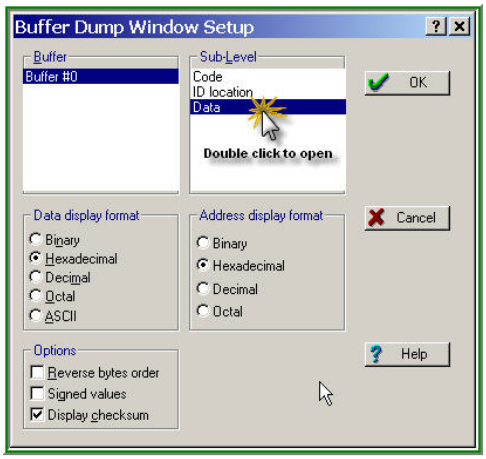


Several Windows of Same Buffer

The picture at left displays three windows representing three parts of the same buffer: the first (largest) shows the buffer contents beginning at address 0h; the second shows the same buffer contents beginning at the same address but displaying data in decimal format; a third window shows the data beginning at address 200h. The left-most column shows absolute addresses of the first cell in a row. The addresses always increment by one byte: 0, 1, 2…. Each address is followed by a semicolon (:). When you resize the window it automatically changes the addresses shown in the address column in accordance with the number of codes or data that go in one line. Some windows may be split into two panes – left pane for data in a selected format and right pane showing the same data in ASCII format. The window has a toolbar for invoking setting dialogs and commands. Right under the toolbar the program displays a full path to a loaded file and a checksum of the dump.

## *Setting up the Buffer Dump window presentation*

The Setup button on the ***Buffer*** window's toolbar invokes a dialog that sets the window presentation. If the active buffer has more than one sub-level then an appropriate buffer sub-level window can be opened (or activated if it is already open) by double clicking the sub-level name (***Data*** sub-level on the picture below). Here you can set the presentation formats for data and addresses for your convenience. By default both formats are hexadecimal.

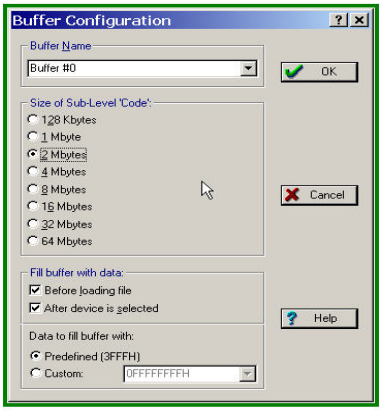



Buffer Dump Window Setup

The ***Reverse byte order*** option is helpful when your are programming a 16-bit EPROM for a microcontroller, for which a C compiler or linker generates 16-bit output files in which the most significant bytes are located at the lower address of the word. Many C compilers for 8051 microcontrollers generate such files. Check this option to program a 16-bit EPROM with the byte order reversed.

Check the ***Signed value*** option to display data as positive or negative figures. If you wish to watch the buffer checksum check in the ***Display checksum*** box.

## *Configuring a Buffer*



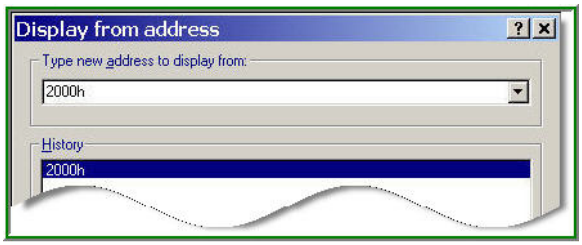

Buffer Configuration

By default the first opened buffer is named 'Buffer #0'. The next buffer gets the name 'Buffer #1', and so on. By default each buffer has a minimal size of 128K RAM in a PC and by default the program initializes this memory with a predefined value (usually 0FFh). You can customize these buffer settings. Click the Configure_Buffer button on the ***Buffer*** window toolbar – this opens the dialog at the left.

In this dialog you can name the buffer as you wish – just type any name into the ***Buffer Name*** box. Check a radio button to assign an appropriate memory size for the code or data to be programmed into the target device or to be read from it.

Specify if you want the ChipProg+ program to initialize the buffer with a predefined data pattern or a custom mask.

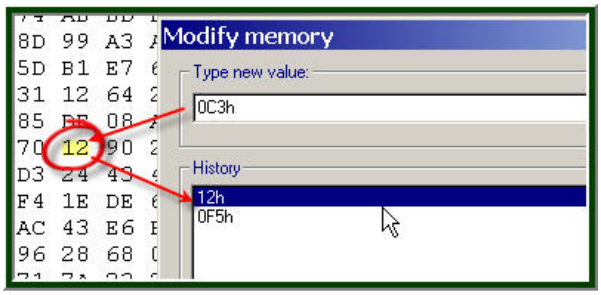
## *Setting the Buffer Dump Start Address*




New Address for Buffer Dump

To change the buffer dump's start address click the Addr button of the ***Buffer Dump*** window toolbar. You may then enter any address in the permitted range or you may pick one from the history list.

