Phyton™

ChipProg Device Programmers

User's Guide

ChipProg-48
ChipProg-40
ChipProg-G4
ChipProg-ISP

ChipProg Device Programmers

Printed: August 2010 in (whereever you are located)

Table of Contents

1 Introduction

Phyton™

ChipProg Device Programmers

User's Guide

ChipProg-48
ChipProg-40
ChipProg-G4
ChipProg-ISP

1.1 Terms and Definitions

Terms used in the document

Target device or **Target**	The device to be programmed by a programmer either in the programmer socket or by an additional adapter or by a cable for in-system programming.
Start and **End Addresses** (of the Target device)	A range of the device physical memory for the programming operations (Read, Write, Verify, etc.).
Device packagePackage	Mechanical characteristics of the target device; ChipProg programmers enable operations on the devices packed in the DIP (DIL) packages with no additional adapters as well as on most non-DIP packed devices, including but not limited to the devices in the PLCC, SOIC, SSOP, TSOP, SSOP, QFP, BGA, QNF and other packages.
Programming socket or **Programming ZIF socket** or **ZIF socket**	A socket installed on a programmer unit or on an adapter (see below) to accommodate the target device for programming. All ChipProg models use ZIF (or Zero Insertion Force) programming sockets that allow for the temporary installation of the target device in the programmer site and easily

removing it after completing the programming procedure.-40, ChipProgChipProg

Adapter or **Package adapter** — A small transition board with dual-in-line rows of pins pluggable into the programmer ZIF socket on the bottom side and with a package-specific ZIF socket (TSOP, PLCC, etc.) on the top. The adapters for in-system programming by means of the parallel programmers are implemented as ribbon cables that connect to the target board via a special header. The adapter boards can carry passive components (ZIF sockets, pins and cables) and active components (drivers, latches, transistors, etc.). Hundreds of Phyton brand adapters as well as third party adapters are available to support devices in most types of mechanical packages.

File — In the ChipProg context the term **file** may represent: a) an image of information on a PC hard drive or other media that is supposed to be written into the target device's physical memory or b) an image read out from the target device and then stored on the disk or other media. Files in a ChipProg can be loaded from and saved on a PC hard drive or CD.

Buffer or **Memory buffer** — A memory segment, physically assigned from the computer operational memory (RAM), for temporarily storing, editing and displaying the data to be physically written to the target device's memory or read out from the device. The program allows opening an unlimited number of buffers of any size while it is not restricted by the computer memory.

Buffer layer or **sub-layer** — A buffer may have a few layers (in some topics also known as sub-layers) that are defined by a particular architecture and memory model of the target device. For example, for some microcontrollers one buffer can include the code and data memory layers (see more details below).

Buffer size — The buffers may have different sizes from 128KB to 32GB each.

Buffer start address — The address to display the buffer contents from.

Checksum — An arithmetic sum of the data located within a specified part of the buffer calculated by the programmer to control the data integrity. The program enables different algorithms for the checksum calculation and enables writing the checksum into a specified location of the target device.

Parallel or **In-socket programming** — Operations on a device being placed into the programmer's ZIF socket or into a programming adapter (opposite to the in-system programming below).

ICP or **in-circuit programming** — Programming devices mounted on the boards (in the user's equipment) via special adapter-cable connecting the programmer to the target.

ISP or **in-circuit programming** — Same as above. Programming devices mounted on the boards (in the user's equipment) via special adapter-cable connecting the programmer with the target.

ISP Mode — Mode of the in-system programming that is usually defined by the programming signals voltage or the ISP interface (JTAG, UART, SPI, etc.). Distinct ISP modes are enabled for different target devices and more than

	one mode may exist for one device.
ISP JTAG Mode	In-system programming via a JTAG interface.
ISP HV Mode	In-system programming that requires applying a relatively high voltage to the target device, (12V for example).
Project	An integrated set of information in the ChipProgUSB that completely describes the target device, properties of the data buffers, programming options and settings, list of the source and destination files with all their properties, etc.. Each project that has its own unique name can be stored and promptly reloaded for immediate execution. Usually a user creates a project to work with one type of device. Working with projects saves a lot of time for the initial configuration of the programmer every time you start working with a new device.

File - Buffer - Target structure

Buffers are intermediate layers between the data in files and the data in the target device. The ChipProg enables no direct interaction between the files and target devices. All the file operations, such as loading and saving files are applicable to the buffers only. All the physical manipulations with the target device memory content pass through the buffers as well. This is a fundamental principle of the programmer operations with data and devices

Examples of the buffer's layer structures of different devices:

1. In the Intel 87C51FA microcontroller each opened buffer includes two layers: **Code** and **Encryption table**.
2. In the Microchip PIC16F84 microcontroller each opened buffer includes three layers: **Code, Data EEPROMIdentifier locations**.

Each buffer layer can be opened for watching or editing by clicking its tab on the top of the buffer window.

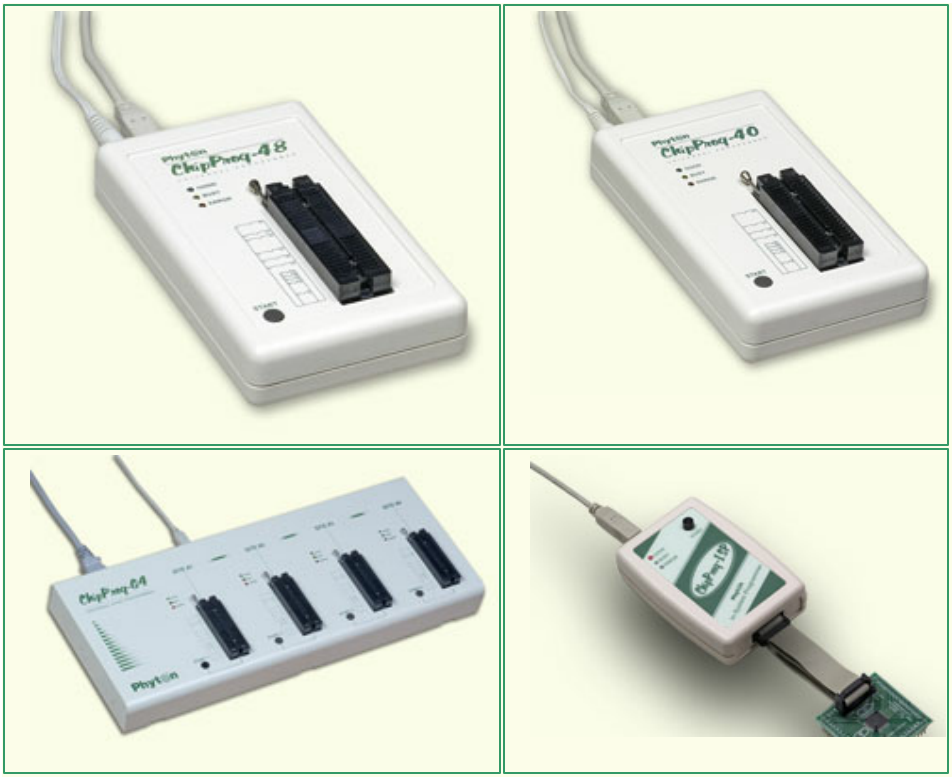
1.2 System Requirements

To run ChipProgUSB and to control a ChipProg programmer, you need an IBM PC-compatible computer with the following components:

- Pentium-III CPU or higher
- Windows XP/Vista/7 for the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers in the all modes
- Windows 98/ME for the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers in the basic mode
- Windows 2000/XP/Vista/7 for the ChipProg-G4 programmer
- Windows XP/Vista/7 for using the Application Control Interface control
- 256MB of RAM
- At least one USB port
- A hard drive with at least 200MB of free space

2 ChipProg Family Brief Description

ChipProg is a family of device programmers produced by Phyton, Inc. Microsystems and Development Tools (hereafter Phyton). All modern ChipProg models are driven via a personal computer USB ports. This line of Phyton programmers works under control of the ChipProgUSB universal software, one for all USB hosted Phyton device programnmers available now and those are planned to be introduced soon. The ChipProg programmers support thousands of programmable memory devices, including EPROM, EEPROM, FLASH, NVRAM and OTP; programmable microcontrollers and logical devices: PAL, PLD and CPLD. The family includes four models shown below: **ChipProg-48** and**ChipProg-40** (top row), **ChipProg-G4** and **ChipProg-ISP** (bottom row). New **ChipProg** models will be added soon.



ChipProg-48 and **ChipProg-40** programmers are intended for engineering and small volume manufacturing. These models allow operating on the devices before they are installed in the equipment (parallel programming) as well as on the devices already installed in the user's equipment (the method known as In-System Programming, or ISP, that uses serial data transmission into the programmable device). The **ChipProg-ISP** is a low-cost programmer for engineering, field service and manufacturing uses. The **ChipProg-G4** is a gang programmer intended for small and middle-volume production; it has four programming sockets.

The ChipProgUSB software is intuitive and easy-to-use. See the **User Interface** topics. The software package includes an embedded script language that enables the automation of many routine operations – see the .

The ChipProgUSB software runs on the IBM PC hardware platform under the control of several

Windows™ versions (see the **System requirements**).

2.1 Comparisson matrix

Programmer Model	ChipProg-G4	ChipProg-48	ChipProg-40	ChipProg-ISP
Major features				
Primarily intended for	Production and chip replication	Engineering and low volume production	Engineering and low volume production	Engineering, low volume production and field service
Method of writing / reading information	Multi-site, concurrent, parallel, in socket	Single-site, parallel, in socket	Single-site, parallel, in socket	Single-site, serial, in system
Target devices	FLASH, EPROM, EEPROM, NVRAM, MCU, PLD	FLASH, EPROM, EEPROM, NVRAM, MCU, PLD	FLASH, EPROM, EEPROM, NVRAM, MCU	FLASH, EEPROM, MCU with ISP capability only
Universality in terms of the target support	Yes	Yes	Yes	Yes
	USB, 2.0		USB, 2.0	USB, 2.0
Multi-programming mode, Number of programmers driven from one PC	Yes, Unlimited	Yes, Unlimited	Yes, Unlimited	Yes, Unlimited
PC interface	USB, 2.0	USB, 2.0	USB, 2.0	USB, 2.0
Programming socket or cable	4 by 48 pin, DIL	48 pin, DIL	40 pin, DIL	Programming cable, 14 pin max
Adapters availability	Phyton brand and third party adapters	Phyton brand and third party adapters	Phyton brand and third party adapters	Phyton brand cables
Software update	Lifetime free of charge	Lifetime free of charge	Lifetime free of charge	Lifetime free of charge
Technical characteristics				
Built-in microcontroller, Fclk	Yes, 32-bit, 60 MHz	Yes, 32-bit, 60 MHz	Yes, 32-bit, 60 MHz	Yes, 8-bit, 10 MHz
Built-in FPGA, Fclk	Yes, up to 100 MHz	Yes, up to 100 MHz	Yes, up to 100 MHz	Yes, up to 10 MHz
Logical pin drivers	Universal, 1.8V to 5.5V	Universal, 1.8V to 5.5V	Universal, 1.8V to 5.5V	Universal, 1.8V to 5.5V
Analog drivers	Universal, 10-bit DAC	Universal, 10-bit DAC	Not universal	Not universal
Adjustability of the write impulses edges' slopes	Yes	Yes	Yes	Yes
	Unlimited	Unlimited	Limited by implementation of the analog drivers	Limited by implementation of the analog drivers
In-system programming capability	Yes, with additional cables	Yes, with additional cables	Yes, with additional cables	Yes
Chip insertion auto detect capability	Yes	Yes	Yes	
Correct chip insertion testing	Yes	Yes	Yes	Yes

Checking bad contact in the programming socket	Yes	Yes	Yes	No
Project management by the software shell	Yes	Yes	Yes	No
Serialization of the programmed devices	Yes	Yes	Yes	No
Writing signatures into the programmed devices	Yes	Yes	Yes	No
Logging programming sessions to files	Yes	Yes	Yes	No
Host computer and operation system	IBM PC, Windows 2000/XP/Vista	IBM PC, Windows 98/ME/ 2000/XP/Vista	IBM PC, Windows 98/ME/ 2000/XP/Vista	IBM PC, Windows 98/ME/ 2000/XP/Vista
Compare the programming + verification time for the selected devices (min: sec)				
M25P20				00:07
SST39VF016Q	00:45	00:45	00:45	02:50
MX28F640C3BB	00:56	00:56	00:56	02:27
MX29LV017A	00:23	00:23	00:23	02:56
MX29LV160CT	00:16	00:16	00:16	01:17
SST49LF008A	00:19	00:19	00:19	01:43
PIC18LF8722	00:11	00:11	00:11	00:19
AT89S51	00:01	00:01	00:01	00:01

2.2 ChipProg-48

The **ChipProg-48** universal programmer can be effectively used for both engineering and low-volume manufacturing. It supports in-socket and in-system programming of thousand of devices and has no valuable limitations in supporting future devices. The unlimited future device support differs **ChipProg-48** from the simplified and less expensive **ChipProg-40** model.

The programmer has a 48-pin DIP ZIF socket that enables inserting any wide or narrow DIP-packed devices with up to 48 leads without the necessity to use any additional adapters. Programming of other devices requires the use of additional adapters available from Phyton and a few selected vendors. The programmer has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”) and the “Start” button for fast launch of the pre-programmed command chains. The palm-size programmer has a wall-plugged power adapter that is not shown on the picture above.

Standard package contents:

- One programmer unit
- One power adapter 12V/1A+
- One USB link cable
- One CD with the ChipProgUSB software

Optionally the package may include one or more programming adapters (if ordered with the programmer) and a “QuickStart” printed manual. See also for more details:

Major features
Hardware characteristics
Software features

2.2.1 Major features

1. Equipped with a 48 pin ZIF socket that allows insertion of the DIP-packed devices with the package width from 300 to 600 mil (7.62 to 15.24 mm) and the number of leads up to 48 without additional adapters.
2. Links to a PC USB 2.0 compatible port, e.g. slower USB connection is also supported.
3. Provides fast programming; for example, completely writes a 64M bit NOR FLASH in less than 50 sec.

4. Can program target devices in the programmer ZIF socket as well as the devices installed in the equipment (ISP mode).
5. ChipProg-48 tools can be driven from multiple USB ports of one computer (or via a USB hub) to provide concurrent programming of multiple devices of the same type.
6. Has a button for fast manual launch of any single operation or a bunch of operations.
7. Has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”).

2.2.2 Hardware characteristics

1. The programmer has a 48-pin ZIF socket with a lever that enables the insertion and clamping of any DIP-packed devices with the package width from 300 to 600 mil (7.62 to 15.24 mm) and with the number of leads up to 48.
2. Adapters for programming devices in the SDIP, PLCC, SOIC, SOP, PSOP, TSOP, TSOPII, TSSOP, QFP, TQFP, VQFP, QFN, SON, BGA, CSP and other packages are available from Phyton and many third parties.
3. The programmer is built on the base of a very fast and productive 32-bit embedded microcontroller and FPGA. These resources allow adding new targets to the device list by a simple software update.
4. Most timing-critical parts of the programming algorithms are implemented on the FPGA devices and do not involve any operations on the embedded microcontroller that would slow down the programming speed.
5. Implementation in the FPGA devices logical drivers enables outputting logical signals of any level (low, high, Pullup, Pulldown and external clock generator) to any pin of the programming ZIF socket.
6. The tool hardware features 10-bit digital-to-analog converters for accurate settings of the analog signals.
7. The tool hardware enables accurate programming of the rising and falling edges of the generated analog signals.
8. The tool hardware automatically adjusts the generated analog signals.
9. The generated analog signals for both the target supplying and programming can be outputted to any pins of the device being programmed.
10. The tool hardware can connect any pin of the device being programmed to the “Ground” level.
11. The tool hardware checks if every pin of the target device is reliably fixed by a ZIF socket’s contacts (“bad contact” checking).
12. The tool hardware protects itself and the target device against incorrect insertions and other issues that cause a sharp increase in the currents though the target device circuits. This “over current” protection is very fast and reliable.
13. The target device pins are protected against the electrostatic discharge.
14. The tool’s hardware has a programmable clock generator.
15.

2.2.3 Software features

1. Works under control of Windows 95/98/ME/2000/XP/Vista.
2. Friendly, intuitive graphic user interface.
3. Includes a set of basic commands: Blank Check, Erase, Read, Write, Verify, Lock, Set Configuration Bits, Data Memory Support, etc., executed by a single mouse click or via menu.
4. Enables programming a batch of the commands above executed one after one either by a manual start or by a mouse click or automatically upon the device insertion into the programming socket.
5. Allows serialization of the programming devices with auto incrementing the device numbers and storing a serialization log.
6. The program can calculate checksums of the selected data array and then write the checksum into a specified memory location of the target device. Several methods of the checksum calculation can be

used.

7. The program allows writing a unique signature into a specified memory location of the target device for the device identification.
8. Project support speeds up and simplifies switching between different programming tasks.
9. The software allows pre-programming a particular operation (or a chain of operations), which is supposed to be automatically triggered when the programmer hardware detects insertion of the target device into the programming socket.
10. An unlimited number of memory buffers can be opened in the main ChipProgUSB window.
11. The software supports a multiple programming mode for concurrent programming of the same type of target devices on the same type of programmers connected to one cluster. The cluster size has no influence on the programming speed.
12. The software includes a full-scale binary editor allowing manual modification of the data in buffers as well as such helpful functions as Search and Replace, Fill, Compare, Copy, Invert, Calculate Checksum, and OR, AND, XOR logical operations on the blocks of data.
13. Loading and saving files in several standard and proprietary formats: Binary, Standard Extended Intel HEX, Motorola S-record, POF, JEDEC, PRG, Holtek OTP, ASCII HEC, ASCII OCTAL, Angstrem SAV. Special non-standard formats can be added on request.
14. The software is featured by a script language and a mechanism of handling the script scenarios for automation of the routine operations and chip replications.

2.3 ChipProg-40

The **ChipProg-40** universal programmer can be effectively used for both engineering and low-volume manufacturing. It supports in-socket and in-system programming of thousand of devices. The programmer hardware has some limitations for supporting certain devices. It does not support any PLDs. This is a difference between the cheaper **ChipProg-40** and the enhanced **-48**

The programmer has a 40-pin DIP ZIF socket that enables inserting any wide or narrow DIP-packed devices with up to 40 leads without the necessity to use any additional adapters. Programming of other devices requires use of additional adapters available from Phyton and a few selected vendors. The programmer has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”) and the “Start” button for fast launch of the pre-programmed command chains. The palm-size programmer has a wall-plugged power adapter that is not shown on the picture above.

Standard package contents:

- One programmer unit
- One power adapter 12V/1A+
- One USB link cable
- One CD with the ChipProgUSB software

Optionally the package may include one or more programming adapters (if ordered with the programmer) and a “QuickStart” printed manual. See also for more details:

Major features

Software features

2.3.1 Major features

1. Equipped with a 40 pin ZIF socket that allows insertion of any DIP-packed devices with the package width from 300 to 600 mil (7.62 to 15.24 mm) and the number of leads up to 40 without additional adapters.
2. Links to a PC USB 2.0 compatible port, e.g. slower USB connection is also supported.
3. Provides fast programming; for example, completely writes a 64M bit NOR FLASH in less than 50 sec.
4.
5. An unlimited number of ChipProg-40 tools can be driven from multiple USB ports of one computer (or via a USB hub) to provide concurrent programming of multiple devices of the same type.
6. Has a button for fast manual launch of any single operation or a batch of operations.
7. Has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”).

2.3.2 Hardware characteristics

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa

eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

2.3.3 Software features

1. Works under control of Windows 95/98/ME/2000/XP/Vista.
2. Friendly, intuitive graphic user interface.
3. Includes a set of basic commands: Blank Check, Erase, Read, Write, Verify, Lock, Set Configuration Bits, Data Memory Support, etc., executed by a single mouse click or via menu.
4. Enables programming a batch of the commands above executed one after one by a manual start, by a mouse click or automatically upon the device insertion into the programming socket.
5. Allows serialization of the programming devices with auto incrementing the device numbers and storing a serialization log.
6. The program can calculate checksums of the selected data array and then write the checksum into a specified memory location of the target device. Several methods of the checksum calculation can be used.
7. The program allows writing a unique signature into a specified memory location of the target device for the device identification.
8. Project support speeds up and simplifies switching between different programming tasks.
9.
10. ChipProgUSB window.
11. The software supports a multiple programming mode for concurrent programming of of the programmers connected to one cluster. The cluster size has no influence on the programming speed.
12. The software includes a full-scale binary editor allowing manual modification of the data in buffers as well as such helpful functions as Search and Replace, Fill, Compare, Copy, Invert, Calculate Checksum, and OR, AND, XOR logical operations on the blocks of data.
13.
14. The software is featured by a script language and a mechanism of handling the script scenarios for automation of the routine operations and chip replications.

2.4 ChipProg-G4

The **ChipProg-G4** is a 4-site gang programmer based on four **ChipProg-48** tools enclosed in onecase and driven from the ChipProgUSB software. It is intended for middle- and low-volume manufacturing. It supports in-socket and in-system programming of thousand of devices and has no valuable limitations for supporting future devices.

Standard package contents:

- One programmer unit
- One power cable
- One USB link cable
- One CD with the ChipProgUSB software

Optionally the package may include one or more programming adapters (if ordered with the programmer) and a “QuickStart” printed manual. See also for more details:

Major features
Hardware characteristics
Software features

2.4.1 Major features

1. Based on four ChipProg-48 tools enclosed in one case and connected to a PC via an embedded USB hub.
2. Allows independent and concurrent programming of up to four devices of the same type.
3. 48 pin ZIF sockets allow insertion of any DIP-packed devices with the package width from 300 to 600 mil (7.62 to 15.24 mm) and the number of leads up to 48 without additional adapters.
4. Links to a PC USB 2.0 compatible port via one link cable.
5. Provides fast programming; for example, completely writes a 64M bit NOR FLASH in less than 50 sec.
6. Can program target devices in its socket as well as devices installed in the equipment (ISP mode).
7. Each programming site has a 'Start' button for fast manual launch of any single operation or a batch of operations.
8. Each programming site has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”).

2.4.2 Hardware characteristics

1. Enclosed in a durable steel case to be used in an industrial environment.
2. The tool gets power from a standard outlet 110-240V, 50-60 Hz.

3. Each programming site based on a single ChipProg-48 programmer has aof any DIP-packed devices with the package width from 300 to 600 mil (7.62 to 15.24 mm) and with the number of leads up to 48.
4. Adapters for programming devices in the SDIP, PLCC, SOIC, SOP, PSOP, TSOP, TSOPII, TSSOP, QFP, TQFP, VQFP, QFN, SON, BGA, CSP and other packages are available from Phyton and many third parties.
5. Single ChipProg-48 programmers inside of the tool enclosure are connected to an embedded USB 2.0 hub
6. Each programming site is built on the base of a very fast and powerful 32-bit embedded microcontroller and FPGA. These resources allow adding new targets to the device list by a simple software update.
7. Most timing-critical parts of the programming algorithms are implemented on the FPGA devices and do not involve any operations on the embedded microcontroller that would slow down the programming speed.
8. Implementation in the FPGA devices logical drivers enable outputting logical signals of any level (low, high, Pullup, Pulldown and external clock generator) to any pin of the programming ZIF socket.
9. The tool hardware features 10-bit digital-to-analog converters for accurate settings of the analog signals.
10. The tool hardware enables accurate programming of the rising and falling edges of the generated analog signals.
11. The tool hardware automatically adjusts the generated analog signals.
12. The generated analog signals for both the target supplying and programming can be outputted to any pins of the device being programmed.
13. The tool hardware can connect any pin of the device being programmed to the “Ground” level.
14. The tool hardware checks if every pin of the target device is reliably fixed by a ZIF socket’s contacts (“bad contact” checking).
15. The tool hardware protects itself and the target device against incorrect insertions and other issues that cause a sharp increase in the currents though the target device circuits. This “over current” protection is very fast and reliable.
16. The target device pins are protected against the electrostatic discharge.
17. The tool’s hardware has a programmable clock generator.
18. The self-testing procedure automatically executes at any time the programmer is powered on.

2.4.3 Software features

1. Works under control of Windows 2000/XP/Vista.
2. Friendly, intuitive graphic user interface allows monitoring the programming sites statuses and can zoom in on operations on each of four programming sites.
3. Includes a set of basic commands: Blank Check, Erase, Read, Write, Verify, Lock, Set Configuration Bits, Data Memory Support, etc., executed by a single mouse click or via menu.
4. Enables programming a batch of the commands above and executed one after one by a manual start, by a mouse click or automatically upon the device insertion into the programming socket.
5. Allows serialization of the programming devices with auto incrementing the device numbers and storing a serialization log.
6.
7. The program allows writing a unique signature into a specified memory location of the target device for the device identification.
8. Project support speeds up and simplifies switching between different programming tasks.
9. The software allows pre-programming a particular operation (or a chain of operations), which is supposed to be automatically triggered when the programmer hardware detects insertion of the target device into the programming socket.
10. Unlimited number of memory buffers can be opened in the main ChipProgUSB window for each of four programming sites.

11. The software includes a full-scale binary editor allowing manual modification of the data in buffers as well as such helpful functions as Search and Replace, Fill, Compare, Copy, Invert, Calculate Checksum, and OR, AND, XOR logical operations on the blocks of data.
12. Loading and saving files in several standard and proprietary formats: Binary, Standard Extended Intel HEX, Motorola S-record, POF, JEDEC, PRG, Holtek OTP, ASCII HEC, ASCII OCTAL, Angstrem SAV. Special non-standard formats can be added on request.
13. The software is featured by a script language and a mechanism of handling the script scenarios for automation of the routine operations and chip replications.

2.5 ChipProg-ISP

The **ChipProg-ISP** is a low-cost universal programmer specifically created for programming devices without removing them from the equipment where they are installed. This type of programming is known as “in-system” or “in-circuit”. The **ChipProg-ISP** supports serial EPROM and EEPROM flash memory devices and embedded microcontrollers with the code and data memory programmable via different types of serial ports: UART, JTAG, SPI and other types, including proprietary interfaces.

The programmer has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”) and the “Start” button for fast launch of the pre-programmed command chains. The tool shown on the picture is very small and requires no power adapter for the operations - it gets power from the USB computer port.
.