## *Editing in Buffer Dump*

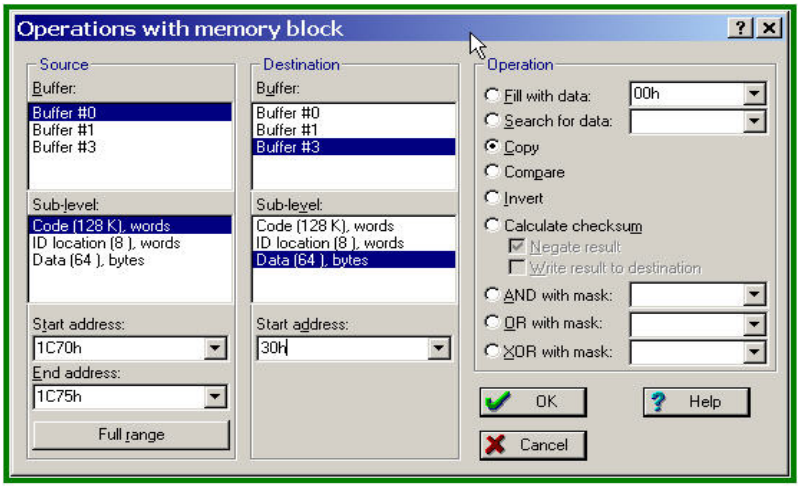



Modiffying Data under Cursor

By default the View button on the ***Buffer Dump*** window's toolbar is pressed down and the Modify button is greyed out. This indicates that the program prevents data in the buffer from accidental or intentional changes – you can only examine the buffer contents. To enable editing, click the View button to release it. Then you may overtype the value of any data under the cursor. Or, you can click the Modify button on the window toolbar to type a new value for the data under the cursor, or pick a value from a history list.

## *Operations with Blocks in Buffer Dumps*

ChipProg+ provides several convenient ways edit data in buffers by operating with blocks of data. To open the ***Operation with memory blocks*** dialog click on the Block button on the window toolbar. This opens the dialog below.




Operations with Memory Blocks

The following operations are available though this dialog:

Within one buffer (***Source***)

- Fill a part of the dump with a specified value
- Search for a particular value or symbol
- Invert the byte order within words in a selected block
- AND, OR or XOR data records within a selected block with a specified value and write the results to specified locations.

Within one buffer (***Source***) or between two buffers (***Source*** and ***Destination***):

- Copy a data block from one buffer to another or to another place in the same buffer
- Compare two blocks for identity of contents
- Calculate a checksum of a selected block and write the result to a specified location

### Fill with data

This operation allows filling a part of the Source buffer with the value specified in the box. Values can be entered in one of two forms:

- a sequence of numbers separated with a space, comma or semicolon. Numbers may have any legal format, for example: 254 or 0A5H or 0, 3, 66, 77h, 2;
- a character string enclosed in double quotes. The string may contain characters as well as symbolic constants in the C language format:

| | | |
|---|---|---|
| New line (line feed) | HL (LF) | '\n' |
| Horizontal tabulation | HT | '\t' |
| Vertical tabulation | VT | '\v' |
| Backspacing | BS | '\b' |
| Carriage return | CR | '\r' |
| Form feed | FF | '\f' |
| Backslash | \ | '\\' |
| Apostrophe | ' | '\'' |
| Quotation mark | " | '\"' |
| Zero character (null) | NUL | '\0' |

The symbol can also be presented by its hexadecimal value, preceded by the symbols '\x' and containing exactly two hexadecimal digits.

Here are some examples:

- “Copyright”
- "Author:\r\nJohn Smith"
- “Version: \x01”

Click on the Full range button if your source block size is equivalent to the full address range of the target device. The program will automatically fill in the ***End address*** box. Or, you can specify the buffer range that you would like to fill with some data. Specify the destination buffer name and sub-level and the start destination address. Then type the data into the box or pick them from a history list and click OK to complete the operation.

### Search for data

This operation launches a search for the data value specified in the box in a specified part of the source buffer. The rules for entering data (or value) for search are absolutely the same as for filling a buffer with specified data (see above).

### Copy

This operation copies the contents of a specified part of the source buffer to the destination area of the same or another buffer, beginning from a specified address. For example, the settings in the dialog picture above specify copying the contents of six locations beginning at the address 1C70h from the sub-level ***Code*** of ***Buffer #0*** to the sub-level ***Data*** of ***Buffer #3*** at the addresses 30h to 35h.

### Compare

This operation compares the contents of a specified part of the source buffer with contents of the destination area of the same buffer or another existing buffer, beginning from a specified address. If compared blocks mismatch, the programmer issues a warning that displays the mismatched data and their addresses, and then prompts to allow continued comparing.

### Invert

The operation inverts data in a specified memory block. By “invert” is meant the changing of each binary 1 to a binary 0 and vice versa.

### Calculate Checksum

If the ***Display checksum*** box is checked in the ***Buffer Setup*** dialog the programmer always displays the checksum for a full buffer at the top of the buffer window. In addition, you may calculate a checksum of the data within a specified block of the source area of memory.

Different algorithms are used for checksum calculation in programmers and in embedded software. By default the ChipProg+ program calculates the checksum as a 32-bit value by simple addition. If the sub-level has byte organization, 8-bit values are added. If it has

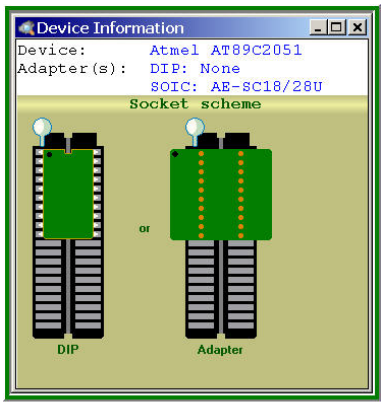
word organization (for example as Code memory of Microchip PIC microcontrollers) then 16-bit values are added for the checksum calculation.

For implementation of more-complicated checksum algorithms use the ChipProg+ embedded Script file language (read more about Scripts in an appropriate chapter of this guide).

You may negate the checksum by checking the ***Negate*** box. This will subtract a calculated checksum from zero. This method of checksum calculations is used by some developers.

If the ***Write result to destination*** box is checked, then clicking OK writes the 32-bit result into a destination buffer location at the specified ***Start Address***. If this option is unchecked, ChipProg+ just displays a checksum in a pop-up box.

## 'Device Information' Window



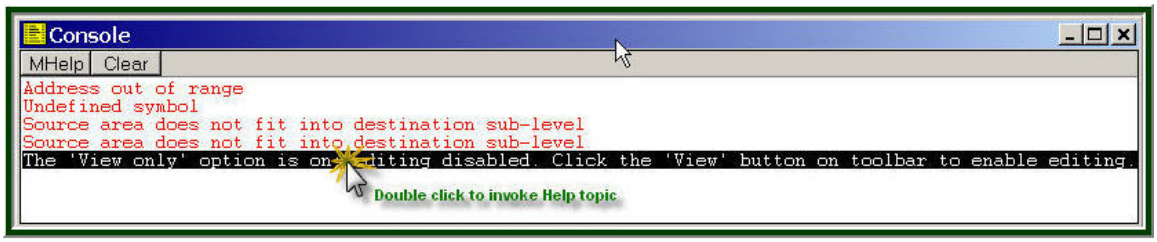

Window 'Device Information'

This window displays the type of selected target device and a list of programming adapters that fit all available packages for the selected device. For example the picture here shows only one Phyton adapter available for a SOIC package of the selected AT89C2051 Atmel microcontroller – 'AE-SC18/28'.

The pictogram at left shows the correct insertion of a DIP-packaged 20-pin AT89C2051 device into a 40-pin ZIF socket. The pictogram at right shows how a 20-pin SOIC-DIP adapter AE-SC18/28U should be inserted into the programmer ZIF socket.

## 'Console' Window

The ***Console*** window displays the ChipProg+ error messages and what-to-do prompts. It stores messages even if it is closed. You can open it at any time to view the last 256 messages, and get help for any of them.




Error Messages and Prompts in the Console Window

The current message line is highlighted. You can move the highlight by using the mouse (a single click moves the highlight bar to the current mouse pointer position) or by using the arrow keys of the keyboard. Double click on the highlighted line to invoke the ChipProg+ context-sensitive Help topic appropriate to the message contents.


Example of Pop-up Error Message

By default, while the ***Console*** window is closed, all error messages and prompts are displayed in pop-up boxes. See an example here. Click the Help button to invoke the ChipProg+ context-sensitive Help topic associated with the error, or click the Close button and continue after correcting a parameter error.

When the ***Console*** window is open the programmer automatically sends error and information messages to it, without popping up individual message boxes. In many situations it is more convenient to use the ***Console*** window because: a) you can see as many as the last 256 messages and get help for any of them by a mouse click on the message line, and b) no user response to the issued message is required. However, you may prefer to save screen space by closing the ***Console*** window and getting pop-up messages in response to each incorrect action or error received from the programmer. To be sure that you do not miss error messages, go to ***Main*** menu > ***Configure*** > ***Display Options*** and check the box ***Message Box Display***. Then all messages will be displayed as dialog message boxes, even when the ***Console*** window is open.

# Chapter 8 The 'Configure' Menu and Main Programming Settings

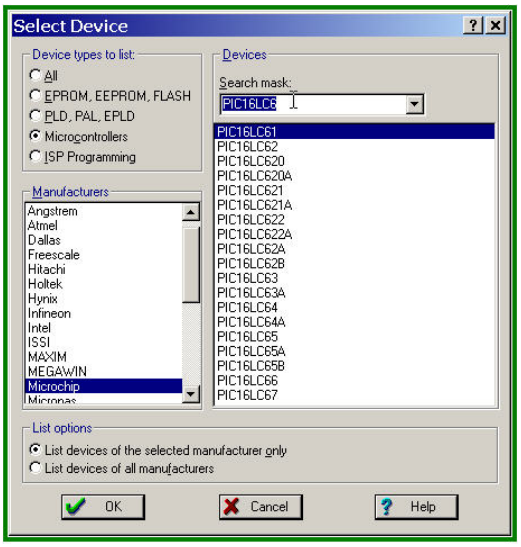
This menu is a control center for configuration dialogs of all kinds. See the menu tree.