Connecting ChipProg-ISP to the target

The programmer has a 14-pin output connector BH-14R. A variety of Phyton adapting cables allow connecting to the target. A simple pin-to-pin ribbon cable is supplied with the programmer by default, and other cables (adapters) can be ordered on demand. The BH-14R connector output information signals for the chip programming and some service signals that enable using the programmer in the automated programming and testing equipment. See the BH-14R pinout:

ChipProg-ISP BH-14R connector	Logical signal
1	Target specific*

2	Target specific*
3	Target specific*
4	Target specific*
5	Target specific*
6	Target specific*
7	Target specific*
8	Target specific*
9	GND
10	Target specific*
11	/Start
12	
13	/Good
14	/Busy

Signals on the pins #1 to #9 and on the pin #10 are used for transmitting and receiving information and synchro impulses to and from the target device. These signals are target specific and depend on the type of target device or a family in general (AVR, PIC, etc.) - see here. They also are shown in the adapters wiring diagrams; see the file included in the ChipProgUSB set.

The pin #9 must be connected to the target's ground.

The signals on the output pins #12, #13 and #14 represent the programmer statuses - logical '0' means an active status, logical '1' - passive. E.g.:

/Error – the operation has failed;
/Good – the operation completed successfully;
/Busy – the programmer is in a process of executing some operation.

An active signal on the input pin #11 (log.'0') starts the preset operation, the device programming by default. Activation of this signal, e.g. a falling edge, is equivalent to pushing the "Start" button on the programmer.

Read also In-System Programming for different devices.

Standard package contents:

- One programmer unit
- One universal ribbon cable wired pin-to-pin
- One USB link cable
- One CD with the ChipProgUSB software

Optionally the package may include one or more programming cable-adapters (if ordered with the programmer) and a “QuickStart” printed manual. See also for more details:

Major features

2.5.1 Major features

1. Has a 14 pin socket for connecting to the target equipment by means of several cable-adapters.
2. Protects itself and the target equipment against incorrect wiring.
3. Links to a PC USB 2.0 compatible port, e.g. slower USB connection is also supported.
4. An unlimited number of ChipProg-ISP tools can be driven from multiple USB ports of one computer (or via a USB hub) to provide concurrent programming of multiple devices of the same type.
5. Has a button for fast manual launch of any single operation or a batch of operations.
6. Has three LEDs for displaying the programming status (“Good”, “Busy”, “Error”).

2.5.2 Hardware characteristics

1. Has a standard 14 pin connector.
2. Adapters for programming devices with in-system programming capability.
3. The programmer is built on the base of a very fast and productive 32-bit embedded microcontroller and FPGA devices. These resources allow adding new targets to the device list by a simple software update.
4. Most timing-critical parts of the programming algorithms are implemented on the FPGA devices and do not involve any operations on the embedded microcontroller that would slow down the programming speed.
5. Implementation in the FPGA devices logical drivers enable outputting logical signals of any level (low, high, Pullup, Pulldown and external clock generator) to any pin of the programming connector.
6. The tool hardware features 10-bit digital-to-analog converters for accurate settings of the analog signals.
7. The tool hardware enables accurate programming of the rising and falling edges of the generated analog signals.
8. The tool hardware automatically adjusts the generated analog signals.
9. The generated analog signals for both the target supplying and programming can be outputted to any pins of the device being programmed.
10. The tool hardware protects itself and the target device against incorrect connection.
11. The target device pins are protected against the electrostatic discharge.
12. Can be started from the external signal.
13. Three status signals “Good”, “Busy”, “Error” are outputted to the programmer connector.
14. The self-testing procedure automatically executes at any time the programmer is powered on.

2.5.3 Software features

1. Works under control of Windows 95/98/ME/2000/XP/Vista.
2. Friendly, intuitive graphic user interface.
3. Includes a set of basic commands: Blank Check, Erase, Read, Write, Verify, Lock, Set Configuration Bits, Data Memory Support, etc., executed by a single mouse click or via menu.
4. Enables programming a batch of the commands above executed one after one by a manual start, by a mouse click or automatically upon the device insertion into the programming socket.
5. Allows serialization of the programming devices with auto incrementing the device numbers and storing a serialization log.
6. The program can calculate checksums of the selected data array and then write the checksum into a specified memory location of the target device. Several methods of the checksum calculation can be used.

7. The program allows writing a unique signature into a specified memory location of the target device for the device identification.
8. Project support speeds up and simplifies switching between different programming tasks.
9. The software allows pre-programming a particular operation (or a chain of operations), which is supposed to be automatically triggered when the programmer hardware detects insertion of the target device into the programming socket.
10. Unlimited number of memory buffers can be opened in the main ChipProgUSB window.
11. The software supports a multiple programming mode for concurrent programming of the same type of target devices on the same type of the programmers connected to one cluster. The cluster size has no influence on the programming speed.
12. The software includes a full-scale binary editor allowing manual modification of the data in buffers as well as such helpful functions as Search and Replace, Fill, Compare, Copy, Invert, Calculate Checksum, and OR, AND, XOR logical operations on the blocks of data.
13. Loading and saving files in several standard and proprietary formats: Binary, Standard Extended Intel HEX, Motorola S-record, POF, JEDEC, PRG, Holtek OTP, ASCII HEC, ASCII OCTAL, Angstrem SAV. Special non-standard formats can be added on request.
14. The software is featured by a script language and a mechanism of handling the script scenarios for automation of the routine operations and chip replications.

3 Quick Start

This chapter includes the topics that describe:

How to install the ChipProgUSB **software**

How to install the ChipProg **USB drivers**

How to install the ChipProg**hardware** and to start up the ChipProg programmers of different type.

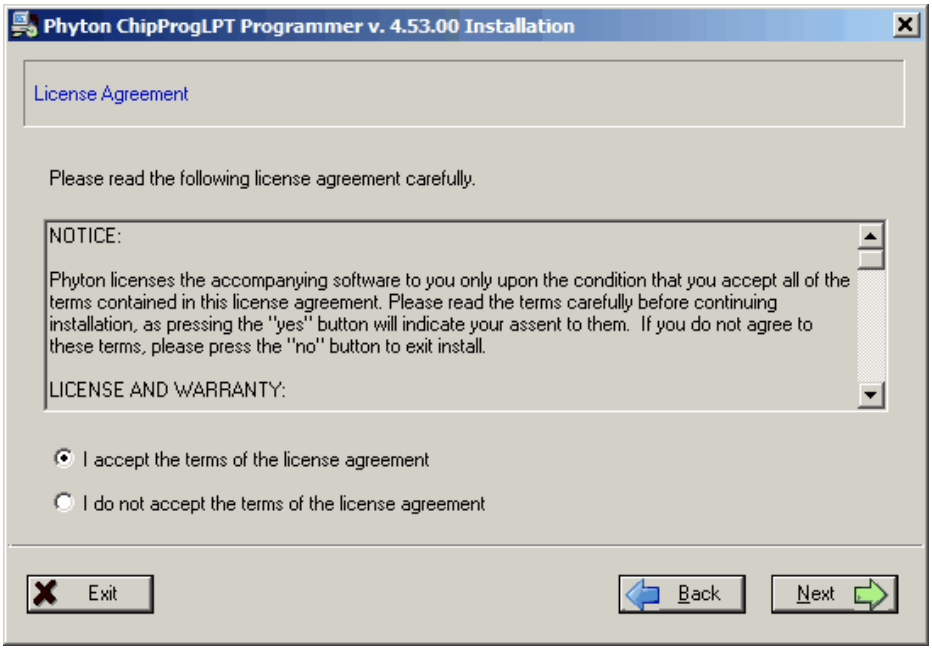
It is **highly recommended**Graphical User Interface and before starting to use the tool.

It is assumed that you are an experienced user of MS Windows and basic Windows operations.

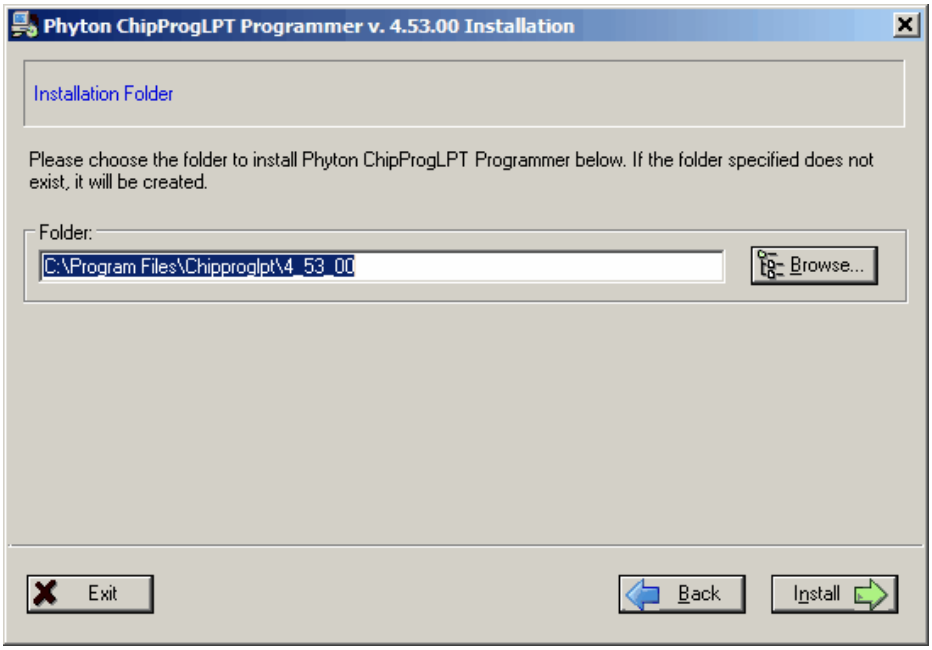
3.1 Installing the ChipProgUSB Software

Insert the distributive ChipProgUSB disc into a CD drive of your PC, click the install button and then follow the series of prompts that will lead you through the installation process.

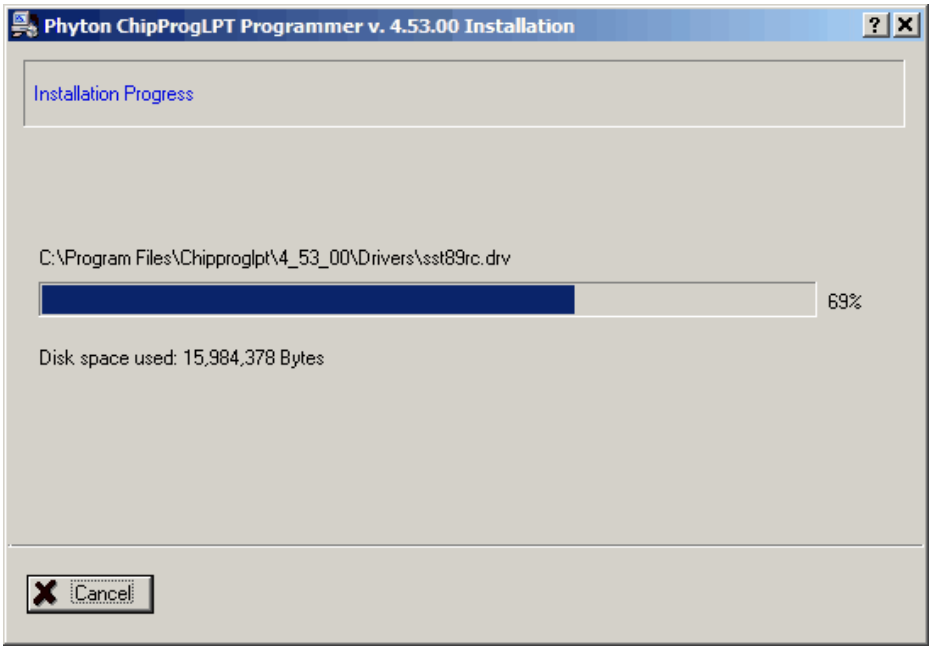
Accept the terms of license agreement



Choose the folder to install

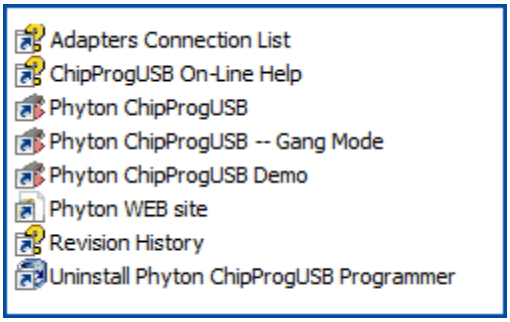


Wait for installation...



Phyton folder

At the end the installer will create a folder with ChipProgUSB tools' and documents' shortcuts:



Phyton Programming Adapter List - opens the adapters.chm file that list all the Phyton programming adapters with their short descriptions and wiring diagrams.

ChipProgUSB On-Line Help - opens the programmer on-line Help document.

Phyton ChipProgUSB - invokes the ChipProgUSB executable file and starts operations for the ChipProgChipProg-40 and ChipProg-ISP programmers working in a single programming mode.

Phyton ChipProgUSB -- Gang Mode - invokes the ChipProgUSB executable file and starts operations for the ChipProg -G4 gang programmer or the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers working in a multiprogramming mode.

Phyton ChipProgUSB Demo - invokes a demo version of the ChipProgUSB software that allows evaluating the product without its hardware.

Phyton WEB site - opens an Internet browser with the www.phyton.com website.

Revision History - opens the ChipProgUSB versions history file.

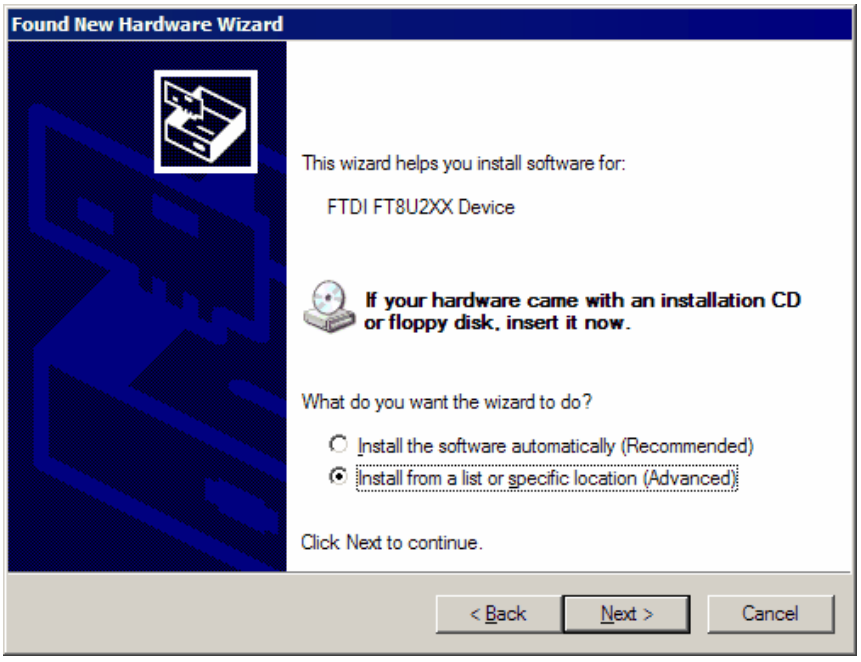
Uninstall Phyton ChipProgUSB Programmer - starts a process of removing the ChipProgUSB program from your computer.

3.2 Installing the USB Drivers

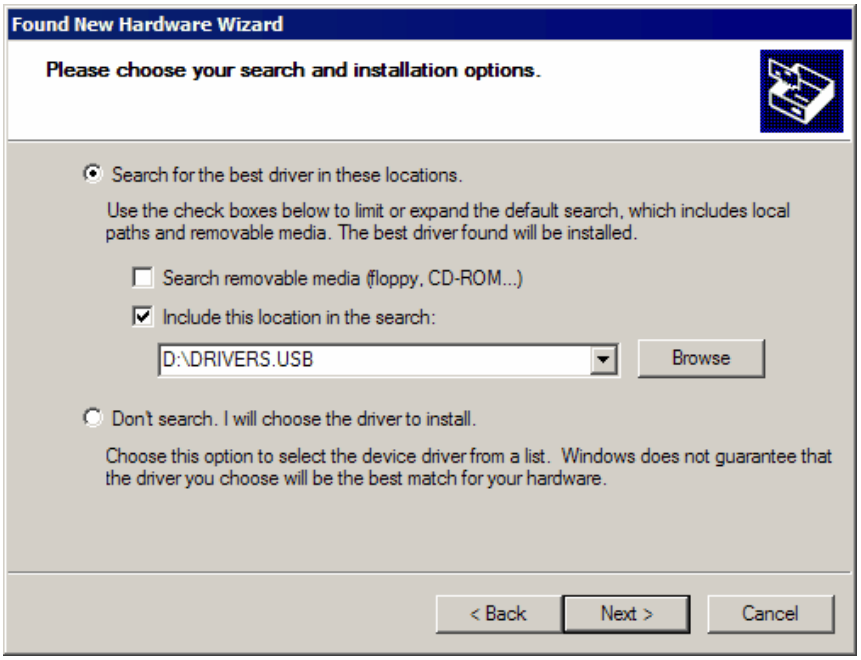
In a process of the ChipProgUSB software installation from a distributive disc the program installs the drivers for the USB devices used in all types of the ChipProg programmers working under control of the **Windows 2000/XP/Vista/7** operating systems. Only the -48, -40 and ChipProg-ISP programmers working under control of the **Windows 98/ME** operating systems require the USB drivers to be installed after the software installation is completed. The guidelines below are for the **Windows 98/ME** operating systems only.

To invoke the USB drivers installation procedure connect a ChipProgUSB to the USB port of your computer via the included USB cable. You should see the **Found New Hardware Wizard** dialog:

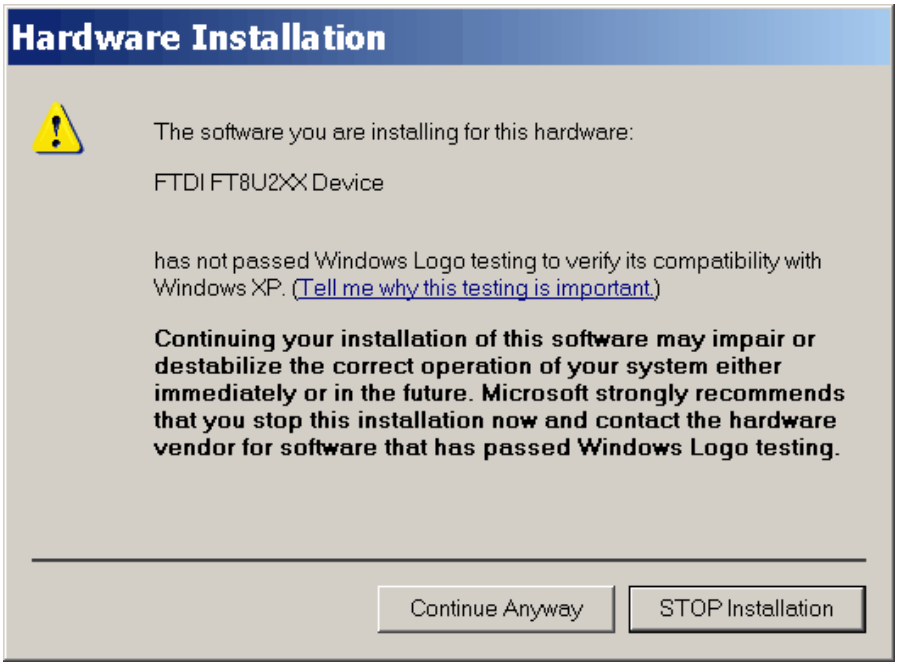
Select the **'No, not this time'** option and click the **Next** button. The wizard below will appear:



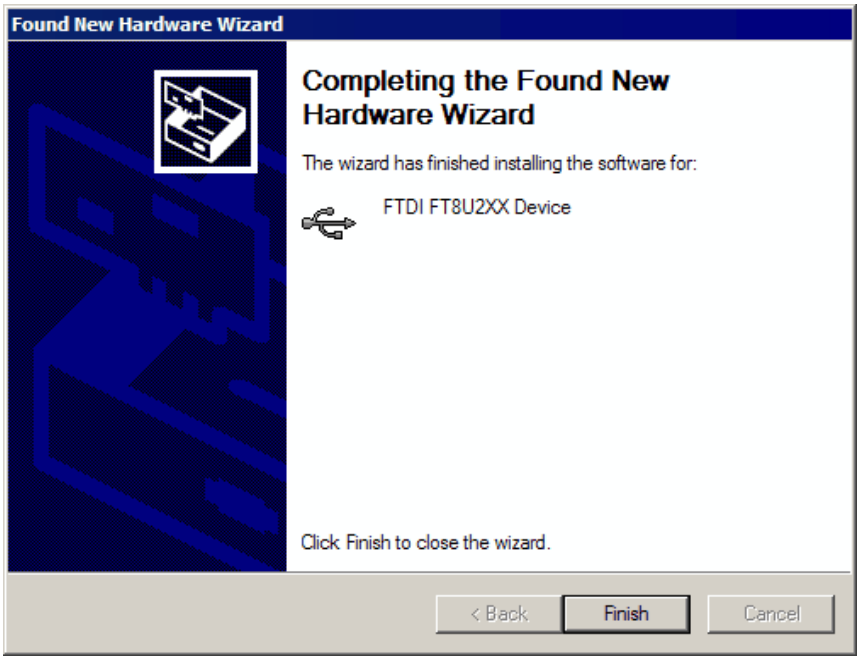
Select the **'Install from a list or specific location'** option and click the **Next**



Browse to the **DRIVERS.USB** folder on the Phyton CD and click the **Next** button (certainly the drive letter can be other than D:). This will start the drivers installation.



Click the **Continue Anyway** button to complete the installation; you will soon get the last prompt:



Click the button. Now you can use the ChipProg

3.3 Hardware installation

It is a mandatory for you to use the original power adapter 12V/1A received with the ChipProg-40 or ChipProg -48 programmer and an original power cord for the ChipProg-G4 gang programmer. Any substitutions should be agreed to with Phyton. It is also highly recommended to use the USB link cables received with the programmers.

The hardware installations for different programmer models vary. Select the topic to see:

- The ChipProg-48 hardware installation
- The ChipProg-40 hardware installation
- The ChipProg-G4 hardware installation
- The ChipProg-ISP hardware installation

3.3.1 ChipProg-48

For the programmer to be used in a single-programming mode:

Powering the programmer	Plug the power adapter to the ~110/240V outlet. Connect a plug of the power adapters to the coaxial connector on the rear panel of the programmers and make sure that the "Good" green LEDs on each of them are on.
Connecting to a PC	Connect the USB port of your PC to the USB connector on the rear panel of the programmer by means of the USB cable. It is highly recommended to connect the programmer to a USB slot on the computer main unit and do not connect it through a USB hub, especially through a passive hub.
Starting up	Start the **Phyton ChipProgUSB** program; if the programmer passes the startup test successfully the first dialog prompts you to choose one of the programmers to work with: ChipProg-48, ChipProg-40 or ChipProg-ISP. Select the ChipProg-48 and continue. The ChipProgUSB main window will open and you will be able to work with the tool.

For the programmers to be used in a multi-programming mode, e.g. connected to one computer:

Powering the programmers	Plug the power adapters of each programmer to be connected in one programming cluster to the 110/240V outlets. Connect plugs of the power adapters to the coaxial connectors on the rear panels of all the programmers and make sure that the "Good" green LEDs on each of them are on.
Connecting the programmers to a cluster	Connect the USB ports of your PCs to the USB connectors on the rear panels of the programmers by means of the USB cables. It's highly recommended to connect the programmers to USB slots on the computer main unit and do not connect them through a USB hub, especially through a passive hub.
Starting up	Start the **Phyton ChipProgUSB - Gang Mode** program; if all the programmers pass the startup test successfully the first dialog prompts you to assign the number from one to N to each programmer included in the cluster. To assign the number push a **Start** button on a top panel of each programmer one by one. Then the ChipProgUSB main window will open and you will be able to work with the tool.

Read about the **Multi-Programming mode**.

3.3.2 ChipProg-40

For the programmer to be used in a single programming mode, e.g. alone:

Powering the programmer	Plug the power adapter to the ~110/240V outlet. Connect a plug of the power adapter to the coaxial connector on the rear panel of the programmer and make sure that the "Good" green LED on the programmer is on.
Connecting to a PC	Connect a USB port of your PC to a USB connector on the rear panel of the programmer by means of the USB cable. It's highly recommended to connect the programmer to a USB slot on the computer main unit and not connect it through a USB hub, especially through a passive hub.
Starting up	Start the **Phyton ChipProgUSB** program; if the programmer passes the startup test successfully the first dialog prompts you to choose one of the programmers to work with: ChipProg-48, ChipProg-40 or ChipProg-ISP. Select the ChipProg-40 and continue. The ChipProgUSB main window will open and you will be able to work with the tool.

For the programmers to be used in a multi-programming mode, e.g. connected to one computer:

Powering the programmers	Plug the power adapters of each programmer to be connected in one programming cluster to the 110/240V outlets. Connect plugs of the power adapter to the coaxial connectors on the rear panels of all the programmers and make sure that the "Good"LEDs on each of them are on.
Connecting the programmers to a cluster	Connect USB ports of your PCs to USB connectors on the rear panels of the programmers by means of the USB cables. It's highly recommended to connect the programmers to USB slots on the computer main unit and not connect them through a USB hub, especially through a passive hub.
Starting up	Start the **Phyton ChipProgUSB - Gang Mode** program; if all the programmers pass the startup test successfully the first dialog prompts you to assign the number from one to N to each programmer included in the cluster. To assign the number push a **Start** button on a top panel of each programmer one by one. Then the main window will open and you will be able to work with the tool.

Read about the **Multi-Programming mode**.

3.3.3 ChipProg-G4

Powering the programmer	Plug the power cord to a power connector on the rear panel of the programmer, then plug an opposite site to the ~110/240V outlet. Make sure that all four "Good" green LEDs on the programmer are on.

Connecting to a PC

Connect a USB port of your PC to a USB connector on the rear panel of the programmer by means of the USB cable. It's highly recommended to connect the programmer to a USB slot on the computer main unit and not connect it through a USB hub, especially through a passive hub. Use of the passive USB hubs for connecting the ChipProg-G4 programmer is not allowed.

Starting up

Important! When you start the programmer first time wait for about 20 seconds to allow the USB driver to be setup. Then, every time when you start the programmer, wait for 5...10 sec before launching the ChipProgUSB software.