Configure Menu

## Selecting a Target Device

Click on the ***Select device*** line of the ***Configure*** menu or click on the Select_Device button on the main toolbar. The dialog below will open.

First choose the type of device and the manufacturer in order to narrow down the list. All devices are grouped in a few categories:




Target Device Selection

- *EPROM, EEPROM, FLASH* – all kinds of parallel and serial programmable memory devices: flash, UV-erasable, one-time-programmable (OTP), etc.
- *PLD, PAL, EPLD* – all types of programmable logical devices.
- *Microcontrollers* –all kinds of embedded microcontrollers with programmable memory: flash, UV-erasable, one-time-programmable (OTP), etc.
- *ISP Programming* – all devices that can be programmed in-target via a special cable adapter, instead of to be placed into the ChipProg+ ZIF socket.

To expedite finding a target device from a long list, type the first letters of the part number for a chosen manufacturer in the ***Search mask*** box. This narrows down the list of devices in the ***Device*** field and expedites finding the device to be chosen. Keep the radio button ***List devices of the selected manufacturers only*** checked if you wish the device list to exclude devices produced by other manufacturers. This will make the device search easier.

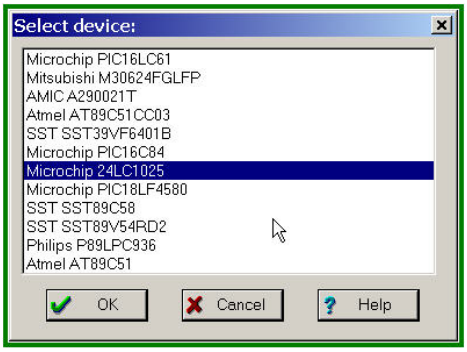
Devices' part numbers are shown in the list from top to bottom in an alphabetical order and in the order of increasing numbers (i.e., for example PICE16LC628, PIC16LC63, PIC16LC63A, etc.).

***IMPORTANT NOTE!***

However all the devices programmed “in-system” are shown below the list of those programmed in the programmer socket. If the same device can be programmed in both modes: in socket and in-system you can find the first placement in the top list and a second one in the bottom list. To avoid a wrong programming mode selection it is highly recommended to always check the ***ISP Programming*** radio button when using ChipProg+ for programming any device “in-system” via a special cable.

## Picking a Device from History List

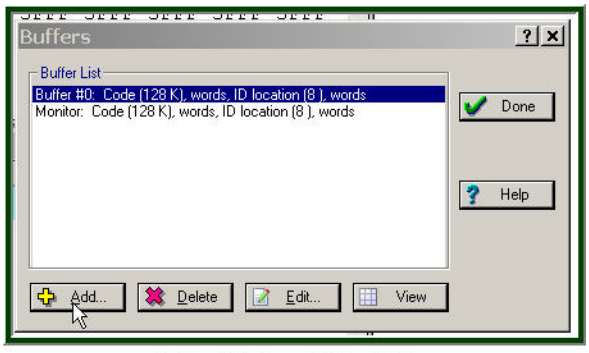
Click the ***Device selection history*** line of the ***Configure*** menu and you will open a list of recently selected devices. The most recent device is always on top of the list. Click OK if that’s what you want. To choose a different device, highlight it and click OK to pick it from the history list.




History List of Selected Devices

## Adding and Deleting Buffers

Click the ***Buffers...*** line of the ***Configure*** menu and you will open the ***Buffers*** dialog; it allows you to reconfigure existing buffers, to remove some of them, or to add new ones to the list.




Adding, Deleting and Editing Buffers

Click the Add button to open a new buffer. This pops up the new buffer properties dialog that was described earlier in the chapter “Configuring a Buffer.” Accept the default properties or enter your own convenient name, and select the buffer size you want in the range 128K to 64Mbytes.

To delete a buffer, highlight its name and click the Delete button.

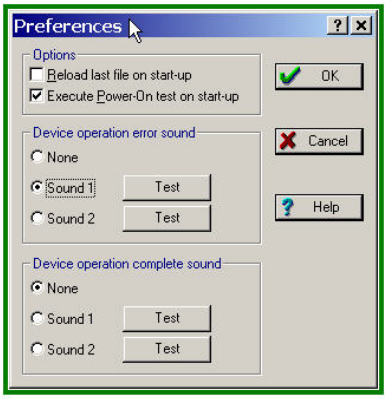
To change any properties of an existing buffer, highlight its name and click the

Edit button. This pops up the same dialog that's used to add a new buffer. You will be able to change the buffer's name and other properties.

To open a buffer window for viewing, highlight the buffer name and click the View button.

## Configuring Preferences

Click on the ***Preferences*** line in the ***Configure*** menu to invoke a dialog that allows you to preprogram some procedures framing major commands executed by the programmer.




Sounds and Other Preferences

Check the ***Reload last file on start-up*** box if you are working with only one file and want to save time on loading it repeatedly. In other cases this option makes no sense and by default it is off.

By default the programmer conducts a self-testing power-on procedure every time it starts up. Phyton highly recommends keeping this option always on. However it can be switched off.