Start the **Phyton ChipProgUSB - Gang ModeStart** button on a top panel of the programmer one by one. Then the ChipProgUSB main window will open and you will be able to work with the tool.

3.3.4 ChipProg-ISP

For the programmer to be used in a single programming mode, e.g. alone:

Connecting to a PC

Connect a USB port of your PC to a USB connector on the rear panel of the programmer by means of the USB cable. Make sure that the "Good"

Start the **Phyton ChipProgUSB** program; if the programmer passes the startup test successfully the first dialog prompts you to choose one of the programmers to work with: ChipProg-48, ChipProg-40 or ChipProg-ISP. Select the ChipProg-ISP and continue. The ChipProgUSB main window will open and you will be able to work with the tool.

Connecting the programmers to a cluster

Connect USB ports of your PCs to USB connectors on the rear panels of the programmers by means of the USB cables. Make sure that the "Good" green LEDs on all the programmers are on. It's highly recommended to connect the programmers to USB slots on the computer main unit and not connect them through a USB hub. The ChipProg programmers get power from the computer's USB port; that is why it's important not to overload the ports. Use of the passive USB hubs for clustering the ISP programmers is not allowed.

Starting up

Start the **Phyton ChipProgUSB - Gang Mode** program; if all the programmers pass the startup test successfully the first dialog prompts you to assign the number from one to N to each programmer included in the cluster. To assign the number push a **Start** button on a top panel of each programmer one by one. Then the ChipProgUSB main window will open and you will be able to work with the tool.

Read about the **Multi-Programming mode**.

3.4 Getting Assistance

3.4.1 On-line Help

The ChipProgUSB software has a pretty comprehensive context-sensitive on-line **Help**. To access it press the **F1** key or use the **Help menu**. Almost every **F1**.

In most cases you can find the necessary topic by searching for a keyword. For example, if you type "Verify" in the first box of the **Find** tab, the third box will list the topics related to the programming verification. Choose an appropriate topic from this list and press **Display**

3.4.2 Technical Support

During a product's warranty period Phyton provides technical support free of charge. Though we have been selling the ChipProg programmers for many years the product software may contain minor bugs and some programming algorithms may not be stable on some of the supported devices. We kindly ask you to report bugs when you get an error message or have a problem with programming a particular device or devices. We commit to prompt checking of your information and fixing the detected bugs.

To minimize difficulties operating with ChipProgUSB it is highly recommended to get familiar with the manual before using the programmer. The ChipProgUSB - user interface is quite standard and intuitive, however it includes some specific functions and controls that the user should learn about.

Before contacting Phyton

- Make sure that you use the latest ChipProgUSB version that is always available for free download from the http://www.phyton.com.
- Make sure the detected error can be reproduced in the same working environment and is not a casual glitch.

When contacting us

Please, provide our technical support specialists with the following information:

-
- Name of the ChipProg model and its serial number, if one exists.
- Date of purchase, the Phyton invoice number, if available.
- Software version number taken from the **About** information box.
- Basic parameters of your computer and operating system.
- The device type, mechanical package and the type of the adapter if one is used.
- Descriptions of detected errors, relevant bug reports and error screen shots.

Please send your requests or questions to **support@phyton.com**. This is the easiest way to get professional and prompt help. Also, see Contact Information.

3.4.3 Contact Information

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

4 Graphical User Interface

The ChipProgUSB graphical user interface (GUI) elements include:

- Menus - global and local
- Windows
- Toolbars - global and local
- Setting Dialogs
- Hot Keys
- Context-sensitive help prompts

GUI featured with several useful additions specifically created for the ChipProg operations.

To make your operations with the ChipProgUSB program easier we highly recommend to learn the chapters Menus and Windows in full. You will be able to use the tools much more effectively.

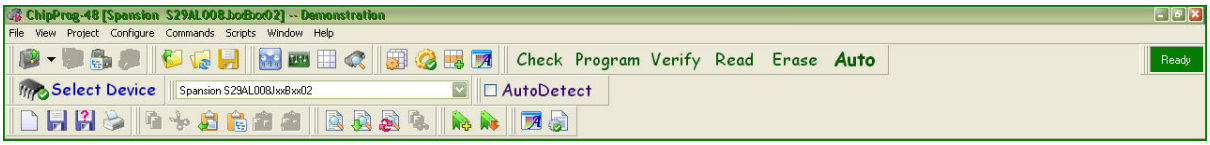
4.1 User Interface Overview

ChipProgUSB features the standard Windows interface with several useful additions:

1. Each window has its own local menu (the shortcut menu). To open this menu, click the right mouse button within the window area or press **Ctrl+Enter** or **Ctrl+F10**. Each command in the menu has a hot key shortcut assigned to the **Ctrl+<letter>** keys. Pressing the hot key combination in the active window executes the corresponding command.
2. Each window has its own local toolbar. The window's toolbar buttons give access to most of the window's local menu commands. The specialized window toolbar buttons operate only within the specialized window. The main window has several toolbars that can be turned on or off (in the **Environment** dialog, the **Toolbar tab**).
3. Each toolbar button has a short prompt: when you place the cursor over a toolbar button for two seconds, a small yellow box appears nearby with a short description of the button's function.
4. To save screen space, you can hide any window's title bar. To do this, use the **Properties** command of the local menu. You can identify the ChipProgUSB windows by their contents and position on the screen (and, if you wish, by color and font). When the title bar is hidden, you can move the window as if the toolbar were the title bar: place the cursor on the free space of the toolbar, press the left mouse button and drag the window to a new position.
5. You can open any number of windows of the same type. For example, you can open several windows.
6. Every input text field of any dialog box has a history list. ChipProgUSB saves them when you close a development session. Then a previously entered string can be picked from the history list.
7. All input text boxes in the dialogs feature automatic name completion.
8. **OK** button. This is convenient when you need to change only one option in the dialog and then close it.

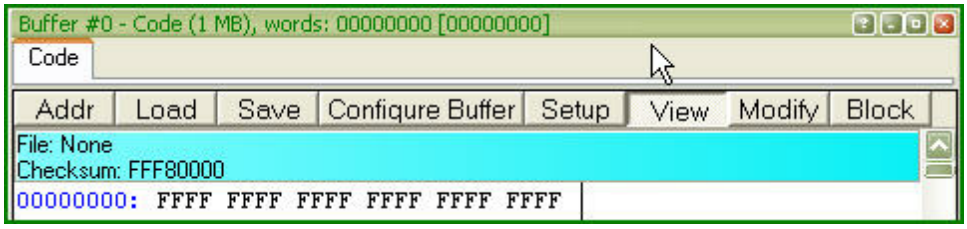
4.2 Toolbars

The ChipProgUSB program opens a few toolbars on top of the main window (see below).



The top line, shown right under the ChipProg main window title, includes the Main menu submenus. A second line under the Main menu line displays icons and buttons of most frequently used commands on files and target devices (Open project, Load file, Save file... Check, Program, Verify, etc.). There is an indicator of the ChipProgUSB status (Ready, Wait, etc.). The third line displays a target device selector. The fourth line, which is not displayed by default, includes an embedded editor options and commands for scripts. The default toolbars can be customized. Read also the topics: The Configure Menu, The Environment dialog, Toolbar.

Besides the toolbars positioned on a top of the main window, each particular window has its own local toolbar with the buttons presenting the most popular commands associated with the window. See for example the Buffer window's toolbar below.



4.3 Menus

ChipProgUSB Main menu bar includes the following pull-down sub-menus:

- **File menu**
- **View menu**
- **Project menu**
- **Configure menu**
- **Commands menu**
- **Scripts menu**
- **Window menu**
- **Help menu**

To access these menus, use the mouse or press **Alt+letter**, where "letter" is the underlined character in the name of the menu item.

4.3.1 The File Menu

The **File** menu's commands control the file operations. For those commands that have a toolbar button, the button is shown in the first column of the table below. If there is a shortcut key for a command, the shortcut key is shown at the right of the command in the menu.

Button	Command	Description
	**Load ...**	Opens the **Load file** dialog that specifies all the parameters of the file to be loaded and the file destination.
	**Reload**	Reloads the most recently loaded file with the most recently specified parameters.
	**Save...**	Saves the file from the currently active window to a disk. Opens the **Save file from buffer** dialog.
	Configuration Files	Gives access to operations with configuration files.
	**Exit**	Closes ChipProgUSB. Alternatively, use the standard ways to close a Windows application (the **Alt+F4** or **Alt+X** keys combination).

4.3.1.1 Configuration Files

On exit ChipProgUSB automatically saves its configuration data in several configuration files with the name **UPROG**. On start, it restores its configuration from the last saved configuration files. In addition, you can save and load any of these files at any time using the **Configuration Files** command of the **File** menu. You can have several sets of configuration files for different purposes.

- The **Desktop** file contains data about the display options and the screen configuration, and the positions, dimensions, colors and fonts of all the opened windows. The extension of this file is **.dsk.** The default file name is **UPROG.dsk.**
- The **Options** file stores the target device type, file options, etc. The extension of this file is . The default file name is **UPROG.opt.**
- The **Session** file, which stores session data and specifies the desktop and options; it can also be saved and loaded by means of the **Save session** or **Load session Configuration Files** command. The extension of this file is **.ses**. The default file name is **UPROG.ses.**
- The file, which contains all the settings entered in the text boxes of all the ChipProgUSB dialogs. This file is hidden from users, but the settings stored earlier are available for prompt pick up from the History lists. The extension of this file is **.hst**. The default file name is **UPROG.hst.**

4.3.2 The View Menu

This menu controls access to the ChipProgUSB windows:

Button	Command	Description
	**Program Manager**	Opens the **Program Manager** dialog.
	**Device and Algorithm Parameters**	Opens the **Device and Algorithm Parameters** dialog.
	**Buffer Dump**	Opens the **Buffer** dialog.
	**Device Information**	Opens the **Device Information** dialog.
	**Console**	Opens the **Console** dialog.

Local window menus

Each window has its own local (shortcut) menu. To open a local window menu, either click the right mouse button within the window or press **Ctrl+Enter** or **Ctrl+F10**.

Most, but not all, of the local menu commands are duplicated by local toolbar buttons that are usually displayed at the top of every window.

4.3.3 The Project Menu

This menu contains commands for working with projects.

Button		Description
	**New**	Opens the **Project Options** dialog.
	**Open**	Opens the **Open Project** for loading an existing project file.
	**Close**	Saves and closes a currently opened project
	**Save**	Saves the currently opened project. Note that when you close a project, create a new project or just exit, the current project will be saved automatically.
	Copy As	Opens the **Save project** dialog. Duplicating projects is helpful for making project clones and other purposes.
	**Repository**	Opens the **Project Repository** dialog.
	**Options**	Opens the dialog

4.3.3.1 The Project Options Dialog

This dialog is used to define the project options.

Element of dialog	Description
Project File Name	Specifies the project file name. The project name does not include a path. The extension may be omitted.
Project Description (optional)	Here you can enter your custom comments for the project.
Desktop	Two radio buttons which allow you to choose if the project has its own desktop or if there is one desktop for all projects.
Files to Load to Buffers	File or list of files to load into the buffers.
Add file	Opens the **Load File**

	Remove the selected file from field **Files to Load to Buffers.**
Edit file options	Opens the **Load File** dialog.
Script to execute before loading files:	Here you can enter the script name to be executed before loading the files to the project.
Script to execute after loading files:	Here you can enter the script name to be executed after loading the files to the project.

4.3.3.2 The Open Project Dialog

This dialog is used to open the project which was previously created.

Element of dialog	Description
Project File Name	Specifies the project file name. The project name does not include a path. The extension may be omitted.
Project Open History	Lists the previously opened projects. Double-clicking a line in the list opens a corresponding project.
Remove from list	Deletes the selected project from the list.

4.3.3.3 Project Repository

The **Project Repository** tree is a small database that stores records with links to the project files. You can use this database to sort and group the projects as needed for better presentation and easier access. The ChipProgUSB program displays the repository in a tree-like form that is similar to Windows Explorer's. Operations with the repository do not change the project files themselves. The repository works only with records about the projects (links to the project files). A tree branch may show projects and other branches. Any branch may contain different projects with the same names. Different branches may contain links to the same project.

To open the **Project Repository** dialog invoke the **Repository** command of the Project menu. Each tree branch displays the name of a particular project file without a path and the project description shown in square brackets. The remembers the state of the tree branches (expanded / collapsed) and restores it next time you open the dialog.

When you install a new version of the ChipProgUSB software and copy the working environment from the previously installed version, the new version will inherit the existing project repository (file repos. ini).

Element of dialog	Description
Add New Branch	**Add New Branch** dialog to specify the name of a new branch. When **OK**

	is pressed, the new branch is attached to the selected branch.
Add a Project to Branch	Opens the **Open Project** dialog to select a project to be added. When **Open** is pressed, the selected project is added to the selected branch.
Add Current Project to Branch	Adds the currently opened project to the selected branch.
Remove Project/Branch	Deletes the selected project or branch from the repository. When deleting a branch, all branches that "grow" from this branch and all projects located on it will be deleted. When deleting a project from the repository, the ChipProgUSB deletes only the repository record about the project, and **does not delete the project from the disc**.
Edit Branch Name	Opens the **Edit Branch Name** dialog for the selected branch.
Move Up	Moves a selected project or branch up the tree within the same level of hierarchy. The branch moves together with all branches that "grow" from it and all its projects.
Move Down	
Save Repository	Writes the repository to a disc file.
Browse Project Folder	Opens MS Windows Explorer with the opened folder of the selected project.
Open Project	
Close	Closes the dialog. If the repository is changed, ChipProgUSB will ask whether to save it.

4.3.4 The Configure Menu

This menu gives access to all the ChipProgUSB configuration dialogs.

Button	Command	Description
	**Select device ...**	Opens the **Select Device** dialog.
	Device selection history	Lists the previously selected devices.
	**Buffers**	Opens the **Buffers** dialog.
	Serialization, Checksum, Log file	Opens the **Serialization, Checksum, Log File**
	**Preferences**	Opens the **Preferences** dialog.
	**Environment**	Opens the **Environment** dialog with tabs: the **Fonts** tab, the **Colors** tab, the **Key Mappings**, the **Toolbar** tab and the **Misc** tab.

4.3.4.1 The Select Device dialog

The dialog allows specification of the device to work with; it has a few groups of settings.

Element of dialog	Description
Devices to list:	In this field you can check the box or boxes to specify the target device type. All the devices are divided in three functional groups: a) EPROM, EEPROM, FLASH; b) PLD, PAL, EPLD; c) Microcontrollers - check one, two or all three boxes. Two check boxes below specify a method of programming - in the programmer socket or in the target system - some devices can be programmed in either way, some only in one certain way. It is recommended to narrow down the searchable database and speed up the search by specifying the device properties if possible.
Search mask:	The box lists the device manufacturers in alphabetic order. Here you can enter a mask to speed up the device search. The character '*' masks any number of any characters in the device part number. For example, the mask 'PIC18*64' will bring up all the PIC18 devices ending with the '64'.
Devices	The file displays all the devices for a chosen manufacturer that match to the search criteria specified in the **Devices to list, Search mask** and **Packages/Adapters** fields.
Packages/Adapters	This field lists all types of the chosen device's mechanical packages that can are supported by the the ChipProg and appropriate adapters.

4.3.4.2 The Buffers dialog

Element of dialog	Description
Buffer list:	Displays names, sizes and sub-layers of all currently open buffers
Add...	Opens the **Buffer Configuration** dialog to create a new buffer
Delete	Deletes the buffer highlighted in the 'Buffer list' box.
Edit...	Opens the **Buffer Configuration** dialog for editing.
View	Switches control to window displaying the buffer highlighted in the 'Buffer list' box. If this window is hidden under others it will be brought to the foreground.
Memory Allocation	This drop down menu allows limiting the memory size allocated from the computer RAM to each buffer. The free memory currently

	available for the allocation is shown here in this screen area.
Swap Files	If the RAM space is limited the ChipProgUSB can use some space on the PC drives by temporary writing the buffer image to the drive. You can select the drive or allow the program to swap the files automatically.
Use network drives	Checking this box enables you to swap files on the network drives connected to your computer.
Amount of space to leave free on each drive (GB):	Here you can limit the space on the drive which will be never affected by the file swapping.

4.3.4.2.1 The Buffer Configuration dialog

The dialog allows the setup of sub-layers in the buffers and to make their presentation easier to work with.

The dialog includes as many tabs as number of sub-layers exist for a particular device. Every buffer has at least one main layer, so the tab 'Code' is always displayed on the dialog foreground. If a chosen device has other address spaces ('Data', 'User', etc.) the buffer has additional sub-layers available for setting up by clicking the appropriate tabs.

4.3.4.2.1.1 Main Buffer Layer

The tab opens the dialog for configuring the main buffer layer - the 'Code' layer.

Element of dialog	Description
Buffer Name	Here you can type in a name for the buffer or pick it from the history list. By default the first opened buffer gets the name "Buffer #0". Then you can open the "Buffer #1", etc. or give the buffer any name you wish.
Size of sub-layer 'Code'	Here you can assign a size of the 'Code' layer from the drop-down menu - from 128KB to 32MB.
Fill sub-layer 'Code' with data:	The program fills the buffer sub-layers with some default information, usually by the 'FF's or zeros. By checking these boxes you specify when the layer 'Code' should be filled with the default information - before loading the file or right after the device type has been chosen.
Data to fill sub-layer with:	These two toggled radio buttons define if the sub-layer 'Code' will be filled with some default information, specific for the selected device, or by the custom bit pattern.
Shrink buffer size when device is selected	The buffer size usually exceeds the target device 'Code' size. By checking this box you downsize the buffer to match the target device and to free some computer memory.

4.3.4.2.1.2 Buffer Layers

The tab opens the dialog for presetting the buffer sub-layers.

Element of dialog	Description
Fill sub-level 'ID location' with data:	By checking these boxes you specify when the chosen sub-layer should be filled with the default information - before loading the file or right after the device type has been chosen..
Data to fill sub-level with:	These two toggled radio buttons define if the chosen sub-layer will be filled with some default information, specific for the selected device, or by the custom bit pattern..

4.3.4.3 The Serialization, Checksum and Log dialog

The dialog includes the following tabs:

Serial Number,

Checksum,

Signature String,

Log File.

4.3.4.3.1 Device Serialization

The dialog allows set up of the procedure of giving a unique number to any single device belonging to a series of devices being programmed.

Element of dialog	Description
Write S/N to address in sub-level:	This option enables writing a unique device serial number into the sub-layer specified here and at the address in the sub-layer also specified here.
Current serial number:	Specify the current serial number in this box.
S/N size, in byte:	Specify a size of the serial number in bytes, for example: 1, 2, 4, etc.
Byte Order	These two toggled radio buttons define an order of bytes that represent the serial number (if it occupies more than one byte) - either the least significant byte (LSB) follows the most significant byte (MSB) or vise versa.
Display S/N as:	These radio buttons set the serial number display format - decimal or hexadecimal.

Increment serial number by:	By checking this radio button you set incrementing the serial number by the fixed value specified here, for example: 1, 2, 10, etc.
Use script to increment serial number:	By checking this radio button you specify the increment value as a result of executing some script file, which can be put in the box here.

4.3.4.3.2 Checksum

The dialog allows to automatically calculate checksums of the data in buffers. Since there are several more or less standard algorithms for the checksum calculation the dialog enables you to set one standard algorithm or to create some custom, complex algorithms by using a script.

Element of dialog	Description
Write checksum to address: in sub-layer:	By checking this box you begin automatically calculating the checksum in accordance to other settings below and to write it to a specified location in the chosen sub-layer.
Address range to calculate checksum for:	The Start and End addresses define the range of buffer addresses for which the program calculates the checksum.
End:	
Use algorithm to calculate checksum:	One of two toggled radio buttons. If checked, one of the preset algorithms of the checksum calculation can be picked from the drop-down list.
Use script to calculate checksum:	This radio button sets an alternative method of the checksum calculation by means of a custom made script.
Size of summation result	These radio buttons allow the selection of the checksum size: one, two or four bytes.
Operation on summation result	These radio buttons allow the application of some operation to the raw result of the data summation: Negate, Compliment or do not apply any operation.
Size of data being summed	These radio buttons allow to select the source data size: one, two or four bytes
Byte Order	These two toggled radio buttons define an order of bytes that represent the checksum - either the least significant byte (LSB) follows the most significant byte (MSB) or vice versa.

4.3.4.3.3 Signature string

The dialog allows set up of the procedure of giving a signature to the devices being programmed. The signature string may include some generic data like the date when the device has been programmed and some unique data like the device serial number.

Element of dialog	Description
Write Signature String to address: **in sub-layer:**	By checking this box you automatically writing a preset string to a specified location in the chosen buffer sub-layer.
Max. size signature string:	This field reserves a maximum length of the signature string in the number of characters.
Use Signature String template:	One of two toggled radio buttons. If checked, the string pattern preset in the **Template String Specifiers** window will be programmed into the target device.
Use script to create Signature String:	This radio button sets an alternative method of composing the signature string by means of a custom made script.
Template String Specifiers:	The list of the signature string specifiers to be placed into the **Use Signature String template** field. It usually includes the date and time of the device programming, its serial number and checksum.

4.3.4.3.4 Log file

The dialog allows set up of a log or logs of the device programming.

Element of dialog	Description
Enable log file	Checking this box enables the device programming log.
Separate log file for each device	These two toggled radio buttons set if the logs will be separated by a manufacturer or by the target device type or a single log that will be kept for all the devices being programmed.
File Name (Generated Automatically)	Another two toggled radio buttons that set what specifier will be included into the log file name: both the manufacturer and device type (for example: Atmel AT89C51, Microchip PIC18F2525, etc.) or just the device type (for example: AT89C51, PIC18F2525, etc.).
Folder for log file:	This is a field for entering a full path to the folder where the log file will be kept. There is also a button for the path browsing.

Single log file for all device types	By checking this radio button you select keeping one common log for all types of the devices being programmed.
File Name	This is a field for entering a full path to the folder where the common log file will be kept. There is also a button for the path browsing.
Log File Contents	A set of the log file options.
Gang mode: Socket #	If the device programming was conducted in the Gang (multiprogramming) mode and if this box is checked the socket number will be logged.
Date/Time	By checking this box you enable logging the date and time of the device programming.
Events (device type change, file names, etc.)	
Device operation	By checking this box you enable logging of all the events associated with the device manipulations.
Detailed Device operation	By checking this box you enable more detailed logging of all the events associated with the device manipulations.
Operation Result	By checking this box you enable logging the results of the programming operations.
Device #/Good devices/Bad devices	By checking this box you enable logging a full number of the devices programmed, number of successfully programmed devices and number of failed ones.
Serial Number	By checking this box you enable logging the serial number read from the device.
Signature string	By checking this box you enable logging the signature string read from the device.
Checksum	By checking this box you enable logging the checksum value read from the device.
Buffer name	By checking this box you enable logging the buffer name.
Programming address	By checking this box you enable logging the ranges of the device locations which have been programmed.
Programming options	By checking this box you enable logging all the programming options.
Log File Format	A pair of toggled radio buttons: one sets the plain text format of the log file, the second sets the tabulated text to be viewed in the Microsoft Excel format.
Log File Overwrite Mode	ChipProg re-starts.

Warn if size exceeds	If this box is checked then every time when the log size exceeds a specified value the ChipProgUSB issues the warning.
Immediately write log file to disk, no buffering	If this box is checked then the ChipProgUSB does not buffer the log to the computer RAM but writes it straight to the drive

4.3.4.4 The Preferences dialog

Description	
Options	
Reload last file on start-up	By checking this box you enable re-loading to the open buffer(s) the last loaded file every time when you start the ChipProg.
Execute Power-On test on start-up	This box is checked by default. By un-checking it you disable executing the start-up ChipProg self-testing.
Sounds	All programmable sounds can be picked from the preset ChipProgUSBsounds
Device operation error:	Select the sound for error operations.
Device operation complete:	Select the sound for successful completion of the programming operations in a single programming mode (one ChipProg is in use).
Device operation complete (Gang Mode):	Select the sound for successful completion of the programming operations in a gang programming mode (either a few single site programmers are connected to one PC for multi-device programming or when the ChipProg
Programming start (AutoDetect Mode):	Select the sound for indicating the start of the device programming when the ChipProg automatically detects the device insertion into the programming socket.

4.3.4.5 The Environment dialog

The Environment dialog includes the following tabs:

Fonts tab,

Colors tab,

tab,

Toolbar tab**,**

Miscellaneous Settings tab**.**

4.3.4.5.1 Fonts

The **Fonts** tab of the **Environment** dialog opens a sub dialog for setting fonts and some appearance elements in the ChipProgUSB windows. Only mono-spaced (non-proportional) fonts (default is Fixedsys) are used to display information in windows. To improve appearance of the windows, you can set up either another font for all windows, or individual fonts for each particular window.

The **Windows** area lists the types of windows. Select a type to set up its options. The set options are valid for all windows of the selected type, including the already opened windows.

Element of dialog	Description
Window Title Bar	Toggles the title bar for windows of the selected type. If the box is checked it adds a toolbar at the position specified by the **Windows Toolbar Location** option**.** To save screen space uncheck the box. Also, see notes below.
	Sets the toolbar location for the selected window.
Grid	
Additional Line Spacing	
Define Font	Opens the **Font** dialog. The selected font is valid for all windows of the selected type.
Use This Font for All Windows	Applies the font of the chosen window type to all ChipProgUSB windows.

Notes

1. To move a window without the title bar, place the cursor on its toolbar, where there are no buttons, and then operate as if the toolbar were the window title bar. Also, you can access the window control functions through its system menu by pressing the **Alt+<grey minus>** keys.
2. Each window has the **Properties** item in its local menu, which can be invoked by a right click. The **Title** and **Toolbar** items of the **Properties** sub-menu toggle the title bar and toolbar on/off for the individual active window.

4.3.4.5.2 Colors

The **ColorsEnvironment** dialog opens a sub dialog for setting colors of such window elements as window background, font, etc.. By default, most colors are inherited from MS Windows; however you can set other colors if you prefer them.

Element of dialog	Description
Color Scheme	Specifies the color scheme name. Your can type in a name or choose a recently used one from the list.
	The **Save** button saves the current scheme to the disc; later you can restore color settings by just a mouse click. The **Remove** button removes the current scheme.
Colors	Lists the names of color groups. Each group consists of several elements.
Inherit Windows Color	When this box is checked, the selected color is taken from MS Windows. If later you change the MS Windows colors through the Windows Control Panel, this color will change accordingly. This option is available only for the

	background and text colors.
Use Inverted Text/ Background Color	When this box is checked, the program inverts the selected window colors (for text and background). For example, if the **Watches** window background color is white and the text color is black, then the line with the selected variable will be highlighted with black background and white text.
Edit	Opens the **Color** dialog if the **Inherit Windows Color** and **Use Inverted Text/ Background Color** boxes are unchecked for this type of window. The **Color** dialog also opens if you double-click a color in the **Colors**
Spread	Sets the selected color for all windows. This option is useful for text and background colors. For example, if you choose blue background and yellow text for the **Source** window and then click the **Spread** button, these colors will be set as the text and background colors for all windows.
Font	For syntax highlighting in the **Source** window, you can specify additional font attributes - Bold and Italic. In some cases when synthesizing bold fonts, MS Windows increases the size of characters and the font becomes unusable, because the bold and regular characters should be of the same size. In these cases, the Bold attribute is ignored. Sometimes this effect occurs with the Fixedsys font. If you need to use Bold fonts, choose the Courier New font.

4.3.4.5.3 Mapping Hot Keys

Key Mapping tab of the **Environment** dialog opens a sub dialog for assigning hot keys for all commands in the ChipProgUSB. The **Menu Commands TreeKey 1** (**Key 2**) columns contain the corresponding hot–key combinations for the commands. The actions apply to the currently selected command.

Element of dialog	Description
Define Key 1 **Define Key 2**	Opens the **Define Key** dialog. In the dialog, press the key combination you want to assign to the selected command, or press **Cancel**. Alternatively, double-click the "cell" in the row of this command and the **Key 1** (**Key 2**) column.
Erase Key 1 **Erase Key 2**	Deletes the assigned key combination from the selected command. Alternatively, right click the "cell" in the row of this command and the **Key 1** (**Key 2**) column.

4.3.4.5.4 Toolbar

The **Toolbar** tab of the **Environment** dialog controls the presence and contents of toolbars of the windows.

Element of dialog	Description
Toolbar Bands	Lists the ChipProgUSB toolbars. To enable/disable a toolbar check its box.
Buttons/Commands	Lists the buttons for the toolbar selected in the **Toolbar Bands** list. To enable/ disable a button on the toolbar check its box.
"Flat" Local Window Toolbars	Toggles between the "flat" and quasi-3D appearance of the local toolbar buttons for the specialized windows.
Toolbar Settings are the Same for Each Project/Desktop File	Employs the current settings from this dialog for other projects or files opened later.

4.3.4.5.5 Messages

Check messages that program should display, uncheck messages that you do not want to be displayed.

4.3.4.5.6 Miscellaneous Settings

The **Miscellaneous** tab of the **Environment** dialog allows the setting of miscellaneous parameters of the ChipProgUSB windows and messages.

Element of dialog	Description
Main Window Status Line	Controls presence and location of the <%CM%> window status line.
Quick Watch Enabled	Turns the Quick Watch function on or off.
Highlight Active Tabs	Turns highlighting on/off for the currently active tab (the MS Windows-style) in windows that have tabs.
Double Click on Check Box or Radio Button in Dialogs	Sets the mouse's double click function equal to a single click, plus pressing the **OK**
Show Hotkeys in Pop-up Descriptions	Turns the Hotkeys display on/off in the short prompts for toolbar buttons.
Do not Display Box if Console Window Opened	If the **Console** window is open, messages will be displayed there. Otherwise, the message box will display messages.
Always Display Message Box	All issued messages will be displayed in the message box. The **Console** window also displays these messages.

Automatically Place Cursor at OK Button	The cursor will always be on the **OK** button when the message box opens and this box is checked. If you prefer you may press the **Enter** key instead of using the mouse to click **OK.**
Audible Notification for Error Messages	If you select this option, there will be a beep along with the error message. Information (as opposed to error) messages are always displayed without the beep.
Log Messages to File	Specifies the log file name. All messages will be written to this file. The method of writing is controlled by the radio button with two options:
Overwrite Log File After Each Start	Specifies erasing the previous log file, if it exists, and creates it afresh for every session.
Append Messages to Log File	Specifies appending messages to the end of an existing log file. In this case, the log file size will grow endlessly.

4.3.4.6 Configurating Editor Dialog

The ChipProgUSB software includes a **built-in editor** that is used for editing one type of the objects of the ChipProgUSB - **Scripts Files**. The **Editor Options dialog** includes the following tabs:

General Editor Settings tab,

Key Mapping tab.

4.3.4.6.1 General Editor Settings

The **General** tab of the dialog sets up all common options applicable to every **Source** window opened.

Element of dialog	**Description**
	Checking/clearing this box toggles the Backspace Unindent mode. See below for explanations.
Keep Trailing Spaces	When this box is checked, the editor does not remove trailing spaces in lines when copying text to the buffer or saving it to a disk. Spaces are removed when the box is unchecked.
Vertical Blocks	If the box is checked, the Vertical Blocks mode is enabled for block operations.
Persistent Blocks	If the box is checked, the Persistent Blocks mode is enabled for block operations.
Create Backup File	.
Horizontal Cursor	If the box is checked it sets the cursor as a horizontal line, like the DOS command prompt.
CR/LF at End-of-file	If the box is checked, it adds an empty line to the file end when

	saving the file to disk (if there is no one yet).
Syntax Highlighting	If the box is checked, it forces syntax highlighting of language constructions.
Highlight Multi-line Comments	If the box is checked it enables highlighting of multi-line comments. By default, the window highlights only single-line comments.
Auto Word/AutoWatch Pane	If the box is checked, any new window will open with the **Auto Word/ AutoWatch** pane at its right and *the automatic word completion function* will be enabled.
Full Path in Window Title	If the box is checked, the **Source** window caption bar displays the full path to the opened file.
Empty Clipboard Before Copying	If the box is unchecked, then previously kept data remains retrievable after copying to the clipboard.
Convert Keyboard Input to OEM	If the box is checked, the **Source** window converts the characters that you input in the window from the MS Windows character set to the OEM (national) character set corresponding to your national version of the Windows operating system. Also, see note below.
AutoSave Files Each … min	If the box is checked, <%CM%> will save the file being edited every 'X' minutes, where 'X' is a settable constant chosen by the user.
Tab Size	Sets the tabulation size for the text display. The allowable value ranges from 1 to 32. If the file being edited contains ASCII tabulation characters, they will be replaced with a number of spaces equivalent to the tabulation size.
Undo Count	
Automatic Word Completion	If the **Enable** box is checked, it allows the automatic word completion function. The **Scan Range** drop-down list sets the number of text lines to be scanned by the automatic word completion system.
Indenting	Toggles automatic indenting on/off for a new line that is created when you press **Enter**.

Note. You should check the **Convert Keyboard Input to OEM** box only if you are going to type something in the **Source** window when working with a file coded in the OEM character set. If you need only to display such a file, specify the Terminal font for the **Source** window in the **Fonts** tab of the **Environment** dialog: select in the **Windows** list and press the **Define Font** button.

The **Backspace Unindent** mode establishes the editing result from pressing the **Backspace** key in the following four cases, when the cursor is positioned at the first non-space character in the line (there are several spaces between the first column of the window and the first non-space character):

	Backspace Unindent enabled	Backspace Unindent disabled
Insert mode	Any preceding blank spaces in the line are deleted. The rest of the line shifts left until its first character is in the first column of the window.	One space to the left of the cursor is deleted. The cursor and the rest of the line to the right of the cursor shift one position left.
Overwrite mode	The cursor moves to the first column of the window. The text in the line remains in place.	Only the cursor moves one position left. The text in the line remains in place.

4.3.4.6.2 The Editor Key Mapping

You can manage the list of available editor commands with the **Key Mappings** tab of the **Editor Options** dialog. You can add and delete editor commands, assign (or reassign) hot keys for new commands and for built-in ones.

The left column of the list contains command descriptions. Command types, corresponding to the command descriptions, are in the second column. (means a built-in ChipProgUSB command; *Script 'XXX'* means an added user-defined command). Two columns on the right specify the hot key combinations to invoke the command, if any.

Element of dialog	Description
Add	Opens the **Edit Command** dialog for adding a new command to the list and setting up the command parameters.
Delete	Removes a selected user-defined command from the list. Any attempt to remove a built-in command is ignored.
Edit	**Edit Command** dialog to change the command parameters. For built-in commands, you can only reassign the hot keys (the **Command Description** and boxes are not available).
Edit Script File	Opens the script source file of this command in the **Script Source** window.

Creating new commands

To create a new command, you should develop a script for it. In fact, you add this script to the editor, not the command. This means that your command is able to perform much more complex, multi-step actions than a usual editor command. Moreover, you can tailor this action for your convenience, or for a specific work task or other need. Your scripts may employ the capabilities of the script language with its entire set of built-in functions and variables, text editor functions and existing script examples.

A script source file is an ASCII file. To execute your command, the editor compiles the script source file. Note that before you can switch to using the script which you have been editing, you must first save it to the disk so that ChipProgUSB can compile it.

Script source files for new commands will reside only in the KEYCMD subdirectory of the ChipProgUSB system folder. Several script example files are available in KEY CMD. For more information about developing scripts, see **Script Files**.

4.3.4.6.2.1 The Edit Key Command Dialog

This dialog **Edit command** sets parameters for a new command or for existing ones.

Element of dialog	Description
Command Description	Enter the command description here (optional). Text placed in this box will be displayed in the list of commands for easier identification of the command.
Script Name	The name of the script file that executes this command.
Define Key 1	

Define Key 2

The script source files for commands will reside only in the KEYCMD subdirectory of the ChipProgUSB

Notes

1. You should not specify the combinations reserved by Windows (like **Alt+–** or **Alt+Tab**
2. We do not recommend assigning the combinations already employed by commands in the **Source** window or ChipProgUSB, because then you'll have fewer ways to access these commands. Some examples are **Alt+F**, **Shift+F1**, **Ctrl+F7**, which are commands that open the application menus. Others are the local menu hot keys of the editor window.
3. You can use more than one control key in the keystroke combinations. For example, you can use **Ctrl+Shift+F** or **Ctrl+Alt+Shift+F** as well as the **Ctrl+F** combination.
4. For some built-in commands, the hot keys cannot be reassigned (for example, the keys for moving the cursor).

4.3.5 The Commands Menu

This menu invokes main commands (or functions) that control the programming process, as well as some service commands.

Command	Description
Blank Check	This command invokes the procedure of checking the target device before programming to make sure that it is really blank. Programming of some memory devices does not require erasing them before re-programming. For such devices the **Blank Check** command is blocked and it is shown grayed out on the screen.
Program	This command invokes the procedure of programming the target device, e.g. writes the contents of the buffer into the target device's cells.
Verify	This command invokes the procedure of comparing the information taken from the target device with the corresponding information in the buffer.
Read	This command invokes the procedure of reading the content of the target device's cells into an active buffer.
	This command invokes the procedure of erasing the target device. Some memory devices cannot be electrically erased. In this case the Erase command is blocked and is grayed out on the screen
Auto Programming	This command invokes the procedure of AutoProgramming.
Local menu	Opens the local menu of active window.
Calculator	Opens the Calculator dialog, which performs calculator functions.

4.3.5.1 Calculator

A prime purpose of the embedded calculator is to evaluate expressions and to convert values from one

radix to another. You can copy the calculated value to the clipboard.

Element of dialog	Description
Expression	The text box for entering an expression or number.
Copy As	Specifies the format of results that will be copied to the clipboard.
	If this box is checked the result of a calculation will be interpreted and displayed as a signed value (for the decimal format only).
Display Leading Zeroes	If this box is checked, binary and hexadecimal values retain leading zeroes.
Copy	Copies the result to the clipboard in the format set by the **Copy As** radio button.
Clr	Clears the **Expression** text box.
Bs	Deletes one character (digit) to the left of the insertion point (Backspace).
0x	Inserts "0x".
>>	Shifts the expression result to the right by the specified number of bits.
<<	Shifts the expression result to the left by the specified number of bits.
Mod	Calculates the remainder of division by the specified number.

While you are typing the expression in the **Expression** drop-down list box ChipProgUSB tries to evaluate the expression and immediately displays the result in different formats in the **Result** area. Statuses of the **Copy As** radio button and two check boxes in this area control the result format.

You can assign values to program variables and SFRs by typing an expression that contains the assignment. For example, you may type `SP = 66h`

Examples of expressions:

```
0x1234
-126
main + 33h
(float)(*ptr + R0)
101100b & 0xF
```

4.3.6 The Script Menu

The ChipProgUSB is featured with the tools known as an embedded script language. This mechanism is

intended for automation of the programming operation, mastering complex operations that include both the programmer itself and the programmer operator's actions. The ChipProgUSB enables composing scripts files (SF) and executing them.

This Script menu contains a few commands associated with script files. The commands can be configured by the ChipProg user and the list can be expanded by adding a new item (command). To add a new item, place a script file into the current folder or into the ChipProgUSB installation folder. The first non-empty line of any script file should contain three slashes followed by the text that will appear in the **Scripts** menu:

```
/// Menu item text
```

When ChipProgUSB builds the **ScriptsScripts** menu.

When you select a **Scripts** menu item and click the **Start** button, ChipProgUSB launches the selected script.

Button	Command	
	**Start...**	Opens the **Script Files** dialog from which you can
	New Script Source	Create a new **Script File** text.
	**Open Watches window**	Opens the Watches window.
	Add watch...	Add watch to the **Watches window** .
	Editor window	Opens a list of the commands to **Compose a new**, **Open**, **Save, Save as, Print** a script file. of the **Editor** window.
	Text Edit	Edit a list of the commands for editing a selected **Script File**
	Example Scripts	Invokes the
	Help on this menu	

Working with scripts is describe in the topics.

4.3.7 The Window Menu

This menu lets you control how the windows are arranged within the computer screen. The list of currently opened windows is shown in the lower part of the menu. By choosing a particular window name in this list you immediately activate it and bring it to the foreground of the computer screen.

Command	Description
Tile	Arranges all windows without overlap. Makes the window sizes approximately equal.
Tile Horizontally	Arranges the windows horizontally without overlap. Makes the window size as close to each other as possible.
Cascade	Cascades the windows.
Arrange Icons	Arranges the icons of the minimized windows.
Close All	Closes all windows.

4.3.8 The Help Menu

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

4.4 Windows

The ChipProgUSB enables opening the following types of windows by means of the View menu:

- **Program manager**
- **Device and Algorithm Parameters' Editor**
- **Buffer**
- **Device Information**
- **Console**

Plus it can operate with two types of windows associated with the ChipProgUSB script files:

- Editor
- Watches

4.4.1 The Program Manager Window

The **Program Manager window** is the major control object on the screen from which an operator controls the ChipProg . While some windows can be closed in a process of programming this one is supposed to be always open and visible.

The window includes three tabs opening three group of settings and status indicators:

The **Project Manager** and **Options** tabs look different and enable different settings for the ChipProg programmers working in single-programming and multi-programming modes. These tabs are identical for the ChipProg-G4 gang programmer and for the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers when they are configured to work in the multi-programming mode.

4.4.1.1 The Program Manager tab

The tab serves for setting major programming parameters, executing the programming operations and displaying the ChipProg statuses.

Element of dialog	Description
Buffer:	The field **Buffer** displays the active buffer to which the programming operations (functions) will be applied. A full list of open buffers is available here via the drop-down menu.
Functions	This field lists the tree of the functions relevant to the selected target device. Some functions represent the ChipProg commands while others integrate a few sub-functions and can be expand or collapsed. Double clicking on the function invokes the command and is equivalent to single clicking the **Execute** button (see below).
Blank check	Checks if the target device is blank
Program	Programs the target device (writes the information from an active buffer to the target device).
	Reads out the content of the target device to an active buffer.
Verify	Compares the content of the target device and an active buffer
Auto Programming	Executes a preset sequence of operations (batch operations) settable in the Auto Programming dialog. The **Edit Auto** button opens this dialog.

Addresses	Here you can set the addresses for the buffer and the target device to which the programming functions will be applied.
Device start:	The very first address in the target device's physical memory which will be programmed or read.
Device end:	The very last address in the target device's physical memory which will be programmed or read.
Buffer start:	The very first address in the buffer memory from which the data will be written to the target device or to which the data will be read from the device.
	There are three alternative ways to activate a highlighted function: a) to click the **Execute** button; b) to double click on the function line; c) to push the **Enter** button on the PC keyboard.
Repetitions:	Any function can be executed repeatedly. The number of repetitions can be set here.
Edit Auto	Clicking on this button opens the Auto Programming dialog.
Operation Progress	ChipProgUSB displays the current operation progress bar and the operation status (OK, failed, etc.).

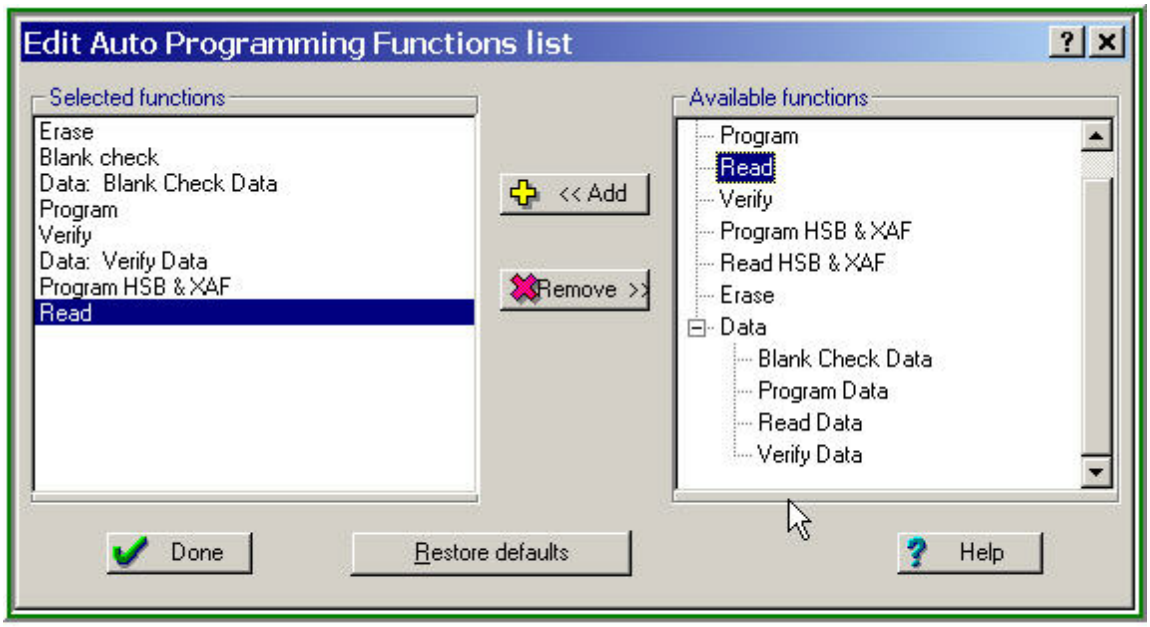
Besides the generic functions () the window **Functions**Device and Algorithm Parameters editor window.

IMPORTANT NOTE!

Any changes made 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.

4.4.1.1.1 Auto Programming

Each device has its own routine set of programming operations that usually includes: **Erasing, Blank Checking, Programming, Verifying** and often **Protecting** against unauthorized reading. The ChipProgUSB stores default batches of these programming operations for each single supported device and allows the invocation of the batch of operations just by a mouse click or pressing the Start button on the programmer panel. It also enables the customization of a sequence of elementary functions (operations) via the **Auto Programming** dialog. To open this dialog click on the **Edit Auto** button.



Adding a Command to the Auto Programming Function List

The tree including all the functions available for the chosen target device is shown in the right pane **Available functions**. To include a function to the batch highlight it in the right pane and click the **Add** button - the function will appear in the left pane **Selected functions.** The functions will be then executed in the order in which they are positioned in the **Selected functions** pane, from the top to the bottom. To correct the function batch highlight the command to be removed and click the **Remove** button.

4.4.1.2 The Options tab

The tab serves for setting additional programming parameters and options:

Element of dialog	Description
Split data	The group of radio buttons in the **Split data** field allows the programming of 8-bit memory devices to be used in the microprocessor systems with the 16- and 32-bit address and data buses. To do this the buffer content should be properly prepared to split one memory file into several smaller file.
Options	
Insert test	If this box is checked the ChipProgUSB will test whether each of the device leads is reliably squeezed by the programming socket contact. If some contact is bad a current operation will be blocked.
Check device ID	By default this option is always on and the ChipProg always verifies the target device identifier given by the device manufacturer. If the box is unchecked the program will skip the device ID checking.

Reverse bytes order	If this box is checked the will sweep the byte order in the 16-bit word while it executes the **Read**, **Program** and **Verify** operations. This option does not affect the data in the ChipProg buffers, as they remain the same after the file loading.
Blank check before program	If this box is checked the ChipProgUSB will always check if the target device is blank before programming it.
Verify after program	If this box is checked the ChipProgUSB will always verify the device content right after it was programmed.
Verify after read	If this box is checked the ChipProgUSB will always verify the device content right after it was read out.
Auto-Detect presence of device in the socket	If this box is checked the ChipProgUSB will test whether each of the device leads is reliably squeezed by the programming socket contact. If so a preset programming function (operation) or Auto Programming will start. Otherwise, if some contact is bad a current operation will be blocked.
On Device Auto-Detect or 'Start' Button:	The group of radio buttons. The checked radio button defines what the ChipProg **'Start'** button.

4.4.1.2.1 Split data

The group of radio buttons in the **Option** tab in the **Split data** field allows programming 8-bit memory devices to be used in the microprocessor systems with the 16- and 32-bit address and data buses. To do so the buffer content should be properly prepared to split one memory file iinto several smaller files. The data splitting enable the conversion of the data read from 16- or 32-bit devices to make file images for writing them to memory devices with the byte organization.

Radio button	Description
No split	This is a default option. A whole buffer is not split and is considered as a whole one byte data array.
Even byte	The data in the buffer are considered as an array of 16-bit words. The buffer-device operations are conducted with **even bytes only**. For example, if the programmer reads the device from the address=0, the byte with this address will be placed to the buffer location also with the address=0, the byte from the device with the address=1 will be placed to the buffer location with the address=2, etc.
Odd byte	The data in the buffer are considered as an array of 16-bit words. The buffer-device operations are conducted with **odd bytes only**. For example, if the programmer reads the device from the

	address=0, the byte with this address will be placed to the buffer location also with the address=1, the byte from the device with the address=1 will be placed to the buffer location with the address=3, etc.
Byte 0	The data in the buffer are considered as an array of 32-bit words. The buffer-device operations are conducted with **the byte #0 only**
Byte 1	The data in the buffer are considered as an array of 32-bit words. The buffer-device operations are conducted with **the byte #1 only**. For example, if the programmer reads the device from the address=0, the byte with this address will be placed to the buffer location with the address=1, the byte from the device with the address=1 will be placed to the buffer location with the address=5, etc.
Byte 2	The data in the buffer are considered as an array of 32-bit words. The buffer-device operations are conducted with **the byte #2 only**. For example, if the programmer reads the device from the address=0, the byte with this address will be placed to the buffer location with the address=2, the byte from the device with the address=1 will be placed to the buffer location with the address=6, etc.
Byte 3	The data in the buffer are considered as an array of 32-bit words. The buffer-device operations are conducted with **the byte #3 only**. For example, if the programmer reads the device from the address=0, the byte with this address will be placed to the buffer location with the address=3, the byte from the device with the address=1 will be placed to the buffer location with the address=7, etc.

4.4.1.3 The Statistics tab

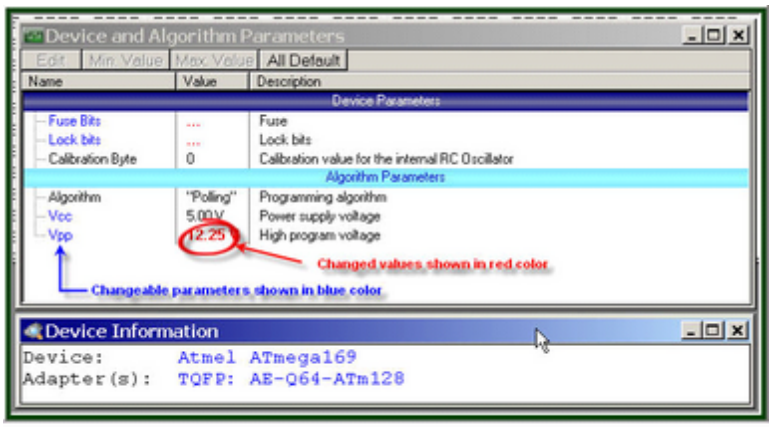
This tab opens the fild displaying the programming session statistical results - **Total** number of devices that were programmed during the session, what was the yield (**Good**) and how many devices have failed (**Bad**Auto Programming only, as execution of other functions makes no effect on the statistics.

Element of dialog	Description
Clear statistics	This button resets the statistics..

Device Programming Countdown	Normally the **Total** counter increments after each Auto Programming; the , **Good** and **Bad** counters also count up. The ChipProgUSB reverses the counters to decrement their content (to count down).
Enable countdown	If the box is checked the ChipProgUSB will count the number of the programmed devices down.
Display message when countdown value reaches zero	If the box is checked the will issue a warning when the counter **Total** is zeroed.
Reset counters when countdown value reaches zero	If the box is checked the ChipProgUSB will reset all the counters when the counter **Total** is zeroed.
Count only successfully programmed devices	If the box is checked the ChipProgUSB will count only the successfully programmed (**Good**). All other statistics will be ignored.
Set initial countdown value	Clicking on the button opens the box for entering a new **Total** number that then will be decremented after each Auto Programming .

4.4.2 The Device and Algorithm Parameters window

The **Device and Algorithm Parameters Editor** window is intended for displaying and editing (where possible) the device's internal parameters and settings, which after editing should be programmed into a target device by executing the **Program** command in the **Program Manager** window.



Device and Algorithm parameters

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

Device Parameters This group includes parameters that are specific for each selected device, such as: **sectors for flash memory devices, lock and fuse bits, configuration bits, boot blocks, start addresses** and other controls for microcontrollers. Usually these parameters represent certain bits in a microcontroller's Special Function Registers (SFRs). Some of these SFRs can be set in the ChipProg buffers in accordance with device manufacturers' data sheets. But setting the parameters in the **Device and Algorithms Parameters** window is much easier and more intuitive. It is impossible to specify absolutely all features that may appear in future devices, and, therefore, new parameters for these new devices.