The programmer is able to generate different tone signals on error and on successful completion of programming operations. You can either disable tones altogether or select one of two preloaded tones for each event. Clicking the Test button generates a tone signal so you can select one you like.

To complete setting the preferences do not forget to click OK in the dialog.

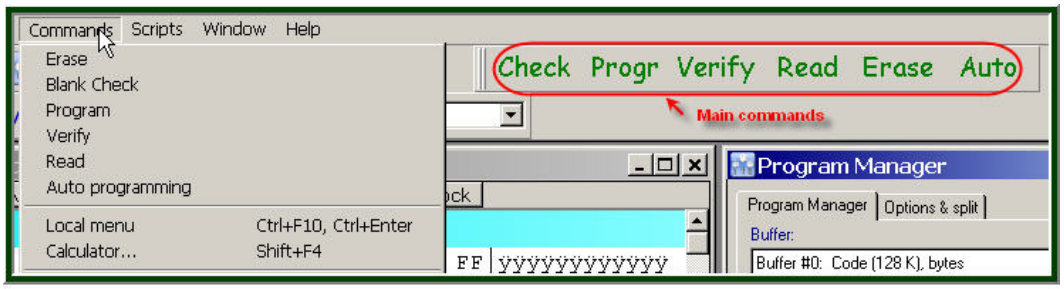
## Configuring the Programming Environment

Click the ***Environment*** line in the ***Configure*** menu to invoke a dialog that allows customizing such elements as fonts and colors in windows, hotkey combinations, toolbar buttons and some other GUI settings. Most of these settings were described in the chapter "Toolbars and Graphic User Interface Customization".

The ChipProg+ software is equipped with a powerful embedded editor that is used for editing script files. Click on the ***Editor Options*** line in the ***Configure*** menu to set up the editor. Script file concepts, methods of use and the script editor are described in the "Scripts" chapter.

# Chapter 9 The 'Commands' Menu

This menu invokes main commands (or functions) that control the programming process, as well as some service commands. A horizontal line separates the two groups of commands. You can execute a command by a single click on the highlighted line of the menu. For your convenience, the major commands, which are common for almost all target devices and frequently used (***Check***, ***Program***, ***Verify***, ***Read***, ***Erase***, and ***Auto***), are duplicated by large buttons on the main programmer toolbar.




Menu Commands

As mentioned earlier, you can also invoke the main commands from the ***Program Manager*** window; choose the option that's most convenient for you.

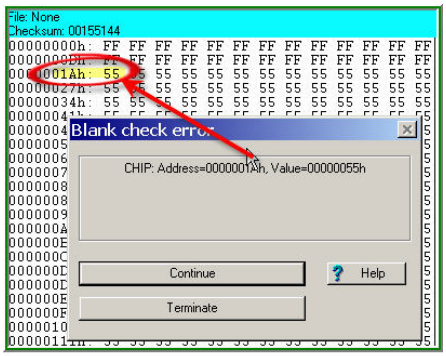
If the Check or Erase buttons are blocked by the programmer ("greyed out") then a selected device can be programmed even if it's not blank; the information in it will be overwritten. In this case, execution of the ***Program*** command will automatically erase previously written data before writing new data to the same cells.

## 'Check' Command

New devices usually come from manufacturers with no information inside; i.e., they are "blank", unless they are programmed at the factory, pursuant to special orders. For many devices the "blank" status means that they are filled with 'FFh' data (some old devices come filled with zeros). The blank status means that the device can be physically programmed. However, even if you work with new devices it is best to check the target device by executing the ***Blank*** command before programming to make sure that it is really blank.

The command checks the contents of a target device in the range of addresses specified in the ***Program Manager*** window. By default it starts at address zero (or from a specified ***Start*** address) and continues to increment addresses until the highest (or specified ***End***) address is achieved. If the device is not blank the programmer issues an error message that displays the non-blank address and its contents.

The picture above shows the error message on the first address at which the programmer has read data different from 'FFh'. As you can see the range from 0 to 19h is blank; i.e. it



Blank Check Error

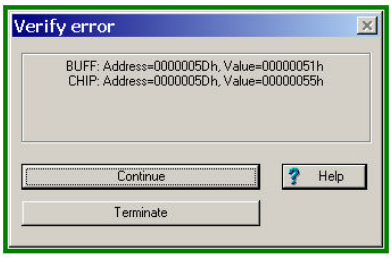
is filled with ‘FFh’. The cell at address 1Ah contains ‘55h’. The device should be erased or, if that is impossible, it should be discarded.

## ‘Program’ Command

This command writes the contents of the buffer into the target device’s cells. It programs a target device in a range of addresses specified in the ***Program Manager*** window; it picks source information from the buffer beginning at the ***Buffer start*** address. By default the program increments addresses from zero (or from a specified ***Start*** address) until the highest (or specified ***End***) address is achieved. If a device cell is not blank or is physically damaged, the programmer fails to program this cell and issues an error message that displays the failed address and the contents of the buffer and device cell at the shown address.