Algorithm Parameters This group includes parameters of the programming algorithm for the selected device – including the algorithm type and editable programming voltages.
.

IMPORTANT NOTE!

Any changes made 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.

The window is separated into three columns: 1) the parameter's name, 2) its value or setting, 3) a short description. **Names** of the editable parameters are shown in blue; other names are shown in black. Default values in the column **Value** are shown in black; after changing a parameter the new value will be shown in red. If the value is too long to display the window represents it 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. Some editable parameters are represented by a set

of check boxes, some require to be typed in prompt boxes.

The local **Device and Algorithm Parameters Editor** window's toolbar includes a few buttons positioned on the top of the window:

Toolbar button	Description
	Clicking on this button opens the editing dialog to modify the highlighted parameter in the format, most convenient for this parameter. A double click on the highlighted parameter also opens the editing dialog.
Min.Value	If the parameter to be modified has an allowed range in which it may be set, then clicking on the **Min.Value** button sets the minimal allowed value to the highlighted parameter.
Max.Value	If the parameter to be modified has an allowed range in which it may be set, then click on the **Max.Value** button to set the maximal allowed value to the highlighted parameter.
Default	Click on this button returns the default value to the highlighted parameter.
All Default	Click on this button returns the default values to all the parameters displayed in the window.

Depending of the parameter's type ChipProgUSB offers the most convenient format for the parameter editing:

Method of editing	Description
Drop-down menu	When the parameter value may be picked from a few preset values the dialog offers a drop-down list with these values. Highlight a new value in the list and click OK to complete the editing. For example, some microcontrollers can be programmed to work with different types of the clock generators, so the menu prompts to select one of them.
Check Box dialog	When some options can be set or reset the dialog appears in a form of several boxes indicating the default or lately set option statuses. To toggle the option check or uncheck the box. For example, some microcontrollers allow the locking of a particular part of the memory by setting several lock bits, so the menu prompts to check the lock bits represented as a set of check boxes.
Customizing the parameter	When the parameter value may be set freely in an allowed range the dialog offers a box for entering a new value and a history list displaying a few recently set values. The dialog prompts with the min and max values that can be set for each parameter and restricts to enter the value out of the allowed range. This type of editing is in use

for setting custom values for Vcc and Vpp voltages.

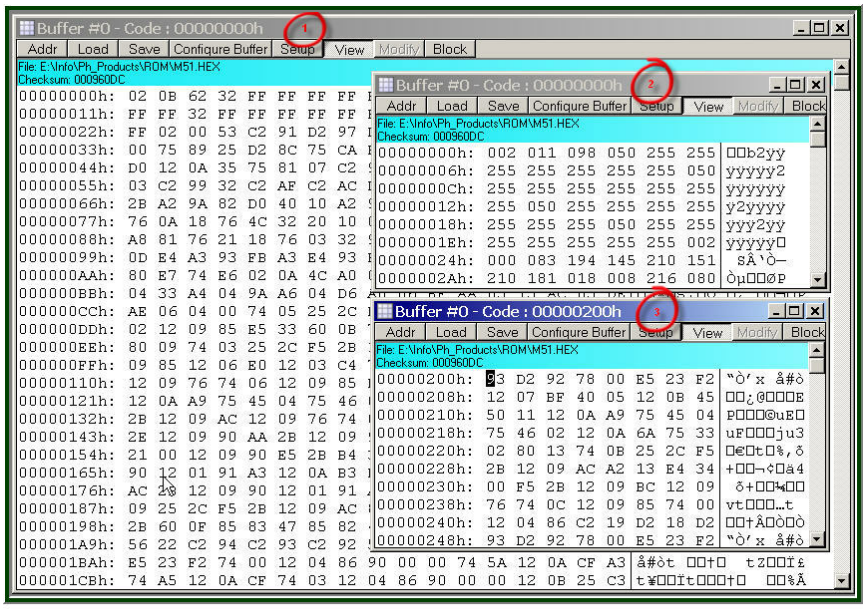
4.4.3 Buffer Dump Window

The **Buffer Dump** window displays the contents of the

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. To open a **Buffer Dump** window, click on the command **Main Menu > View > Buffer Dump**.



Several Windows of Same Buffer

The picture above displays three **Buffer Dump** windows representing three parts of the same buffer:

- #1 (the largest) shows the buffer contents beginning at address 0h;
- #2 shows the same buffer contents beginning at the same address but displaying data in decimal format;
- #3 window shows the data beginning at address 200h.

The left-most column in the windows above 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.

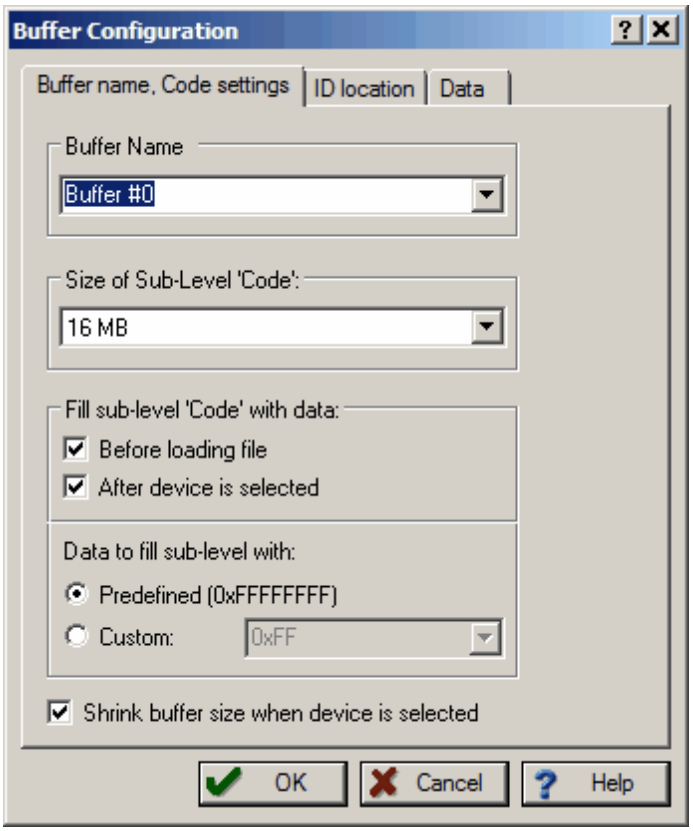
Local menu and Toolbar

The local menu, which can be opened by the right mouse click, includes the **Buffer Dump** window context commands and dialog calls. Most, but not all, of the local menu lines are duplicated by the local toolbar buttons displayed at the top of the window. Here are the local menu and toolbar items:

Menu Command or Call	Toolbar button	Description
New address...	**Addr**	Opens the **Display from address** dialog.
Load file to buffer...	**Load**	Opens the **Load window Dump** dialog.
Save data to file...	**Save**	Opens the **Save window Dump** dialog.
Configure buffer...	**Configure buffer**	Opens the **Configuration Window Dump** dialog.
Window setup...	**Setup**	Opens the dialog.
View only, edit disabled	**View**	By default editing in the buffer dump windows is disabled and you can only view the data. If the box is unchecked the editor will be enabled. Then you may overtype the value under the cursor.
Modify data	**Modify**	Opens the **Modify data** dialog. This call is enabled only when the **View only, edit disabled** is off.
Operations with memory blocks	**Block**	Opens the **Operations with memory blocks** dialog.
Swap fields	No button	This command allows swapping the cursor position between the right and left window panes.

4.4.3.1 The 'Configuring a Buffer' dialog

By default the first opened buffer is named ‘Buffer #0’. The next buffer gets the name ‘Buffer #1’, and so on. You can, however, rename the buffer as you wish.



By default each buffer has a minimal size of 128K RAM in a PC and by default the program fills the buffer with a predefined value (usually 0FFh). You can customize these buffer settings - check the Custom radio button and type in the pattern to fill the buffer.

4.4.3.2 The 'Buffer Setup' dialog

The dialog allows controlling the data presentation in the **Buffer Dump** window. You can open the dialog using the windows local menu (the **Windows Setup** command) or by clicking the **Setup** button on the window toolbar.

Element of dialog	Description
Buffer:	The field displays a list of all open buffers. The programming functions will be applied to the active one.
Display Format	
Display Data As:	Is represented by four radio buttons. Here you can select the data presentation format in the buffer: 1, 2, 3 or 4 Byte.

Options	The options here customize the display format.
ASCII pane	If the box is checked the right pane will display ASCII characters corresponding to the data in the buffer dump.
Display checksum	If the box is checked the calculated checksum will be displayed in the blue strip over the data dump, right under the window local toolbar.
Limit dump to sub-layer size	If the box is checked the window dump will display a part of memory equal to the active sub-layer's size.
Signed decimal and hex values	If the box is checked the most significant bit (MSB) in the data shown in the binary or hexadecimal formats will be treated as a sign. If MSB=1 the data is negative, if MSB=0 they are positive.
Always display '+' or '-'	This is a sub-setting for the **Signed decimal and hex values** option. If both boxes are checked then the signs '+' and '-' will be displayed.
Leading zeroes for decimal numbers	If the box is checked then each decimal data will be shown with a number of zeros before the first significant digit - for example the value of 256 will be presented as 00000256.
Reverse bytes in words (LSB first)	If the box is checked then the order of bytes in words will be reversed, e.g. the MSB will follow the LSB.
Reverse words in dwords	If the box is checked then the order of 16-bit words in 32-bit words will be reversed.
Reverse dwords in qwords	If the box is checked then the order of 32-bit words in 64-bit words will be reversed.
Non-printable ASCII characters	The characters from the ranges **0x00...0x20** and **0x80...0xFF** are non-printable. The options here customize presentations of non-printable ASCII characters in the ASCII pane of the buffer dump window.
Replace characters 0x00...0x20	If the box is checked then all the characters belonging to the range **0x00...0x20** will be replaced with the character dot ('.') or space (' '). The pair of toggling radio buttons **Replace with:** sets the replacement character - dot ('.') or space (' ').
Replace characters 0x80...0xFF	If the box is checked then all the characters belonging to the range **0x80...0xFF** will be replaced with the character dot ('.') or space (' '). The pair of toggling radio buttons **Replace with:** sets the replacement character - dot ('.') or space (' ').

4.4.3.3 The 'Display from address' dialog

The dialog enables setting a new address that will become the first address of the visible part of the **Buffer Dump** window.

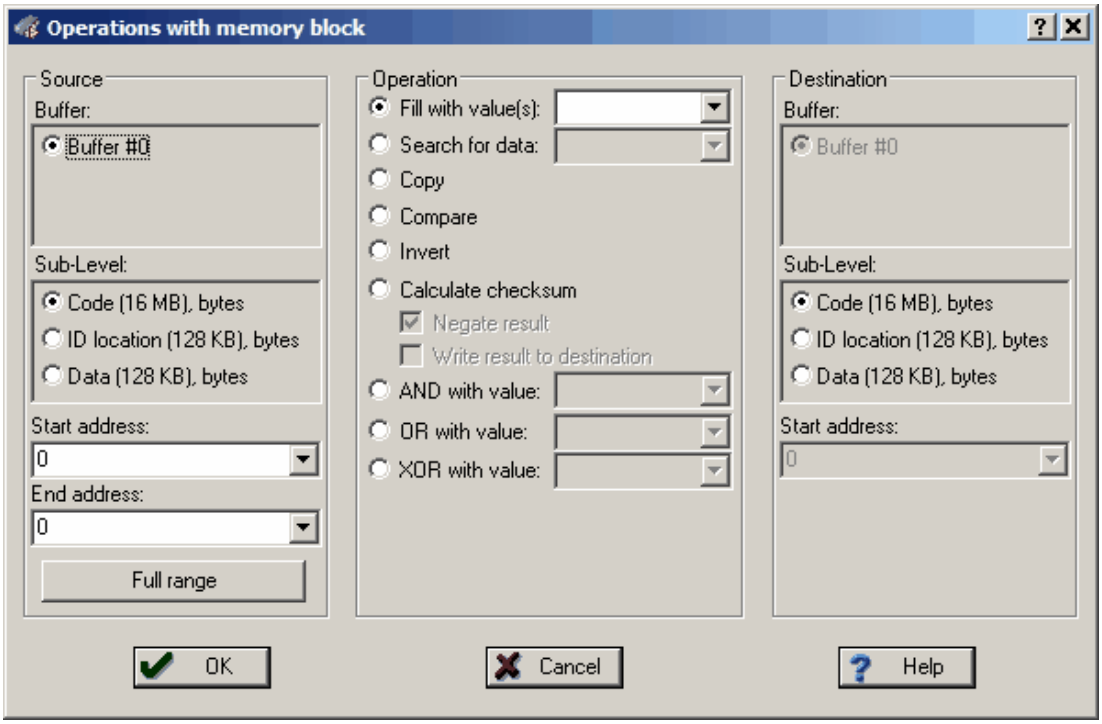
Element of dialog	Description
Type new address to display from:	Here you may enter any address within the allowed range.
History	Displays the list of previously set addresses. Here you can pick one for displaying the buffer dump.

4.4.3.4 The 'Modify Data' dialog

The dialog enables editting the data in the **View** button on the window's toolbar if off, otherwise the editing is blocked. To modify particular data in the buffer appoint the location by a cursor and click the **Modify** button on the window's toolbar. Then enter a new data value in the pop-up box or pick one from the history list. Or, alternatively, appoint the location by a cursor and type over the new data on the PC keyboard.

4.4.3.5 The 'Memory Blocks' dialog

The ChipProgUSB program allows complex operations with memory blocks. This dialog controls operations with blocks of data within one selected buffer or between different buffers.



The dialog box splits in three columns. The **Source** parameters, shown in the left column, specify the source memory area for the operations shown in the middle column. The operation's result will be placed in the area specified by the **Destination** shown in the right column. By default the destination is equal to the source space. Two operations – Fill and Search - do not require a destination address so the dialog disables the **Destination** radio button if these two operations are chosen.

Element of dialog	
Start Address (of the Source)	The start address of the memory area in the selected **Source buffer**, to which the operation will be applied.
	The memory area's end address. It can be set only for the **Source**. After the source address range is defined, the program automatically calculates the destination area's end address.
Full Range (of the Source)	Sets the start and end addresses equal to the entire address space of the selected target device.
Start Address (of the Destination)	The start address of the memory area in the **Destination buffer** where the result of the chosen **Operation** will be placed to.

The following operations are available through this dialog. Each operation starts when you click **OK** in the dialog box. (see notes below).

Operation	**Description**
Fill with Value	Fills the source buffer with a value (or a sequence of values) specified in the text box at the right.
Search for Data	Searches the source memory area for a particular value (or a sequence of values) specified in the text box at the right.

Copy	Copies a specified area of memory to a new destination address. The block can be copied within the same address space or to another one.
Compare	Compares contents of the specified source and destination memory areas. The sizes of the source and destination areas are equal. If there's a mismatch, the mismatch message box will require permission to continue the comparison.
Invert	Inverts the selected source area contents bit-wise and places the results in the destination area.
Calculate Checksum	Calculates the checksum, as a 32-bit value, for the source area of memory. The calculation is done by simple addition. See the note below.
Negate Result	If the box is checked then a checksum, calculated as a 32-bit value by simple addition, will be then subtracted from zero (this is a known method of the checksum calculation).
Write Result to Destination	If this box is checked a calculated 32-bit checksum will be written to the destination sub-level beginning at a specified destination **Start Address**. If this box is cleared the checksum will be displayed as a message only.
	Performs bit-wise AND operation on the contents of the specified source memory locations with the operand specified in the text box on the right and places the results in the destination. See notes below.
OR with Value	Performs bit-wise OR operation on the contents of the specified source memory locations with the operand specified in the text box on the right and places the results in the destination.
XOR with Value	Performs bit-wise XOR operations on the contents of the specified source memory locations with the operand specified in the text box on the right and places the results in the destination.

Notes

1. The source and destination memory areas may overlap. But, since operations with memory blocks are carried out using a temporary intermediate buffer, the overlap does not corrupt the results.
2. The **Copy** and **Compare** commands use the blocks specified in the **Source** address space and the **Destination** address space.
3. The checksum is calculated as a 32-bit value by simple addition. If a memory space has byte organization, then 8-bit values will be added. If it has word organization then 16-bit values will be added.
4. Logical operations (AND, OR, XOR) are performed with the contents of the **Source** address space, while the operation result will be written to the **Destination** address space. The program takes care of converting the operands to the appropriate memory size for a selected type of memory (16-bit for the **Prog, Data16, Reg** and **Stack** memory, 8-bit for the **Data8** memory).

4.4.3.6 The 'Load File' dialog

The dialog defines parameters of the file to be loaded to the buffer.

Element of dialog	Description

File Name:	Enter a full path to the file in this box, pick the file name from a drop-down menu list or browse for the file on your computer or network.
File Format:	**format** of the file to be loaded can be selected here by checking one of the radio buttons in the field of the dialog.
Buffer to load file to:	Select the buffer in which the file will be loaded by checking one of the **Buffer#** radio buttons. There may be just one such button.
Layer to load file to:	The **Buffer to load file to** can have more than one memory layer. Select the layer in which the file will be loaded by checking one of the radio buttons. There may be just a single button available for choosing.
Start address for binary image:	**Binary** file format do not carry any address information and are required to define the start address for the loading. If the file to be loaded is a binary image enter the start address in the box here.
Offset for loading address:	Files in any formats, except the **Binary** file format, can carry the information about the start address for the loading. If the file to be loaded is not a binary image enter the offset for the file addresses in the box here. The offset can be positive or negative.

4.4.3.6.1 File Formats

The ChipProgUSB program supports a variety of file formats that can be loaded to the ChipProg buffers.

File format	Description
Standard/Extended Intel HEX (*.hex)	The Intel HEX file is a text file, each string of which includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information. The ChipProgUSB
Binary image (*.bin)	The binary image includes the data to be loaded only. These data will be loaded to the buffer beginning from a specified start address.
Motorola S-record (*.hex, *.s, *.mot)	The Motorola S-record is a text file, each string of which includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information. The ChipProgUSB loader supports all kinds of the Motorola S-records with the extensions .hex, .s, .mot.
Altera POF (*.pof)	The Altera POF-file is a text file, each string of which includes the beginning address to load the data to the buffer, the data to load,

	checksums for the string and some additional information. The format is mostly used for programming PALs and PLDs.
JEDEC (*.jed)	This format is used for programming PALs and PLDs. The JEDEC-file includes the beginning address to load the data to the buffer, the data to load, test-vectors and some additional information.
Xilinx PRG (*.prg)	The Xilinx PRG-file is a text file, each string of which includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information. The format is used for programming the Xilinx PLDs.
Holtek OTR (*.otp)	This format is presented by Holtek company. The OTP-file includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information.
Angstrem SAV (*.sav)	This format is presented by Angstrem company (Russia). The SAV-file includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information.
ASCII Hex (*.txt)	The ASCII TXT-file includes the beginning address to load the data to the buffer, the data to load, checksums for the string and some additional information.

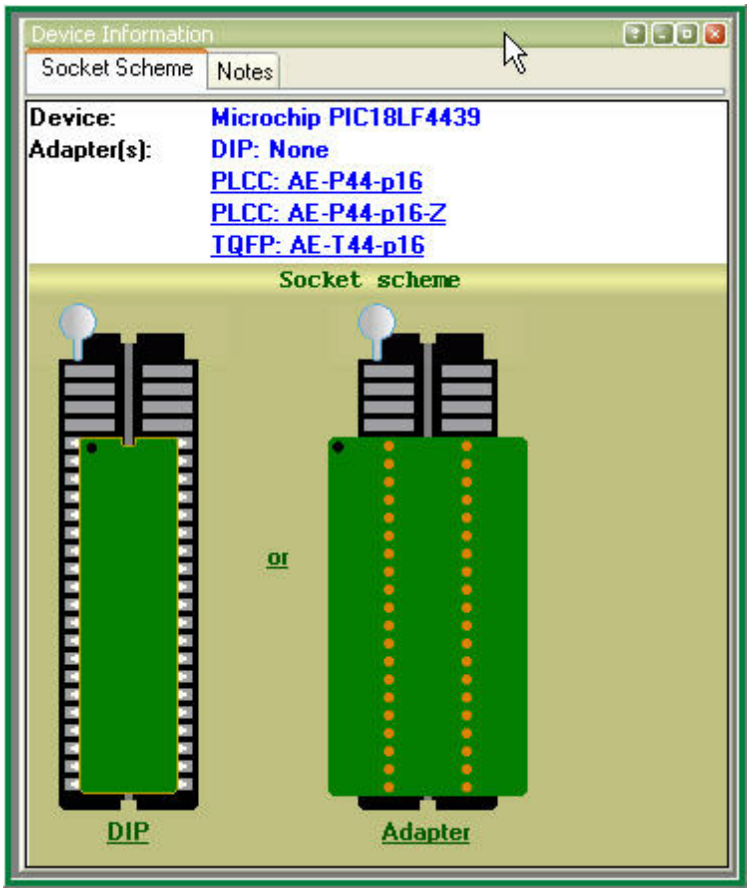
4.4.3.7 The 'Save File' dialog

The dialog defines parameters of the file to be saved from the buffer.

Element of dialog	Description
File Name:	Enter a full path to the file in this box, pick the file name from a drop-down menu list or browse for the file on your computer or network.
Addresses	**Start and End Addresses** define the buffer data space that will be saved in the **File**. For saving an entire buffer click the **All** button.
File Format:	The **format** of the file to be saved can be selected here by checking one of the radio buttons in the **File Format** field of the dialog.
Buffer to save file from:	Select the source buffer from which the file will be saved by checking one of the **Buffer#** radio buttons. There may be just a single button available for choosing.
Sub-level to save file from:	The **Buffer to save file from** can have more than one memory layer. Select the source layer by checking one of the radio buttons. There may be just a single button available for choosing.

4.4.4 The Device Information window

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 below shows all Phyton adapters available for the selected PIC microcontroller. The Socket scheme pictograms below show the correct positions of a DIP-packaged 40-pin PIC chip and the adapter board into a 48-pin ZIF socket (for the ChipProg -48 programmer).



The adapter part numbers are linkable and the links being clicked opens the **adapters.chm** file with a description and wiring diagram of the chosen adapter. The cable adapters for in-system programming are also included into the **adapters.chm** file. There are some peculiarities that such ISP adapters use depending on the target device type.

4.4.4.1 Phyton programming adapters

The adapters.chm file includes short descriptions of the Phyton programming adapters and their wiring diagrams. Having the adapter diagram a ChipProg user can master it is own adapter or to find the adaptor available from a third party, which can be used as a replacement for the Phyton brand adapter.

The adapters diagram are presented in a table form, where the rows show connections of the elements installed on the adapter transition board and the columns (from the left to right) represent:

1st column - Pin numbers of the dual-row pins pluggable to the programmer ZIF socket
2nd column - Pin numbers of the ZIF socket installed on the adapter top
3rd, 4th, 5th, etc. - Pin numbers of the passive and active components installed on the adapter board.

See an example of the AE-P44-A32/64 adapter connection table below:

Pin# of the dual-row 40-pin plug (ChipProg ZIF socket)	Pin# of the PLCC 40-pin adapter socket	74HC14 latch	C1 (.1uF)	C2 (.1uF)
1	2			
2	4			
3	6			
4	28			
	29			
6	9			
7	10			
8	11			
9	12			
	40			
11	7			
12	13			
13	14			
14	16			
15	17			
16	18			
17	19			
18	20			
19	21			
20	22,30,42		1	1
21	24			
22	25			
23	26			
24	27			
25	8			
26	31			
27	32			
28	33			

	34			
30	5			
31	36			
32	35,3,15,23	14	2	
33	37			
34	38			
35	39			
36	40			
37	41			
38				
-	43	12		
39	44			2
40	1			
		10,13		

4.4.4.2 Adapters for in-system programming

The adapters.chm file includes short descriptions of the Phyton programming adapters for in-system programming (e.g. the programming in the user's equipment) and their wiring diagrams the schematic of connecting the adapter cables to the target. The cable adapters may have 10 to 20 pin headers to be connected to the pins or complimentary connectors installed in the user's equipment. The pin connection is specific for certain target devices. The connection diagrams are presented in a table form, where the columns (from the left to right) represent:

1st column - Pin numbers of the cable adapter header inputs and outputs
2nd column - Signals of the target device to be connected

As an example see below a schematic of connecting a 10-pin header BH-10 of the Phyton AE-ISP-U1 cable adapter to the Zilog Z8Fxxx microcontroller for in-system programming.

BH10	Z8Fxxxx
1	**Vcc**
2	**RESET**
3	**GND**
4	**DBG**
5	**GND**
6	
7	
8	
9	
10	

As you can see here not all the BH10 lines should be necessarily used. Only five signals are required for programming this device and only two of them are used for sending the the programming signals into the chip - RESET and DBG. The diagrams in the adapters.chm file use the mnemonic of signal from the device manufacturers' data sheets.

4.4.5 The Console Window

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

4.4.6 Windows for Scripts

ChipProgUSB is featured with the windows specifically supporting operations with scripts. That includes:

- **(Script) Editor** windows
- **Watches**
- **User** windows
- **I/O Stream** windows

These windows cannot be open from the **View menu**; they can be opened only when you work with scripts. Operations with these windows are described in the chapter **Scripts Files**.

5 Operating with Programmers

The topics included in this chapter briefly describe basic operations with the ChipProg programmers.

5.1 Inserting devices to a programming socket

Inserting devices in DIP (dual-in-line) packages.

The -40, ChipProg-48 and ChipProg-G4 programmers are equipped with 40- or 48-pin ZIF sockets allowing operating on any DIP-packed devices without additional adapters. They can accommodate DIP-packed devices with different number of leads (from 4 to 48) and different widths of the package up to 600 mil. Just a few old DIP-packed devices require special adapters to be programmed by ChipProg **Device Information** window prompts if some adapter is required for the selected device and, if so, it displays the adapter type. The pictogram showing a correct insertion position of the device is on the programmer at the left of the socket as well as in the **Device Information** window. Practically all DIP-packed devices can be inserted in the way shown on the pictogram. However, there are a few old devices with a non-standard insertion positioning. If such a device is chosen the **Device Information** window displays how to insert the device.

Inserting devices in non-DIP packages.