## ‘Verify’ Command

This command reads information from a target device in a range of addresses specified in the ***Program Manager*** window and compares it byte-by-byte with the corresponding information in the buffer beginning at the ***Buffer start*** address. By default it increments addresses from zero (or from a specified ***Start*** address) until the highest (or specified ***End***) address is achieved.



Verification Error

If the content of the device’s cell is not equal to the corresponding data in the buffer, the programmer issues a verification error message. The picture here shows such a message. The device cell with the address 5Dh contains the data 55h while the buffer location with the same address contains 51h.

## ‘Read’ Command

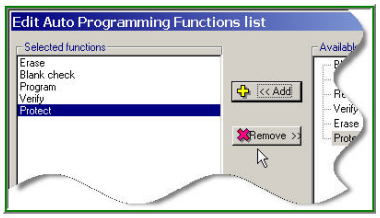
This command reads the content of the target device’s cells into an active buffer. It reads from the device in a range of addresses specified in the ***Program Manager*** window. By default reading starts from zero (or from a specified ***Start*** address) and goes until the highest (or specified ***End***) address is achieved. The information is stored in the buffer beginning at the ***Buffer start*** address. You can observe the information in the buffer window; you can also edit and save it on a computer disk.

## ‘Erase’ Command

This command erases all cells of the target device installed in the programmer socket. The device’s cells will be filled by some default value specific for each type of device (usually ‘FFh’). Some electrically erasable memory devices cannot be erased, but information can be written over the information in their cells. If such a device is selected, the programmer rejects the command and issues a warning. The large Erase button grays out when such a device is selected.

When you try to erase a pre-programmed one-time programmable (OTP) device, or a device that is protected against erasing, the programmer issues a warning that indicates the address at which the data cannot be erased.

## ‘Auto Programming’ Command



Command Batch for Auto Programming

When you invoke this command you actually start a batch of consecutively executing commands pre-selected in the ***Program Manager*** window (the Edit Auto button). For example, on the picture here the commands are shown in the order of execution from ***Erase*** down to ***Protect***. Thus when you start ***Auto Programming***, the programmer will first erase data in the target device and check if the device is blank; then it will program the device, verify the programming and then it will protect the device against reading.

## ‘Local Menu’ Command

This command invokes a pop-up local menu of the most-frequently used commands associated with an active window. Each window has its own set of such commands. The same local menu can also be invoked if you place a mouse cursor within a window and click the right mouse button.

## ‘Calculator’ Command

This command invokes a convenient embedded calculator that helps to evaluate expressions and convert resulting values from one supported number base to another.

# Chapter 10 The 'Scripts' Menu

The programmer software is equipped with an embedded script language for creating and performing custom-made routines that allow automation of complex programming operations. Among these are:

- Loading files
- Setting up programming options
- Programming
- Data verification
- Checksum calculations
- Data analysis
- Checksums and branching actions upon the results
- Manipulation of data in data buffers
- Displaying messages in the 'Console' window
- Displaying graphical data in special windows
- Creation of user menus
- Storing data in files

Script files are written in a C-type language. Almost all C constructions are supported, except for structures, conjunctives and pointers. Many built-in functions are available, such as printf(), sin(), and strcpy(). A full description of the programmer script language is published in the 'Scripts' chapter of the ChipProg+ on-line Help.

A script is a file with extension *.cmd*. It can be created and debugged right in the programmer's software shell by means of an embedded editor and a debugger. Then it can be started for execution. Several examples of scripts supplied with the programmer are placed in the '\Examples.cmd' folder.


Menu 'Scripts' with Script Examples

The ***Scripts*** menu invokes all the commands associated with script controls. A horizontal line divides the menu commands into two groups. The commands above the line represent scripts supplied with the programmer as well as those prepared by a programmer user and placed in the '\Examples.cmd' folder. You can start the execution of any script from this group by double clicking its name.

The commands below the line invoke scripts or their properties for editing, debugging or control.

## Creating and Editing Scripts

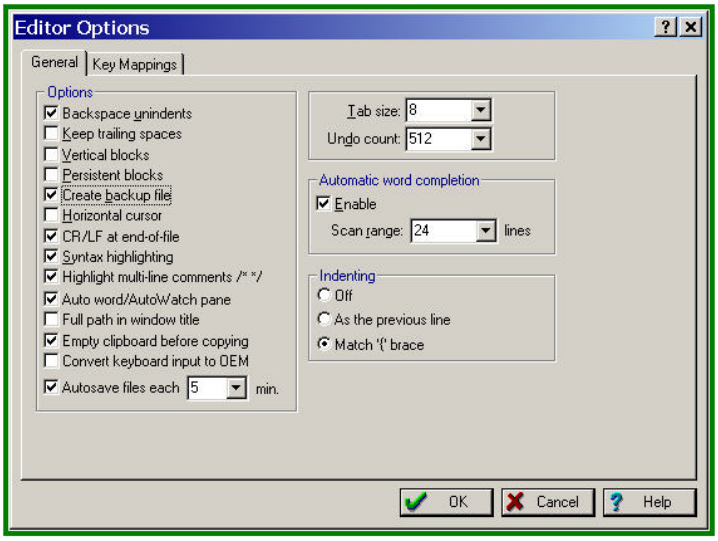
To create a new script, click the ***New Script Source*** menu line in the ***Script***. tree Then you can type a new script in a pop-up window, following the rules for making scripts for ChipProg+. When you finish and attempt to close the window, the program offers to save the script source in a location on the PC's hard disk. Use the *.cmd* extension as part of the name. Then you can always invoke this script for editing and debugging.

You can also open a window to create a new script by selecting the ***Editor window*** menu entry and then selecting ***New***.

To open an existing script for editing, select ***Editor window***, then select ***Open***, and browse for a script. Alternately, you can just type its name into the ***File name*** box or pick a script from a history list.

The embedded editor supports an operation repertoire that includes entering text, copying, searching, etc.

### *Setting Editor Options*




Editor Options

The ChipProg+ software is equipped with a powerful embedded editor that is used for editing script files. Click on the ***Editor Options*** line in the ***Configure*** menu to invoke the dialog below. All editor options applicable to all its windows are set up here. Functions of most of the settings in this dialog are clear from their names.

#### Indenting

If this option is off, each new line of the entered text will start from the first character of the column. If you check the ***As the previous line*** radio button, the editor will automatically set the cursor at the same character position whenever you complete a current line by pressing the Enter key on the PC.

#### Backspace unindents

When this option is checked (enabled) and the insert mode is active, and the cursor is positioned at the first character of a line, then if you press the Backspace key, any preceding blank spaces on the line are deleted and the first character of the line appears in the first character column.

When this option is checked and the overwrite mode is active, and the cursor is positioned at the first character of a line, then if you press the Backspace key, the cursor moves to the first non-blank character column while the first character of the line remains in place.

When this option is unchecked (disabled) and the insert mode is active, and the cursor is positioned at the first character of a line, then if you press the Backspace key, the cursor and all the characters in the line move one character column to the left.

When this option is unchecked and the overwrite mode is active, and the cursor is positioned at the first character of a line, then if you press the Backspace key, only the cursor moves one character column to the left while all the first characters of the line remain in place.

### Keep Trailing Spaces

If this box is checked any trailing spaces on a line are saved in the buffer and on the disk during editing. Otherwise, trailing spaces are not saved.

### Vertical Blocks

The process of writing and editing script source files differs slightly from writing and editing other types of text documents. The basic difference is that program text is formatted and has a more-or-less regular structure.

- Text is divided into lines
- These lines often have the same structure; for example, they are divided into fields and every field starts at a specific position
- The same fields are located in different lines one under another

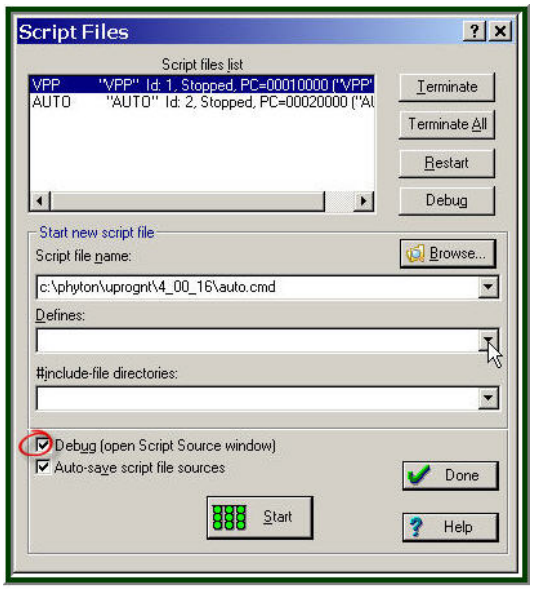
For example:

```
Timer0      .DB 2
Timer1      .DB 2
Int0        .DB 1
Int1        .DB 1
```

ChipProg+’s built-in editor has features for working with formatted as well as non-formatted text. The difference first of all relates to block operation: "standard" blocks as well as vertical and string blocks are supported. Automatic indent mode is available. This mode can be turned off.

As mentioned above, when developing a program, a programmer works mainly with formatted text. "Standard" blocks that are used in most of the Windows programs are not convenient for such text. That is why ChipProg+ supports two additional block types (line blocks and vertical blocks) as well as standard blocks. It should be noted that standard blocks are turned on by default.