Programming of the devices in SOIC, PLCC, QFP, BGA and other non-DIP packages requires special adapters. The adapters design allows plugging them into the programmer ZIF sockets. The **Device Information** window prompts the adapter type for a selected device.

Any adapter is implemented as a small transition board with two rows of dual-in-line pins pluggable into the programmer ZIF socket on a bottom side and a ZIF socket of a particular type (SOIC, PLCC, QFP, BGA, etc.) on a top. The adapter transition board is labeled with a "#1 pin" key mark that helps to properly position the adapter into the programmer socket. The **Device Information** window displays the adapter position into the programmer ZIF socket.

5.2 Auto-detecting the device

If you checked the **AutoDetect** checkbox on the main window toolbar then a ChipProg programmer will automatically detect insertion of the device into a programming socket and will check if the device's leads are reliably squeezed by the socket contacts. In case of the bad contact with any single lead the programmer blocks further operations and issues a warning that indicates the pin numbers with bad contacts. This prevents destroying the device or incorrect programming.

The **AutoDetect** signal can be used for triggering a programming operation by checking the **Auto-Detect presence of device in the socket** box in the **Options** tab of the **Program Manager** window. One of the following options can be set here:

- Execute the function selected in the 'Function' list (the **Program Manager** tab);
- **AutoProgramming**
- Execute script.

At this point the **AutoDetect** trigger replaces the programmer command executed by a mouse click or pressing the **Start** button. This significantly speeds up and simplifies programming of the device series.

5.3 Basic programming functions

Sub-topics of this chapter describe all the basic ChipProg-40 and -48 operations in a **single programming mode**, when a device is programming in the programmer socket. Specific operations for programming more than one device at one time are described in the Multi- and Gang programming

5.3.1 How to check if a device is blank

1. Select the target device type, pressed the button **Select Device** in the Main toolbar or select the command **Main menu** > **Configure** > **Select device**.

2. Insert a device of the selected type into the programmer socket or into the adapter socket.

3. a) Click the **Check** button on the main toolbar or
   b) Double click on the **Blank check** function line in the **Function** list of the **Program Manager** window or
   Blank check function line in the **Function** list of the **Program Manager** window and click the **Execute** button or
   Main menu > **Commands** and click on the **Blank check** line

   then wait for the message **Checking ... OK** in the **Program Manager** window, or for the warning message if the device is not blank

5.3.2 How to erase a device

1. 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).

2. If the device is electrically erasable:
   a) Click the **Erase** button on the main toolbar or
   b) Double click on the **Erase** function line in the **Function** list of the **Program Manager** window or
   c) Select the **Erase** function line in the **Function** list of the **Program Manager** window and click the **Execute** button or
   d) Select the **Main** menu > **Commands** and click on the **Erase** line

   then wait for the message **Erasing ... OK** in the **Program Manager** window or for the warning message if the device is not blank after erasing.

5.3.3 How to program a device

In order to program a blank device you need to perform a few consecutive operations:

- load the file
- (if necessary);
- the device to be programmed (if necessary);
- write the prepared information into the device and verify the programming.

5.3.3.1 How to load a file into a buffer

1. Select the **Main menu** > > **Load** or click the **Load** button on the local toolbar of the **Buffer** window.

2. In the pop-up dialog box enter the source file name, select the file format, addresses, buffer and sub-level to load the file to.

3. Wait for the message **File loaded: "....."** in the **Program Manager** window or for a warning message if the file cannot be loaded for some reason.

5.3.3.2 How to edit information before programming

1. **Buffer Damp** window. Never forget that the **View** button should be released to enable editing.

2. Make necessary changes in the window via the **Modify** dialog or appoint the data to be modified and type the new data over the old data.

5.3.3.3 How to configure the chosen device

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 other parameters that should be changed. All preset changes will become effective in the target device only upon programming via the **Program Manager** programming function.

5.3.3.4 How to write information into the device

1. Click the **Options** tab in the **Program Manager** **window**. Check the options you need. We recommend that you always check the **Blank check** before programming and the **Verify** after programming check-boxes to make programming more reliable.

2. Click the **Program Manager** **tab**. Select the **Program** line in the **Function box**, and double click it to start programming of the primary memory layer (**Code**) or click the **Execute** button to do so. Alternatively, you can do the same by clicking the big **Program** button or selecting the command **Menu** > **Commands** > .

   Wait for the message **Programming ... OK** in the **Operation Progress box** of the **Program Manager****tab**. If an error has occurred the ChipProgUSB issues an error message.

4. Execution of the main **Program** function (always shown in the beginning of the **Function list**) writes a specified buffer layer content to the **Code** device memory. However, other buffer layers may exist for the selected device (**Data**, **User**, etc.). If more than one buffer layer exists for the selected device go down to the list of functions, expand those that are collapsed and execute the **Program** functions for as many types of memory as the device has (**Data**, **User**, etc.). Skip this if just one memory layer **Code** exists for the device.

IMPORTANT! After programming of all the memory layers (**Code, Data**, **User**, etc.) you need to program the options preset in the **Device and Algorithm Parameters Editor window**, if they have been modified. Go down to the **Device parameters & ID** line, expand it if collapsed, select the **Program** function and double click it.**Device and Algorithm Parameters window**

6. Some microcontrollers can be protected against unauthorized reading of the written code by setting a set of **Lock bits**. Go down to the **Lock bits** line, expand it if collapsed and double click the lock bit# lines one by one. You can optionally lock only certain parts of the device memory. Continue until every lock bit is set.

7. After every operation above make sure that you watch the **Ok [xxxxx... Ok]** message in the **Operation Progress** box of the **Program Manager tab**. In case you get an error message stop the programming and troubleshoot the issue.

5.3.4 How to read a device

There are several ways for reading the device content to an active buffer:

a) Click the **Read** button on the main toolbar or
b) Double click on the **Read** function line in the **Function** list of the window or
c) Select the **Read** function line in the **Function** list of the **Program Manager** window and click the **Execute** button or
d) Select the **Main** menu > **Commands** and click on the **Read** line

then wait for the message **Reading ... OK** in the **Program Manager** window or for the warning message if the device could not be read out.

5.3.5 How to verify programming

There are several ways for checking if the device was programmed correctly:

a) Click the **Verify** button on the main toolbar or
b) Double click on the **Verify** function line in the **Function** list of the **Program Manager** window or
c) Select the **Verify** function line in the **Function** list of the **Program Manager** window and click the **Execute** button or
d) Select the **Main** menu > **Commands** and click on the **Verify** line

then wait after that which **Verifying ... OK**Program Manager window or for the warning message if the device failed during the verification process.

5.3.6 How to save data on a disc

1. After you have read out the device content into the Buffer or a specified Buffer layer you may need to save the read data on a PC disc. To save the data:
   a) Click the **Save** button on the local toolbar of the **Buffer** window or
   b) Select the **Main menu** > **File** > **Save**

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.

5.3.7 How to duplicate a device

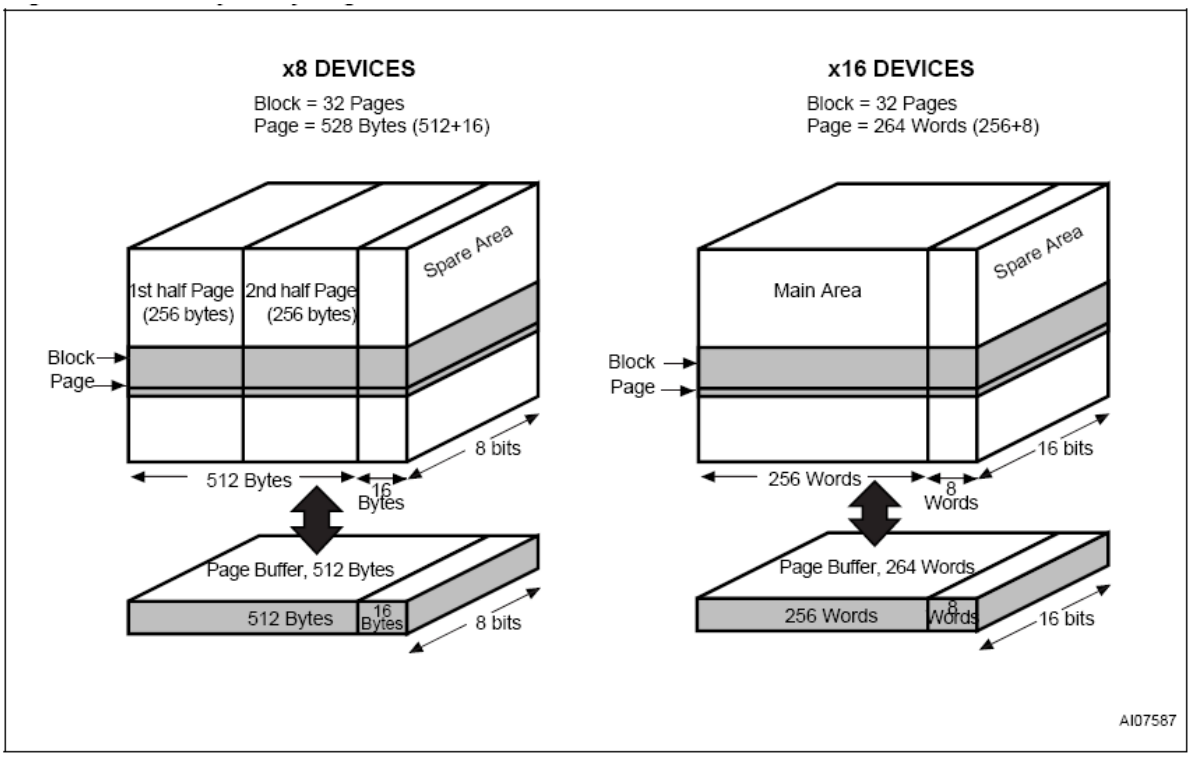
1. Insert the master device to be copied (duplicated) into the programmer socket.
2. **Read** it to an active buffer
3. Wait for the message **Reading... OK** in the **Operation Progress box** of the Program Manager tab in the **Program Manager** window. Make sure the master device content is in a current buffer.
4. Remove the master device from the socket and replace it with a blank device to be programmed. If necessary, check to see if it is blank.
5. **Program** the device. If you need to make more than one copy of the master device repeat the operations #4 and #5 as may times as necessary.

5.4 Programming NAND Flash memory

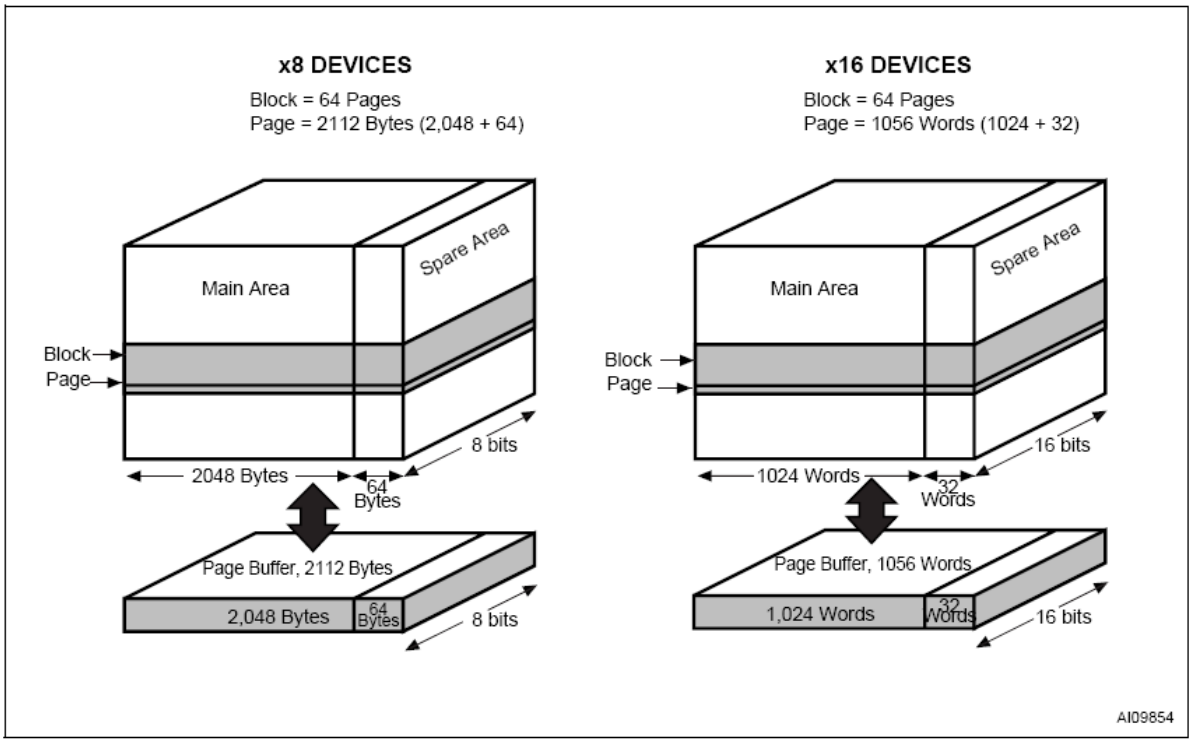
This chapter describes some peculiarities of the NAND Flash memory devices programming. The NAND Flash and NOR Flash memory architectures and physical implementations are very different and, therefore, operations with NOR and NAND Flash devices have their own peculiarities. In terms of the programmer setup and operations, working with the NAND Flash devices is more complex and the programming results are very sensitive to the accuracy of the programming options setup. Inaccurate setup causes wrong device programming.

5.4.1 NAND Flash memory architectures

The NAND Flash memory array comprises of the blocks of pages. Each block usually includes 16, 32, 64 and more pages. Conditionally, the NAND Flash devices can be divided in two groups: the "small page" and "large page" devices. The "small page" size is 512 bytes for the 8-bit devices and 256 bytes for the 16-bit devices; the "small page" NAND Flash memory devices' capacity varies from 128K to 512K bits. The picture below shows the "small page" NAND Flash memory architecture of the STMicroelectronics™ NAND devices.



The "large page" size is 2048 bytes for the 8-bit devices and 1024 bytes for the 16-bit devices; the "large page" NAND Flash memory devices' capacity varies from 256K to 32G bit capacity and higher. The picture below shows the "large page" NAND Flash memory architecture of the STMicroelectronics™ NAND devices. The latest "large page" NAND Flash devices have as large as 4096 byte page size.



Read also about bad blocks in the NAND Flash memory devices.

5.4.1.1 Invalid blocks

NAND Flash memory devices have invalid memory blocks that cannot be used for storing data because some memory cells inside of the device have physical defects - either inherent in a process of the device manufacturing or acquired in a process of the device exploitation and reprogramming in the user's equipment. Since a percentage of invalid blocks is pretty small inside of the chip (usually less than 1%) it is possible to use the device for data storing. In order to use NAND devices with bad blocks these blocks should be marked in a certain way to prevent fetching data from these blocks or writing in it. This document equally uses both known terms for such blocks: invalid and bad.

Locations of the invalid blocks or the invalid blocks map should be accessible by the application for skipping the bad blocks or handling them in other way. To keep the invalid block map every NAND Flash device has a special cell array, known as the for storing addresses of invalid blocks. See the **Spare Area** location in the NAND Flash memory architecture

The Spare Area in "small page" 8-bit devices is 16 large, 16-bit devices - 8 Words. The Spare Area in "large page" devices - 64 Bytes and 32 Words respectfully. Though the Spare Area is dedicated for marking bad blocks it can be also used as a general purpose memory for storing the user's data. To avoid accidental losing of the bad block map it is recommended to assign a whole entire Spare Area for storing the invalid block map and do not write in this area anything else.

5.4.1.1.1 Managing invalid blocks

There are three mostly used methods of handling invalid memory blocks:

Skip Block method

Error Checking and Correction

The ChipProg programmers support all the methods above.

5.4.1.1.1.1 Skipping invalid blocks

This is the simplest method of managing invalid blocks. The programming algorithm first reads the entire **Spare Area** to collect the addresses of invalid memory blocks. Then, the programming equipment writes data to the device page by page with checking the block addresses. If the current block's number is marked as bad the programmer skips this block and write into the next valid one.

5.4.1.1.1.2 Reserved Block Area

This method is based on the idea of replacing invalid blocks with good blocks by re-directing reading and writing operations to these good blocks. To implement this method the programming equipment splits the entire memory in three areas following each other from the start address of the memory device. Each of these areas may include both good and bad blocks:

- **User Block Area** (UBA) - a linear memory array for storing the user's data;

- The **Block Reservoir** - a linear memory array that follows right after the User Block Area; good blocks from the Block Reservoir replaces invalid blocks from the User Block Area;
- The **Reserved Block Area** (RBA) - this part of the device's memory stores the information about bad blocks in the User Block Area replaced by good blocks from the Block Reservoir. This map is represented by pair of addresses of the invalid UBA's blocks and corresponding good blocks from the data reservoir. The first good block in the RBA stores the the RBA map table, the second a duplicate of it in case of the RBA table corruption.

The programming algorithm works in the following way:
1) it splits blocks of the device in three areas: **User Block Area, Block Reservoir** and **Reserved Block Area;**
2) it reads the **Spare Area**

Field:	RBA Marker	Count Field	Invalid Block	Replaced Block	Invalid Block	Replaced Block	.....	Invalid Block	Replaced Block
Size:	*2 байта*	*2 байта*	2 байта	2 байта	2 байта	2 байта	.....	2 байта	2 байта

RBA Marker - is 0FDFEh (there is an equivalent term for this parameter used in some NAND Flash device data sheets: **Transition Field**).
Count Field - starts from 1 and increments by one for each page of the map table.
Invalid Block - Number of the invalid block in the UBA being replaced.
Replaced Block - Number of the valid block in the Block Reservoir that replaces the invalid block above.

The **Invalid Block - Replaced Block** pairs follow each other till the page break.

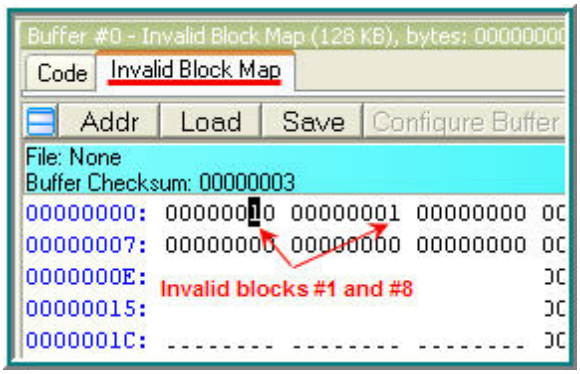
When the programming equipment detects an invalid block in the User Block Area it appoints the first available valid block in the Block Reservoir and updates the RBA table to keep track of relation between invalid blocks in the User Block Area and replaced good ones in the Block Reservoir.

5.4.1.1.1.3 Error Checking and Correction

To maintain the stored code integrity it is recommended to use known **Error Checking and Correction** (ECC) algorithms. Most NAND Flash device manufacturers publish application notes that describe the ECC algorithms suitable for using their devices in different applications. To implement a particular ECC algorithm please check the manufacturer's website. All the ECC-related information are written into the **Spare Area**.

5.4.1.1.2 Invalid block map

ChipProg programmers create the invalid block map into the buffer layer **Invalid Block Map** as a continues bit array. Valid (good) blocks are represented by zeros (0), invalid (bad) - by ones (1). See the tab **Invalid Block Map** in the memory buffer:



For example above:

- the value 02h (or 00000010B) at the address 0 means that the blocks #0, 2, 3, 4, 5, 6, 7 are valid while the block #1is invalid;
- the value 01h (or 00000001B) at the address 1 means that all the blocks in the range #9 to #15 are valid while the block #8 is invalid.

5.4.1.2 Marking invalid blocks

After the device final testing the device manufacturer' programming equipment fills the working memory cells with the FFh value. Blocks that are considered to be invalid are marked by writing a non FFh value (usually 00h) at a certain address in first page (page #0). This address in the NAND Flash Spare Area is the device dependant; it is specified in the manufacturer data sheet.

Memory organization	**The marker address in the Spare Area**
8-bit array, page size - 512 Byte.	5
16-bit (word) array, page size - 512 Words.	0
8-bit array, page size - 2048 Byte.	0 or 5
16-bit (word) array, page size - 1024 Words	0

Take in account that the device itself has no special protection against occasional erasing of the Spare Area cells when you intentionally erase a whole memory array. However, these Spare Area cells may store the bad blocks markers written ether by the chip manufacturer or by the chip user after reprogramming. Being lost the bad block map cannot be restored unless you keep the invalid block map as a file, etc. It is important to keep track of the invalid block map changes by storing the markers before the memory erasing and restoring them after the chip erasing. The ChipProg programmers automatically restore the invalid block map unless the Invalid Block Management is not the **Do Not Use**.

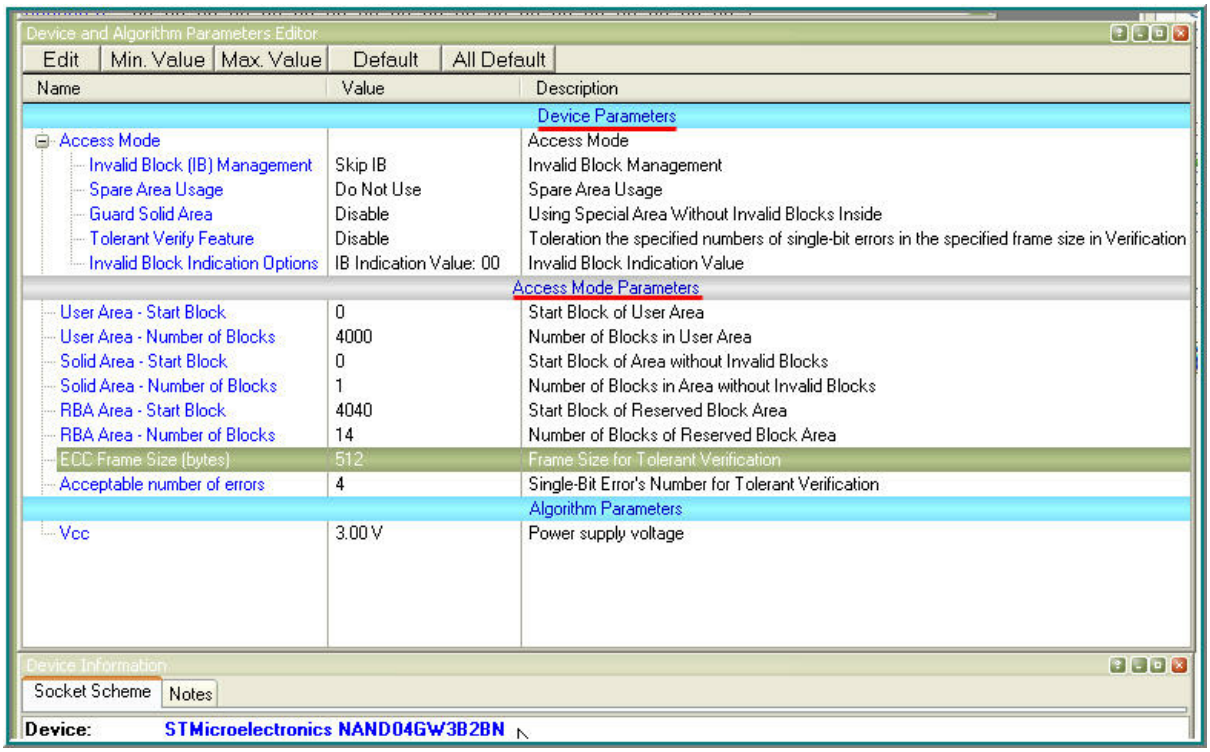
The ChipProg creates the invalid block map into the buffer layer **Invalid Block Map** as a continues bit array. Valid (good) blocks are represented by zeros (0), invalid (bad) - by ones (1). For example:

the value 02h at the address 0 means that the blocks #0, 2, 3, 4, 5, 6, 7 are valid while the block #1is invalid;
the value 01h at the address 1 means that all the blocks in the range #9 to #15 are valid while the

block #8 is invalid.

5.4.2 Programming NAND Flash devices by ChipProg

Programming NAND Flash memory devices by a Phyton ChipProg programmer begins from accurate setting of the programming options and parameters in the **Device and Algorithm Parameters** window. The screen capture below shows the window for the NAND04GW3B2BN device. The **Device Parameters** are divided in two setting groups: **Access Mode** and **Access Mode Parameters**.



IMPORTANT NOTE!

Any changes made 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.

5.4.2.1 Access Mode

The **Access Mode** line, normally collapsed, can be expanded to invoke setting dialogs for one of the following modes:

Invalid Block Management
Spare Area Usage

Tolerant Verify Feature
Invalid Block Indication Option

5.4.2.1.1 Invalid Block Management

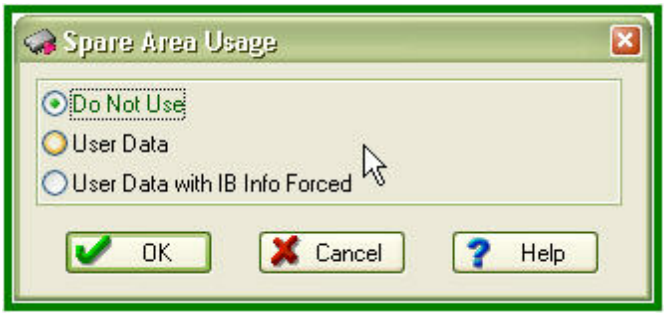
Here you can specify the algorithm of managing invalid blocks. Clicking the **Invalid Block Management**

Select one of four options:

Do Not Use	Ignore information about invalid blocks and do not care of the invalid block management. Writing into invalid blocks is enabled.
	Skip invalid blocks
Skip IB with Map in 0-th Block	Skip invalid blocks, put the Invalid block map in the block #0.
RBA (Reserved Block Area)	Use the RBA algorithm

5.4.2.1.2 Spare Area Usage

Here you can specify of how to use the **Spare Area**. Clicking the **Spare Area Usage** menu line opens the pop-up dialog:

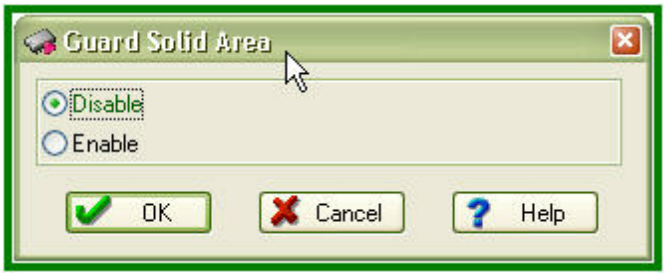


Select one of three options:

Do Not Use	This default option means: "in no case do not use Spare Area for storing user's data". Choosing this default option prevents overwriting invalid block markers in the Spare Area by the user's data. After erasing the device Flash memory markers of the invalid blocks will be restored in the Spare Area.
User Data	The Spare Area can be used for storing the user's data; the invalid block markers written into the Spare Area **will not be protected** against overwriting by user's data. If this option is chosen the programmer writes the data from its buffer into the major device memory and when this memory is completely full the programmer begins writing into the Spare Area. The programmer buffer displays merged memory pages, including the Spare Area.
User Data with IB Info Forced	The Spare Area can be used for storing the user's data but the invalid block markers written into the Spare Area against overwriting by user's data. Even if the programmer had overwritten the invalid block markers in the Spare Area it will restore these markers after completion of the programming operation.