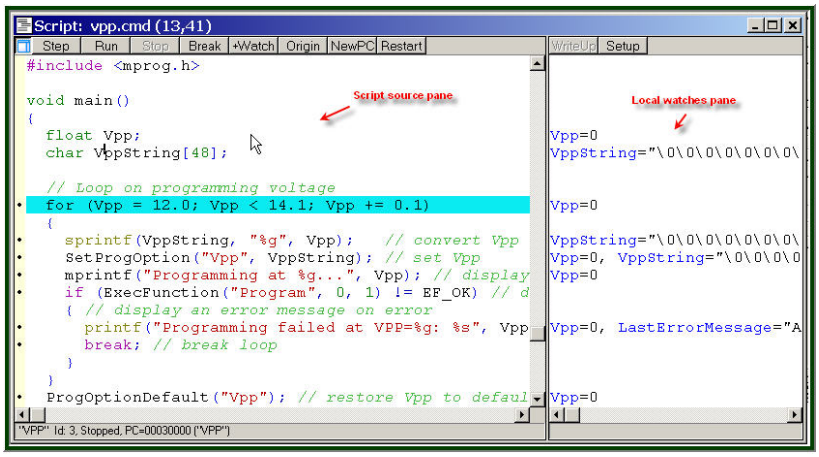
## Debugging and Running Scripts




Starting and Debugging Scripts

Click on the ***Start...*** line in the ***Script*** menu. The ***Script Files*** window to the left will pop up. To debug a script, browse for a script file name or just type it into the ***File name*** box, or pick the script from a history list. To debug a selected script check the ***Debug*** box at the bottom of the window. Click the Start button at the bottom. It will open a script window (see below).

The window below has two panes. The left one represents a source text of the loaded script and the right pane displays local variables, or watches, belonging to the lines of the script text. Each pane has a toolbar with the a few hotkeys, so you can step through the script text, start it or restart it for execution, set breakpoints, change addresses, etc.




Script Window

# Chapter 11 How to Operate the Programmer

This chapter briefly describes most frequently used operations with the programmer.

## How to Check if a Device is Blank

1. Select the target device type
2. Insert a device of the selected type into the programmer socket
3. Click the Blank button and wait for the message ***Programming … OK*** in the ***Program Manager*** window, or a warning message if the device is not blank.

## How to Erase a Device

1. If the device in the programmer socket is not blank, make sure the selected type corresponds to an actual device's label.
2. Make sure the device is electrically erasable. Some devices are not erasable; these may be programmable once, UV erasable, or over-writable – in this case the Erase button is blocked (grey out).
3. If the device is electrically erasable click on the Erase button and wait for the message ***Erasing … OK*** in the ***Program Manager*** window, or a warning message if the device is not blank.

## How to Program a Device

In order to program a device you need to perform a few consecutive operations: to load the file that you want to write to the device; edit the file (if necessary); configure the device to be programmed (if necessary); write the prepared information into the device and verify the programming.

### *How to Load a File to be Written into the Device*

1. From the ***Main*** menu select via the ***File > Load*** command.
2. Enter the source file name, select the file format, the destination buffer and addresses and click OK.
3. Wait for the message ***File loaded*** in the ***Program Manager*** window, or an error message.

### *How to Edit Information to be Written into the Device*

1. If you need to modify source data before writing into the target device, then release the View button on the local toolbar of the ***Buffer*** window to enable editing.
2. Make necessary changes in the window.

### *How to Configure the Device to be Programmed*

1. If any parameters displayed in the ***Device and Algorithm Parameters*** window can be changed by editing, their names are shown in blue.
2. Click on the name of the parameters to be changed to open an appropriate dialog. Set a new value for the parameter or check/uncheck appropriate boxes and click OK. The parameter value will change its color to red.

3. Continue for every parameter that should be changed. All preset changes will become effective in the target device only upon programming via the ***Program Manager*** programming function.

### *How to Write Information to the Device*

1. Click the Options & Split tab in the ***Program Manager*** window. Check the options you need. We recommend that you always check the ***Blank check before program*** and ***Verify after program*** check-boxes to make programming more reliable.
2. Click the Program Manager tab. Select the ***Program*** line in the ***Function*** window, and double click it to start programming. Alternatively you can do the same by clicking the Execute button or the big Program button or the ***Program*** command in the ***Commands*** menu.
3. Wait for the message ***Programming … OK*** in the ***Program Manager*** window. If an error has occurred there will be an error message.
4. Go down to the list of functions, expanding them if collapsed. Select the function and double click it to program a current parameter in the device.
5. Wait for the message ***Locking… OK***, or ***Setting… OK***, or other ***OK*** message in the ***Program Manager*** window, or an error message.
6. Continue until every parameter that was changed in the ***Device and Algorithm Parameters*** window, is successfully programmed.

## How to Verify Programming

1. Usually the programmer compares the contents of the device with the contents of the source buffer right after the device is programmed.
2. For additional verification click the Verify button on the main toolbar.
3. Wait for the message ***Verifying … OK*** in the ***Program Manager*** window, or an error message.

## How to Read a Device

1. Click the Read button on the main toolbar.
2. Wait for the message ***Reading… OK*** in the ***Program Manager*** window, or an error message.

## How to Save the Data Read out from a Device

1. After the device was read out its contents are copied into the buffer. On the local toolbar of the buffer window click the Save button.
2. In the pop-up dialog specify the destination file name, format, start and end addresses of the source (the buffer), and the source sub-level, and click OK.

## How to Duplicate a Device

1. Insert the master device, which is to be copied (duplicated) to a blank device, into the programmer socket.
2. Click on the Read button on the main toolbar.

3. Wait for the message ***Reading… OK*** in the ***Program Manager*** window. Make sure the master device content is in a current buffer.
4. Remove the master device from the socket and replace it with a blank device. If necessary, check to see if it is blank.
5. Go back to the section "*How to Edit Information to be Written into the Device"* and proceed as if you were writing a file to a device.