5.4.2.1.3 Guard Solid Area

Some applications require fetching the information with strictly linear address range, e.g. the memory must be free of invalid blocks in this range. In particular, initialization of a microcontroller is possible only if the loading code is fetching from the memory device with continiously linear address space, so the source memory must not have invalid blocks. By default the ChipProgUSB disables guarding the memory area. Clicking the **Guard Solid Area** menu line opens the pop-up dialog where you can toggle the options:



When you select Enable in the dialog above you should specify this area by setting two parameters in the Solid Area setting dialog:

Start Block - the address of the first memory block that does not include invalid blocks
Number of Blocks

If in a process of the programming verification the ChipProg locates an invalid block within the specified **Solid Area** it will issue an error message and stop the current programming operation.

5.4.2.1.4 Tolerant Verify Feature

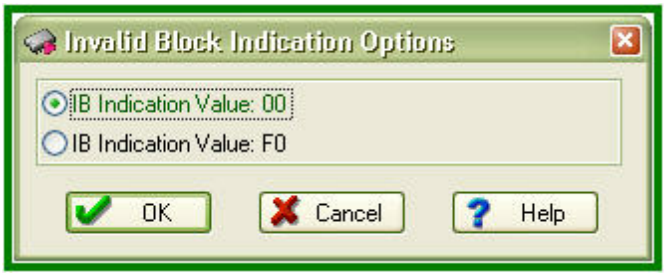
Tolerant Verify Feature menu line opens the pop-up dialog where you can toggle the options:

Usually this option is applicable in case of use the **Error Checking and Correction** (ECC) method of managing invalid blocks when you can tolerate with some errors in the data fetched from the memory device. When you select **Enable** in the dialog above**Acceptable number of errors** dialog:

ECC Frame size (Bytes) - Size of the memory array where you allow to have errors, in Bytes.
Acceptable number of errors - Acceptable number of single bit errors.

5.4.2.1.5 Invalid Block Indication Option

Here you can choose the invalid block presentation in the ChipProgUSB**Invalid Block Indication Options** menu line opens the pop-up dialog where you can select either the '00h' value (default) or the '0F0h':



5.4.2.2 Access Mode Parameters

This **Access Mode Parameters** submenu of the **Device Parameters** menu allows to invoke setting dialogs for the following parameters:

User Area
Solid Area
RBA Area
ECC Frame size
Acceptable number of errors

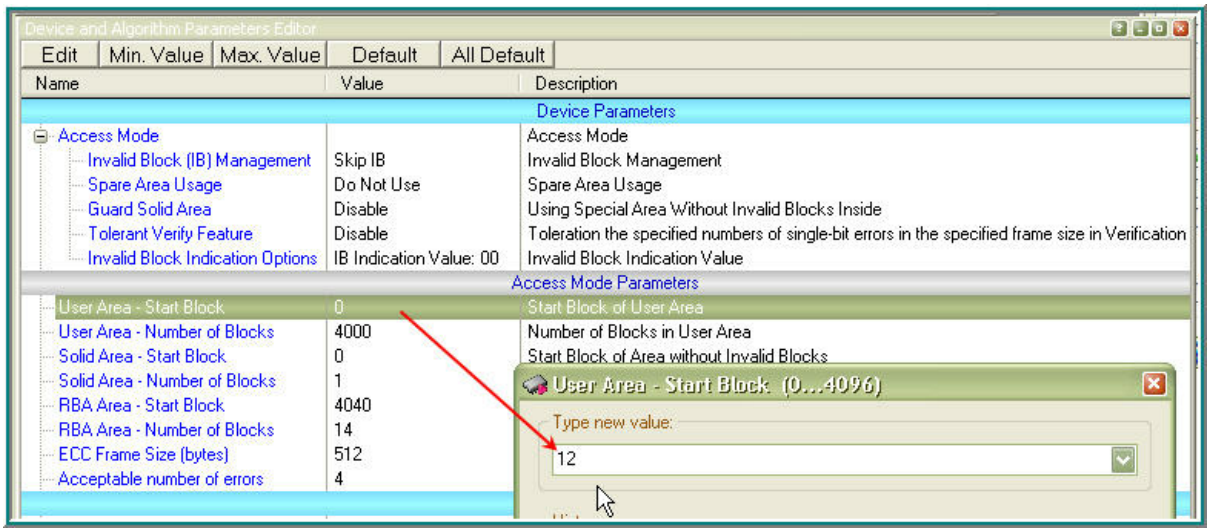
Some of the parameters above are associated with appropriate Access Modes.

5.4.2.2.1 User Area

Program, **ReadVerify** can be set applicable not to an entire NAND Flash memory device but to a specified part of the device memory - the **User Area**. The **Erase** and **Blank Check** operations are applicable only to the entire device. The **User Area's** boundaries are set by individual setting of a pair of the following parameters:

User Area - Start Block - the first memory block of the User Area.
User Area - Number of Blocks - the number of blocks in the User Area.

To set the **User Area** first click the **User Area - Start Block** submenu line. The setting dialog will pop up:



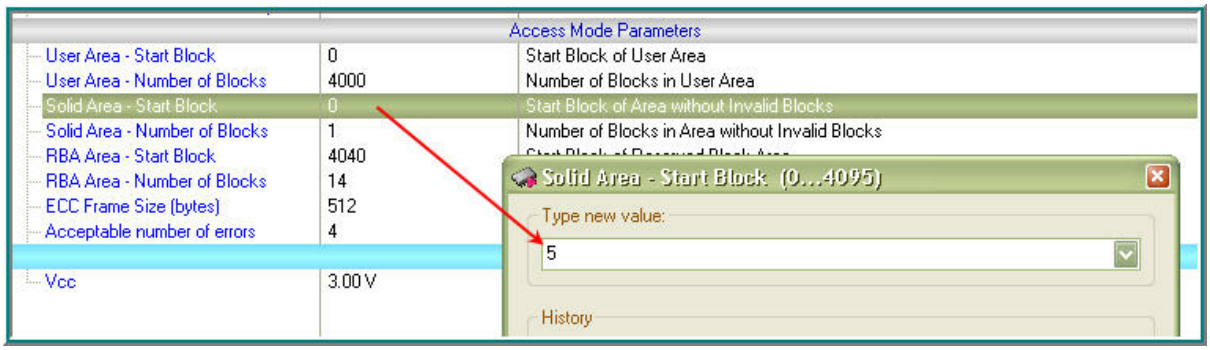
Type the value and click OK. Then click the **User Area - Number of Blocks** submenu line and enter the number of blocks into the pop-up dialog; then click OK to complete the settings.

5.4.2.2.2 Solid Area

The **Solid Area**'s boundaries are set by individual setting a pair of the following parameters:

Solid Area - Start Block - the first memory block of the memory area free of invalid blocks.
Solid Area - Number of Blocks - the number of blocks in the Solid Area.

To set the **Solid Area** first click the **Solid Area - Start Block** submenu line. The setting dialog will pop up:



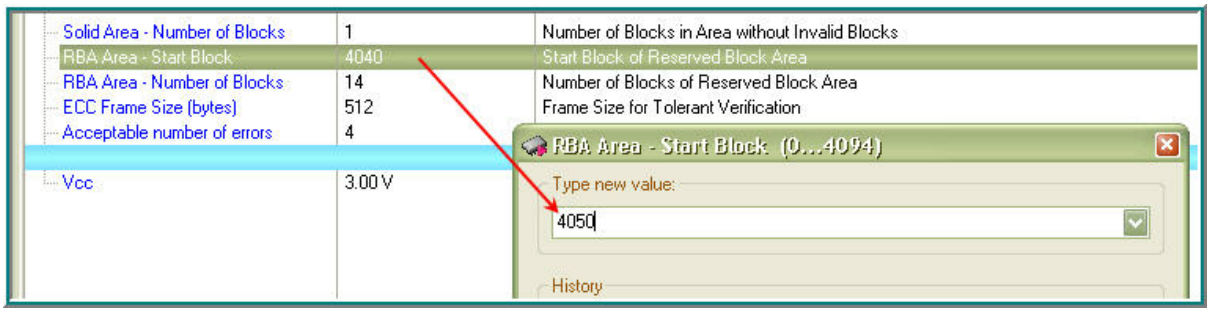
Type the value and click OK. Then click the **Solid Area - Number of Blocks** submenu line and enter the number of blocks into the pop-up dialog; then click OK to complete the **Solid Area** settings.

5.4.2.2.3 Reserved Block Area

The **Reserved Block Area** boundaries are set by individual setting a pair of the following parameters:

RBA Area - Start Block - the first memory block of the RBA.
RBA Area - Number of Blocks - the number of blocks in the RBA.

To set the RBA first click the **RBA - Start Block** submenu line. The setting dialog will pop up:



Type the offset value and click OK. Then click the **RBA Area - Number of Blocks** submenu line and enter the number of blocks into the pop-up dialog; then click OK to complete the **RBA** settings.

5.4.2.2.4 ECC Frame size

This parameter of the mode defines a size of the memory array where you allow to have errors. To set a parameter click the **ECC Frame size** submenu line. Then specify the parameter in bytes and click OK to complete the setting.

5.4.2.2.5 Acceptable number of errors

This parameter, associated with the mode, defines an acceptable number of single bit errors in the the memory array defined by the **ECC frame size**. **Acceptable number of errors** submenu line. Then enter the number and click OK to complete the setting.

5.5 Multi- and Gang-programming

This document operates with two **programming modes**:

- **Single-programming** mode means programming one device at a time by means of one ChipProg

programmer (excluding the ChipProg-G4 gang programmer).

- **Multi-programmingGang-programming** mode means concurrent programming of multiple devices at a time by:
  -- either a multiple single site programmers of one type connected in one programming cluster driven from one computer;
  -- or a special 4-site ChipProg-G4 gang programmer.

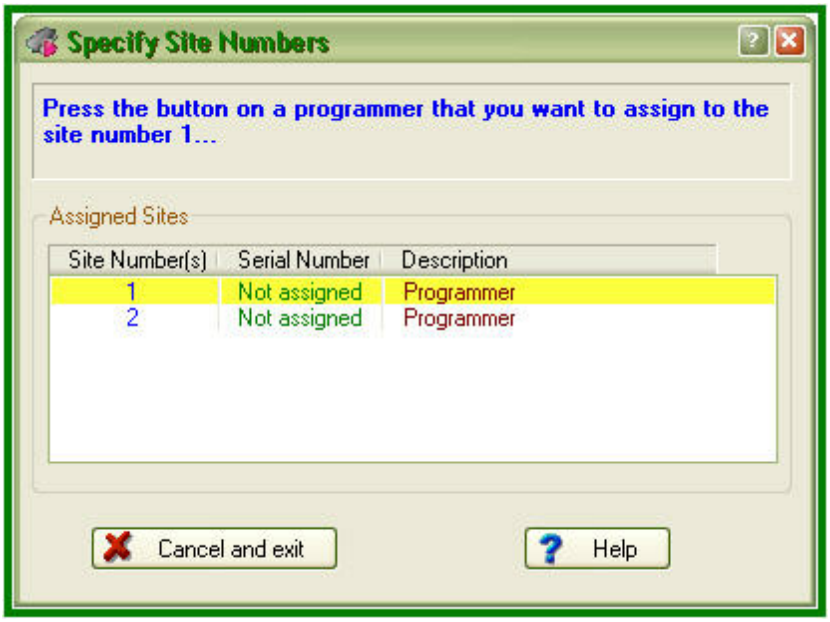
The **Multi-programming** mode differs from the **Single-programming** mode in the following items:

1. ChipProg-40 or ChipProg-48 or ChipProg-ISG programmers;
2. Only the same type of the device may be selected for every single programmer connected in one programming cluster;
3. Only the same set of buffers can be opened for every single programmer connected in one programming cluster;
4. Only the **AutoProgramming** function can be executed by the ChipProgUSB in this mode. There is however one exception - ChipProg-G4 gang programmers can be combined with ChipProg-48 tools;
5. The **Program Manager** tabs and dialogs are very different.

The **Multi-programming** mode is intended for small- and middle-volume manufacturing. The programmers in the **Multi-programming** mode work concurrently, e.g. you can start programming on one site, insert a new device into a second socket, start the programming, insert a new device into a third socket, start the programming, remove the first programmed device, etc.. An ability to linearly increase the programming system productivity by adding a new ChipProg programmer gives you flexibility and save money.

In terms of the control there is no difference whether the ChipProgUSB controls a ChipProg-G4 gang programmer or the program drives a cluster of multiple single ChipProg-40 or -48 or ChipProg-ISG programmers connected to one PC. To launch ChipProgUSB program in the **Multi-programming** mode it should be invoked either by using the **ChipProgUSB-GANG**ChipProgUSB folder or from the **command line** with the key **/GANG**.

The first dialog that appears when you started the **ChipProgUSB-GANG** shortcut (for the case when only two programmers forms a two-site programming cluster):



Now you should press the button on the programmer to which you would like to assign the site #1. Then the ChipProgUSB will prompt to assign the site #2 to another programmer (in case there are more than two programmers in the programming cluster), etc. After assigning numbers to the programmers you will get the **Program Manager** window that differs from the same window that you get when you work with one programmer.

5.5.1 The Program Manager Window

The **Program Manager** window is the major control object on the screen from which an operator controls the ChipProg . While some windows can be closed in a process of programming this one is supposed to be always open and visible. The window appearance differs from the same **Program Manager** window that you get when you work with one programmer.

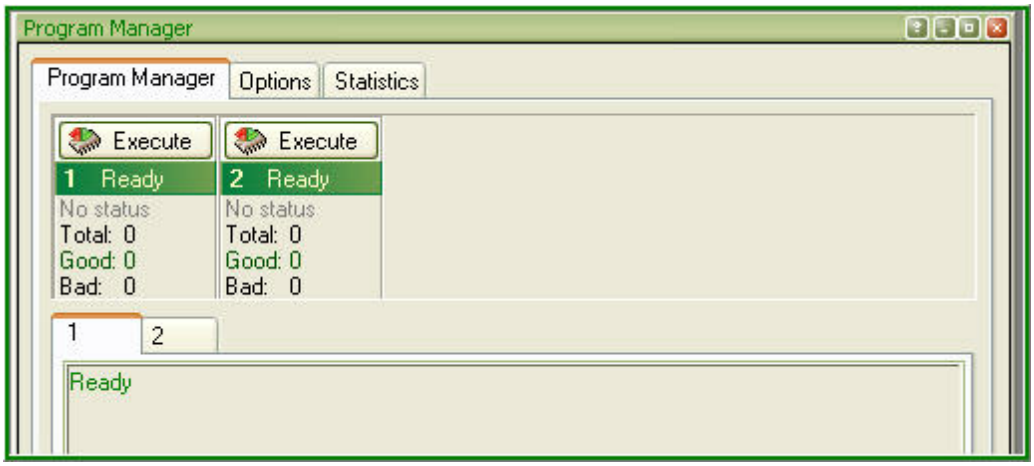
The window includes three tabs, opening three groups of settings and status indicators:

The Project Manager tab
The Options tab
The Statistics

Project Manager and **Options** tabs look differently and enable different settings for the ChipProg programmers working in the single-programming and multi-programming modes. These tabs are identical for the ChipProg-G4 gang programmer and for the -48, ChipProg-40 and ChipProg-ISP programmers when they are configured to work in the multi-programming mode. See:

5.5.1.1 The Program Manager tab

Since the only **AutoProgramming** is available in the **multi-programming** mode this tab serves for manual **AutoProgramming** initiation, displaying the site statisticsChipProgUSB program.



5.5.1.2 The Options tab

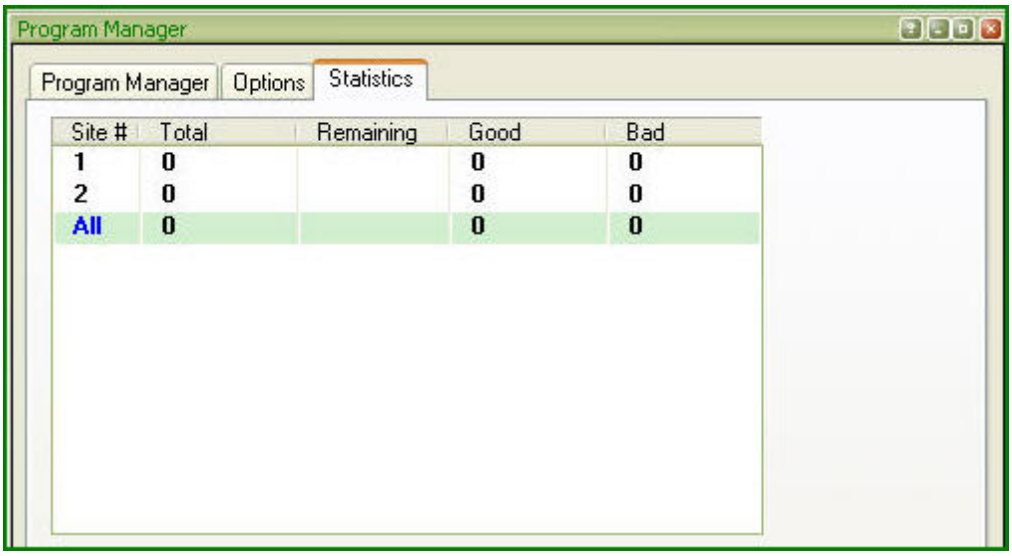
The tab serves for setting all programming parameters and options for **multi-programming** mode.

Element of dialog	Description
Buffer:	The field **Buffer** displays the active buffer to which the programming operations (functions) will be applied. A full list of open buffers is available here via the drop-down menu.
Addresses	Here you can set the addresses for the buffer and the target device to which the programming functions will be applied.
Device start:	The very first address in the target device's physical memory which will be programmed.
Device end:	The very last address in the target device's physical memory which will be programmed.
Buffer start:	The very first address in the buffer memory from which the data will be written to the target device.
Split Data	The group of radio buttons in the **Split data** field allows to program 8-bit memory devices to be used in the microprocessor systems with the 16- and 32-bit address and data buses. To do this the buffer content should be properly prepared to split one memory file into several smaller files.
Device-Auto-Detect	If this box is checked then **AutoProgramming** will start immediately after the ChipProg programmer has detected that the device is in the programming socket.
Check device ID	By default this option is always on and the ChipProg always verifies the target device identifier given by the device manufacturer. If the box is unchecked the program will skip the device ID checking.

	If this box is checked the ChipProgUSB will test whether each of the device leads is reliably squeezed by the programming socket contact. If some contact is bad a current operation will be blocked.
Reverse bytes order	If this box is checked the ChipProgUSB will sweep the byte order in the 16-bit word while it executes the **Read**, **Program Verify** operations. This option does not affect the data in the ChipProg buffers, they remain the same after the file loading.
Blank check before program	If this box is checked the ChipProgUSB will always check if the target device is blank before programming it.
Verify after program	If this box is checked the ChipProgUSB will always verify the device content right after it has been programmed.
Verify after read	If this box is checked the will always verify the device content right after it has been read out.

5.5.1.3 The Statistics tab

This tab opens the field displaying the programming session statistical results for each programming site - **Total** number of devices that were programmed during the session, what was the yield (**Good**) and how many devices have failed (**Bad**).



Element of dialog	Description
Clear statistics	This button resets the statistics.. Normally the **Total** counter increments after each Auto Programming; the, **Good** and **Bad** counters also count up. The

	ChipProgUSB reverses the counters to decrement their content (to count down).
Enable countdown	If the box is checked the ChipProgUSB
Display message when countdown value reaches zero	If the box is checked the ChipProgUSB will issue a warning when the counter **Total** is zeroed.
Reset counters when countdown value reaches zero	If the box is checked the ChipProgUSB will reset all the counters when the counter **Total** is zeroed.
Count only successfully programmed devices	If the box is checked the ChipProgUSB will count only the successfully programmed (**Good**). All other statistics will be ignored.
Set initial countdown value	Clicking on the button opens the box for entering a new **Total** number that then will be decremented after each Auto Programming .

5.6 In-System Programming

The ChipProg programmers generate all the signals necessary for programming devices installed in the user's equipment (in-system). In order to program devices in-system the programmers connect to the target via special adapters. When a device to be programmed is chosen, the ChipProgUSB software displays a part number of the appropriate **cable-adapter** in the **Device Information** window. The adapters.chm file includes wiring diagrams for all **cable-adapters**, that allows use of the adapters made by customers themselves.

General requirements for connecting ChipProg programmers to the target system

Connections

1. Connections must be done in accordance to the adapter's wiring diagram published in the adapters.chm file.
2. The target system should not shunt or overload the logical signals generated by the programmer.
3. Some IPS algorithms require generating logical signals with the voltage levels of 10 to 15V exceeding normal voltages used in electronic systems (3 to 5V). The target system should be tolerant to applying

such "high voltages".

Powering

1. **The target gets power from the ChipProg**. This is possible only if the target does not consume too much energy. The current supplied from the programmer may not exceed 80 mA, a capacity of the target power circuitry should not exceed 50 uF.
2. **The target gets power from a built-in or external power supply.** In this case the power output from the ChipProg

It is strictly prohibited to power the target from both the programmer and built-in or external power supply simultaneously.

Electrical characteristics of the ChipProg signals

Max current load for the logical signals - 5 mA.

Max current load for the Vcc line - 80 mA.

Max current load for the Vpp line - 80 mA.

NOTE! Always carefully check connecting your ChipProg programmer to the target. Wrong connecting may, and probably will cause destruction of the programmer's and/or the target system's hardware.

Most embedded microcontrollers have different algorithms for the ISP procedure. See the following topics regarding the ISP for popular microcontrollers:

Specifics of the in-system programming of the Microchip PICmicro

Specifics of the in-system programming of the Atmel AVR microcontrollers

Specifics of the in-system programming of the Atmel 8051 microcontrollers

6 Programming Automation via DLL

Any ChipProg programmer can be controlled not only by an operator from the ChipProgUSB user interface but also from an external computerized environment, mostly for the programming automation. This chapter describes how to integrate a ChipProg programmer into an external environment by means of the Phyton's proprietary **Application Control Interface** (hereafter ACI). Remember that the Application Control Interface use requires the ChipProg to be driven from a PC under Windows XP, Vista or 7.

6.1 Application Control Interface

What is the Application Control Interface?

The **Application Control Interface** (hereafter **ACI**) is a set of proprietary Phyton software allowing integration the ChipProg programmers into an external computerized environment. The ChipProgUSB software includes three Application Control Interface components:

ACI.DLL file that specifies a set of ACI functions, which can be invoked from external applications to perform programming operations. This DLL is completely conformable to the Microsoft's dynamically-linked shared library concept.

2) The **aciprog.h** header file written in the C/C++ language that lists all the ACI functions exported to the ACI.DLL.DLL and the structures associated with these functions.

3) A few program examples that control programmers from external applications

Requirements and Restrictions

1) The ChipProgUSB software must be installed on the computer that controls the operations (hereafter the instrumental or host computer). The latest ChipProgUSB software version is available for free download from the http://www.phyton.com/htdocs/support/update.shtml webpage.

2) The **ACI.DLL.DLL** requires an operational system Windows 98/ME/2000/XP/Vista and newer.

3) It is necessary to position the **windows.h** file ***before*** the **aciprog.h** file in the application program.

How does the Application Control Interface works?

The ACI.DLL launches the programmer executable file by means of the **ACI_Launch()** function and then controls the ChipProgUSB software by calling other ACI functions. The ChipProg executable, universal for all USB-hosted programmers, is the **UProgNT2.exe**.

Each ACI function, being called by an external application, sends back to this application a unique function return code. The return code constants - **ACI_ERR_xxx** - are defined into the aciprog.h

An external application can call either an ACI function without any parameter (just by the function name) or by the function name with adding a pointer to the structure of parameters. The very first parameter of any structure is always the 'UNIT size' parameter that defines the structure size. This insures compatibility of different ACI.DLL versions. The only exemption is the function **ACI_IDECommand()** - here we sacrificed uniformity of the structure format in behalf of the pseudo-function declaration simplicity.

Names of all the ACI objects (functions and structures) always begin from the prefix **ACI**. Names of the structure patterns complete with the suffix _**Params**.

Numbers of the memory buffers and layers in buffers begin from zero. All addresses have a 64-bit format and are presented by two 32-bit halves of this address (low and high) to be the compiler-independent. For example, if the compiler recognizes the **uint64** type of data then the function, which assigns a 64-bit memory buffer address in the structure ACI_Memory_Params, the function call can be presented as:

```
ACI_Memory_Params params;
*((uint64 *)params.AddressLow) = 0x123456789ABC;
```

In most cases, in a process of the programming being under control of the external application it is not necessary to make visible the ChipProgUSBChipProgUSB GUI. However, it may be necessary to unhide the programmer GUI, or just some windows and dialogs, for setting up the programming environment and for the debugging purposes (for example, for selecting the target device, loading the file, etc.). Then the ChipProgUSB user interface can be hidden to free more display space for the controlling application.

6.2 ACI Functions

In order to set up and control a ChipProg tool the program running on the instrumental computer calls the Application Control InterfaceApplication Control Interface functions requires structures that specify memory locations, pointers and other objects affiliated with the called function while other functions do not require any structures. Here is the list of the ChipProg Application Control Interface functions:

Application Control Interface function name	Brief description	Associated windows and dialogs	Associated Application Control Interface structures
ACI functions that start and stop programming sessions			
ACI_Launch	Starts the ChipProgUSB program. This function must be always the very first in the chain of other Application Control Interface functions that form the programming session.	NA	ACI_Launch_Params
	Closes the ChipProgUSBApplication Control Interface functions. It completes the external control session.	NA	NA
ACI functions that configure the programmer or get its current configuration			
ACI_LoadConfigFile	Loads the programmer configuration parameters from the host computer to the programmer.	NA	ACI_Config_Params
ACI_SaveConfigFile	Saves the programmer's current configuration parameters to the host computer.	NA	ACI_Config_Params
ACI functions that get the target device properties or set them			
	Gets the manufacturer's name (brand) and the part number of the device being currently programmed from the programmer to the host computer.	Select Device	ACI_Device_Params
ACI_SetDevice		Select Device	
ACI functions that get current parameters of the buffers and layers or configure them			
ACI_GetLayer	Gets the parameters of a specified memory buffer and layer from the programmer to the host computer.	Buffer Dump	ACI_Layer_Params
ACI_CreateBuffer	Creates a memory buffer with specified parameters in the programmer.	Buffer Dump	ACI_Buffer_Params
ACI_ReallocBuffer	Changes a size of the layer #0 in a specified memory buffer in the programmer.	Buffer Dump	ACI_Buffer_Params
ACI functions that read the buffer layer or write into it			
ACI_ReadLayer	Reads data from a specified memory buffer from the programmer to the host computer.	Buffer Dump	ACI_Memory_Params
ACI_WriteLayer	Writes data into a specified memory buffer from the host computer to the programmer memory buffer.	Buffer Dump	ACI_Memory_Params
ACI_FillLayer	Fills a whole selected layer of a specified memory buffer with a specified data pattern.	Buffer Dump	ACI_Memory_Params
ACI functions that read the content of the buffer layer or write into it			
ACI_GetProgrammingParams	Gets current programming parameters from the programmer to the host computer.	Program Manager > Options	ACI_Programming_Params
ACI_SetProgrammingParams	Sets programming parameters from the host	Program	ACI_Programming_Params

Application Control Interface function name	Brief description	Associated windows and dialogs	Associated Application Control Interface structures
	computer to the programmer.	Manager > Options	
ACI functions that get device-specific programming parameters or set them			
ACI_GetProgOption	Gets current programming options from the programmer to the host computer.	Device and Algorithm Parameters	ACI_ProgOption_Params
ACI_SetProgOption	Sets programming options from the host computer to the programmer.	Device and Algorithm Parameters	ACI_ProgOption_Params
ACI_AllProgOptionsDefault	Sets default programming options and programming algorithms in the programmer.	Device and Algorithm Parameters	ACI_ProgOption_Params
ACI functions that control programming operations			
	Initiates a specified programming operation keeping under control its successful completion or failure. It controls a single programmer.		ACI_Function_Params
ACI_StartFunction	It controls a single programmer.	Program Manager	ACI_Function_Params
ACI_GangStart	Used to control multiple device programmers. Initiates auto programming in the gang (multi-programming	Program Manager	ACI_GangStart_Params
ACI_GetStatus	Gets a current programmer status information.	Program Manager	ACI_PStatus_Params
ACI_TerminateFunction	Terminates a current programming operation.	Program Manager	NA
ACI functions that save and load files to the programmer			
ACI_FileSave	Saves a specified file from a specified buffer's layer of the programmer into the instrumental computer.	Buffer Dump	ACI_File_Params
ACI_FileLoad	Loads a specified file from the instrumental computer to a specified buffer's layer in the programmer.	Buffer Dump	ACI_File_Params
ACI functions that display programmer's windows and dialogs for setting up and debugging external programming sessions			
ACI_SettingsDialog	Displays the programmer Preferences dialog.	Configure > Preferences	NA
ACI_SelectDeviceDialog	Displays the Select Device dialog.	Select Device	NA
ACI_BuffersDialog	Displays the memory buffers setting dialog.	Buffer Dump	NA
ACI_LoadFileDialog	Displays the file loading dialog.	Buffer Dump	NA
ACI_SaveFileDialog	Displays the file saving dialog.	Buffer Dump	NA

6.2.1 ACI_Launch

ACI_FUNC ACI_Launch(ACI_Launch_Params * params);

Description

This function launches the ChipProgUSB software. Optionally this ACI function can launch the programmer with a specified command line key and load the file that will configure the ChipProg hardware.

This ACI function must always be called before any other ACI function !

6.2.2 ACI_Exit

ACI_FUNC ACI_Exit();

Description

Call of this function stops the ChipProgUSB software. In most cases the programmer practically immediately stops running. Sometimes, after calling the ACI_Exit function, it continues working for a while to correctly complete an earlier launched process. After all the ChipProg will stop and quite itself after finding that the controlling process has ended.

It is possible, however, that the software will keep running even after the control process has completely stopped. This is abnormal situation and, as the result, it will be impossible to re-establish communication with the programmer hardware by launching the ACI_Launch function. In this case you should manually close the ChipProgUSB program via the Window Task Manager.

.

6.2.3 ACI_LoadConfigFile

ACI_FUNC ACI_LoadConfigFile(ACI_Config_Params * params);

Description

This function loads the ChipProg configuration parameters that include all the settings available via the ChipProgUSB dialogs (memory buffer configurations, programming options, test of the device insertion, etc.).

The ChipProgUSB program automatically saves some programming options and settings including a type of the selected device, the device parameters, the start and end addresses of the device being programmed, the buffer start address, and a set of the AutoProgramming commands and then automatically restores these parameters when the user changes the device type.

See also: ACI_SetProgrammingParams, , ACI_GetProgrammingParams, ACI_GetProgOption, ACI_SaveConfigFile

6.2.4 ACI_SaveConfigFile

ACI_FUNC ACI_SaveConfigFile(ACI_Config_Params * params);

Description

This function saves the ChipProg options specified in the tab Option of the Program Manager window (memory buffer configurations, programming options, test of the device insertion, etc.).

The ChipProgUSB program automatically saves some programming options and settings including a type of the selected device, the device parameters, the start and end addresses of the device being programmed, the buffer start address, and a set of the AutoProgramming commands and then automatically restores these parameters when the user changes the device type.

См. также: ACI_SetProgrammingParamsACI_SetProgOption, ACI_GetProgrammingParams, ACI_GetProgOption, ACI_LoadConfigFile

6.2.5 ACI_SetDevice

ACI_FUNC ACI_SetDevice(ACI_Device_Params * params);

Description

This function chooses the device to be programmed. Along with the device type the function automatically loads the device parameters, start and end addresses and the buffer start address. Plus it restores the **AutoProgramming** command list if the selected device type has been ever selected earlier but the parameters listed above were changed in a process of the programming session.

6.2.6 ACI_GetDevice

ACI_FUNC ACI_GetDevice(ACI_Device_Params * params);

Description

6.2.7 ACI_GetLayer

ACI_FUNC ACI_GetLayer(ACI_Layer_Params * params);

Description

This function gets the parameters of a specified memory buffer and buffer's layer.

See also the ACI_Layer_Params structure description.

6.2.8 ACI_CreateBuffer

ACI_FUNC ACI_CreateBuffer(ACI_Buffer_Params * params);

Description

This function creates a buffer with the parameters specified by the ACI_Buffer_Params structure. The ChipProgUSB program automatically assigns the buffer #0 so it is not necessary to create this buffer by a separate command.

See also the ACI_Buffer_Params structure description.

6.2.9 ACI_ReallocBuffer

ACI_FUNC ACI_ReallocBuffer(ACI_Buffer_Params

Description

ACI_Buffer_Params structure.

See also the ACI_Buffer_Params structure description.

6.2.10 ACI_ReadLayer

ACI_FUNC ACI_ReadLayer(ACI_Memory_Params * params);

Description

This function reads data from a specified memory buffer. The data size is limited by 16M Bytes.

Note! This function reads the data from the programmer's memory buffer but **does not physically read out the content of the selected target device**. In order to physically read out the device memory content execute the programmer command (function) **Read** by means of the ACI_ExecFunction or ACI_StartFunction

6.2.11 ACI_WriteLayer

ACI_FUNC ACI_WriteLayer(ACI_Memory_Params * params);

Description

This function writes data to a specified memory buffer. The data size is limited by 16M Bytes.

Note! This function writes the data to the programmer's memory buffer but **does not physically program the device**. In order to physically write data from the buffer to the device's memory execute the programmer command (function) **Program** by means of the ACI_ExecFunction or ACI_StartFunction with appropriate attributes.

6.2.12 ACI_FillLayer

ACI_FUNC ACI_FillLayer(ACI_Memory_Params * params);

Description

This function fills a whole active layer of a specified memory buffer with a specified data pattern. This function works much faster than the ACI_WriteLayer function which writes data to the buffer layer.

Note! This function fills the programmer's memory buffer with a specified data pattern but **does not physically write them to the device being programmed**. In order to physically write data from the buffer to the device execute the programmer command (function) **Program** ACI_ExecFunction or ACI_StartFunction with appropriate attributes.

6.2.13 ACI_GetProgrammingParams

ACI_FUNC ACI_GetProgrammingParams(ACI_Programming_Params * params);

Description

This function gets current programming parameters specified in the tab Option of the Program Manager window (memory buffer configurations, programming options, test of the device insertion, etc.).

See the ACI_Programming_Params structure description.

6.2.14 ACI_SetProgrammingParams

ACI_FUNC ACI_SetProgrammingParams(ACI_Programming_Params * params);

Description

This function sets programming parameters specified in the tab Option of the Program Manager window (memory buffer configurations, programming options, test of the device insertion, etc.).

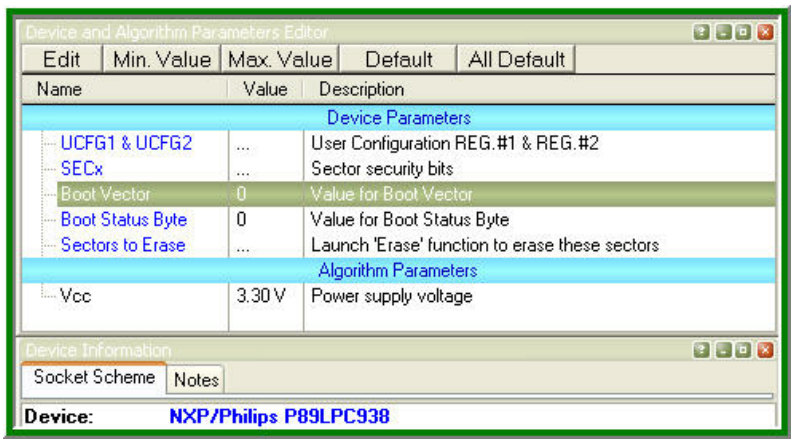
See also the ACI_Programming_Params structure description.

6.2.15 ACI_GetProgOption

ACI_FUNC ACI_GetProgOption(ACI_ProgOption_Params * params);

Description

This function gets current settings from the **Device and Algorithm Parameters Editor** window. As an example see this window for one of the microcontrollers below.



Note! This function **does not physically read the specified information from the device being programmed**. It reads from some virtual memory locations in the host PC's RAM, associated with physical locations in the target device's memory and registers. If the option, which you would like to check, is a property of the device's memory or registers then first you have to execute the programmer command (function) **Read** in the command group **Device Parameters** by means of the ACI_ExecFunctionACI_StartFunction with appropriate attributes. Then you can read the execute the ACI_GetProgOption function.

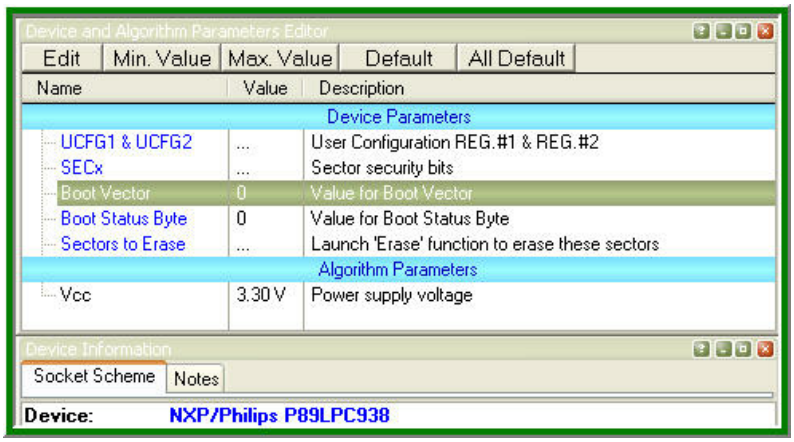
See also the ACI_ProgOption_Params structure description.

6.2.16 ACI_SetProgOption

ACI_FUNC ACI_SetProgOption(ACI_ProgOption_Params * params);

Description

This function sets device-specific options and parameters, which are specified in the **Device and Algorithm Parameters Editor**



This function **does not physically write the specified information into the device being programmed**. It actually writes to some virtual memory locations in the host PC's RAM, associated with physical locations in the target device's memory and registers. In order to complete programming the device parameters and to physically program them into the device's memory you should execute an appropriate **Program** command (function) in the command group **Device Parameters** by means of the ACI_ExecFunction or ACI_StartFunction with appropriate attributes.

See also the ACI_ProgOption_Params structure description.

6.2.17 ACI_AllProgOptionsDefault

ACI_FUNC ACI_AllProgOptionsDefault();

Description

This function sets default device-specific options and parameters specified in the **Device and Algorithm Parameters Editor** window. These default parameter sets vary. They are defined by the device manufacturers in the device data sheets.

Note! This function **does not physically restore the default settings into the device being programmed**. It actually writes to some virtual memory locations in the host PC's RAM, associated with physical locations in the target device's memory and registers. In order to complete programming the device parameters and to physically fix them in the device's memory you should execute an appropriate **Program** command (function) in the command group **Device Parameters** by means of the ACI_ExecFunctionACI_StartFunction with appropriate attributes.

6.2.18 ACI_ExecFunction

ACI_FUNC ACI_ExecFunction(* params);

Description

This function launches one of the major programming operation (**Read**, **Erase**, **Verify**, etc.) specified by the ACI_Function_Params. Being executed the **ACI_ExecFunction** does not allow to call any other ACI function until the programming operation, initiated by the **ACI_ExecFunction** function, completes the job. This differs the from the ACI_StartFunction that returns the control immediately after it was called.

6.2.19 ACI_StartFunction

ACI_Function_Params * params);

Description

This function launches one of the major programming operation (**Read**, **Erase**, **Verify**, etc.) specified by the ACI_Function_Params and immediately returns control to the external application no matter whether the programming operation, initiated by the **ACI_StartFunction**, has completed or not. This differs the **ACI_StartFunctionACI_ExecFunction**. It is possible to check if the operation has completed by the **ACI_GetStatus** function call. This allows to monitor executing programming operations if they last for a quite long time.

6.2.20 ACI_GangStart

ACI_FUNC ACI_GangStart(ACI_GangStart_Params * params);

Description

This function is used to control multiple device programmersChipProgUSB program were launched from the command line with the key **/gang** to drive a ChipProg gang programmer or a cluster of multiple programmers connected to one PC! See also the ACI_Launch function. For controlling a single ChipProg device programmer use the ACI_StartFunction or ACI_ExecFunction function.

The **ACI_GangStart** function launches Auto Programming on multiple ChipProg device programmers for the programming socket specified in the parameter**SiteNumber** of the ACI_PStatus_Params structure. The function returns control immediately. In order to detect a moment of ending the **ACI_GangStart** execution use the ACI_GetStatus function.

6.2.21 ACI_GetStatus

ACI_PStatus_Params * params);

Description

This function gets the programmer status that includes:

1) The status of the programming operation initiated by the ACI_StartFunction call (whether it has completed or it is still in progress);
2) The device insertion status (certainly if this option is enabled in the tab Option of the Program Manager window).

6.2.22 ACI_TerminateFunction

ACI_FUNC ACI_TerminateFunction();

Description

This function terminates a current programming operation initiated by the ACI_StartFunction call.

6.2.23 ACI_FileLoad

ACI_FUNC ACI_FileLoad(ACI_File_Params * params);

Description

ChipProgUSB software takes care of this.

6.2.24 ACI_FileSave

ACI_FUNC ACI_FileSave(ACI_File_Params * params);

Description

This function saves a specified file from a specified buffer's layer. The ChipProgUSB software enables saving files in all popular formats: HEX, Binary, etc..

6.2.25 ACI_SettingsDialog

Description

This macros opens the **Configure > Preferences**ChipProgUSB main window status - the main window can remain closed but the **Configure > Preferences** setting dialog will appear on the computer screen allowing manipulations in the dialog.

6.2.26 ACI_SelectDeviceDialog

ACI_SelectDeviceDialog();

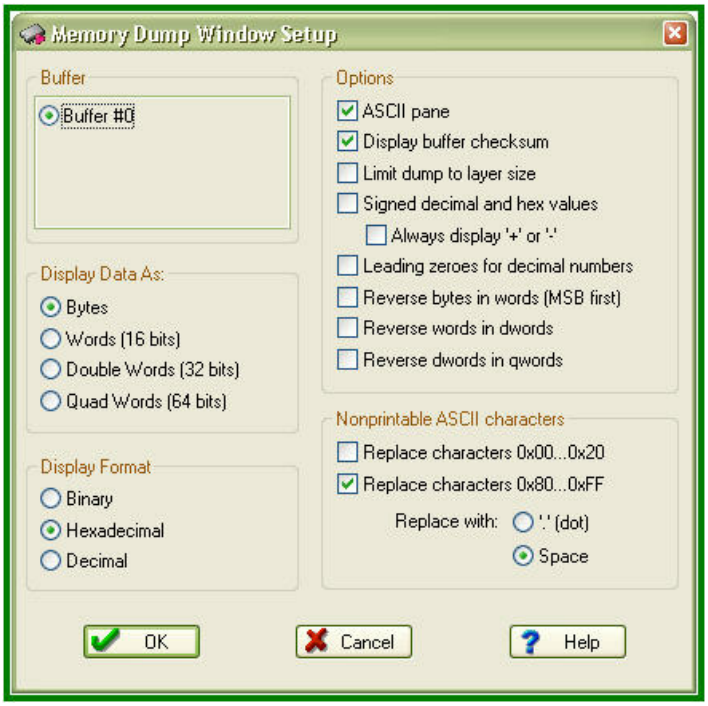
Description

This macros sends a command that opens the **Select Device** dialog. The dialog will appear on the screen regardless of the ChipProgUSB main window status - the main window can remain closed but the **Select Device** dialog will appear on the computer screen.

6.2.27 ACI_BuffersDialog

ACI_BuffersDialog();

This macros opens the **Memory Dump Window Setup** dialog. The dialog will be visible regardless of the ChipProgUSB main window status - the main window can remain closed but the **Memory Dump Window Setup** dialog will appear on the computer screen to allow the buffer setup. See the dialog example below.

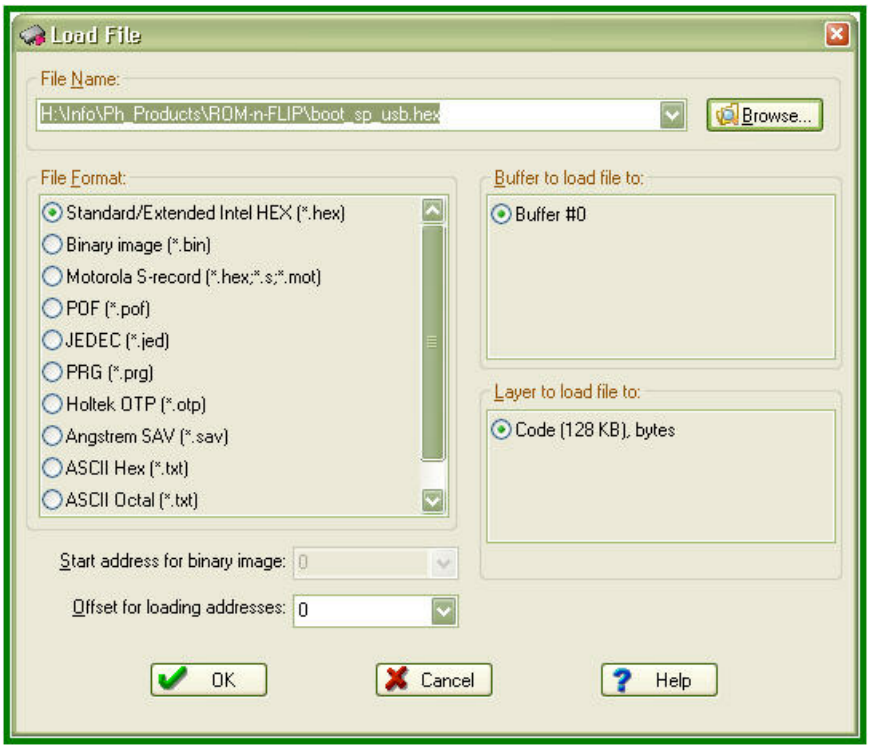


6.2.28 ACI_LoadFileDialog

ACI_LoadFileDialog();

Description

This macros opens the **Load File** dialog. The dialog will be visible regardless of the ChipProgUSB main window status - the main window can remain closed but the **Load File**

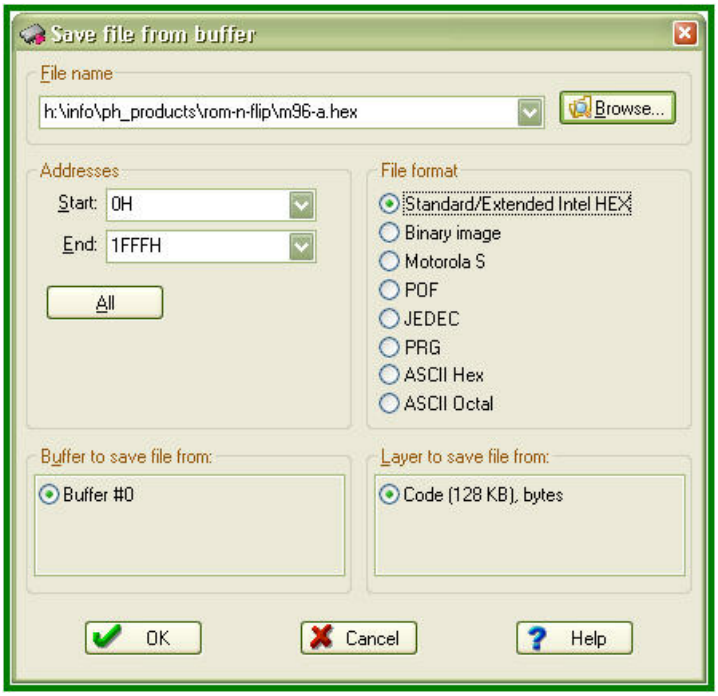


6.2.29 ACI_SaveFileDialog

ACI_SaveFileDialog();

Description

This macro sends a command that opens the **Save File** dialog. The dialog will be visible regardless of the ChipProgUSB main window status - the main window can remain closed but the **Save File** dialog will appear on the computer screen. See the dialog example below.



6.3 ACI Structures

This chapter describes the structures used by the

Structure	The ACI function that uses the structure
ACI_Launch_Params	ACI_Launch
ACI_Config_Params	ACI_LoadConfigFile, ACI_SaveConfigFile
ACI_Device_Params	ACI_GetDevice,ACI_SetDevice,
ACI_Layer_Params	ACI_GetLayer
ACI_Buffer_Params	ACI_CreateBuffer, ACI_ReallocBuffer
ACI_Memory_Params	ACI_ReadLayer, ACI_WriteLayer, ACI_FillLayer
ACI_Programming_Params	ACI_SetProgrammingParams, ACI_GetProgrammingParams
ACI_ProgOption_Params	ACI_GetProgOption, ACI_SetProgOption
ACI_Function_Params	ACI_ExecFunction, ACI_StartFunction
ACI_PStatus_Params	
	ACI_FileLoad, ACI_FileSave
ACI_GangStart_Params	ACI_GangStart, ACI_GetStatus

Here is an example of the structure syntax:

```
typedef struct tagACI_Buffer_Params
{
  UINT   Size;             // (in)  Size of structure, in bytes
  DWORD  Layer0SizeLow;    // (in || out) Low 32 bits of layer 0 size, in bytes
  DWORD  Layer0SizeHigh;   // (in || out) High 32 bits of layer 0 size, in bytes
                           //       Layer size is rounded up to a nearest value
supported by programmer.
  LPCSTR BufferName;       // (in)  Buffer name

                           // For ACI_ReallocBuffer(): in: Buffer number to realloc
  UINT   NumBuffers;       // (out) Total number of currently allocated buffers
  UINT   NumLayers;        // (out) Total number of layers in a buffer
} ACI_Buffer_Params;
```

Each structure includes a number of parameters (here Size, Layer0SizeLow, NumBuffers, etc.). The parameter's name follows its format (UINT, DWORD, LPCSTR, CHAR, BOOL, etc.). The comment to the parameter begins from a bracketed symbol showing the direction of sending this parameter:

- **(in)** - the parameter is sent from the instrumental computer **to** the programmer;
- **(out)** - the parameter is sent **from** the programmer to the instrumental computer;
- - the parameter can be sent in **either directions** depending on the ACI function context.

6.3.1 ACI_Launch_Params

```
typedef struct tagACI_Launch_Params
{
  UINT   Size;             // (in)  Size of structure, in bytes
  LPCSTR ProgrammerExe;    // (in)  Programmer executable file name

  BOOL   DebugMode;        // (in)  Debug mode. Programmer window is not hidden
} ACI_Launch_Params;
```

ProgrammerExe	This is the name of the programmer executable file. If the parameter does not include a full path then the program will search for the UprogNT2.EXE file into the folder where the ACI.DLL resides. The target folder name, where the the UprogNT2.EXE file resides, is defined by the parameter "Folder" of the ""HKLM\SOFTWARE\Phyton\Phyton ChipProgUSB Programmer\x.yy.zz" key. It is supposed that multiple ChipProgUSB versions can be installed on the host computer.
CommandLine	This structure member specifies the command line options. One of the option is NULL (no keys). If the host computer drives a cluster of multiple programmers then the only way to launch a certain programmer is to specify the /N<serial number> for the **CommandLine** structure member.
DebugMode	This key controls the ChipProgUSB main window visibility. Setting TRUE for this structure member makes the ChipProgUSB main window visible. Then you can manipulate with the programmer using its user interface - open windows, set any programmer resources, execute programming operations, etc..

See also: ACI_Launch

6.3.2 ACI_Config_Params

```
typedef struct tagACI_Config_Params
{
  UINT   Size;        // (in)  Size of structure, in bytes
  LPCSTR FileName;    // (in)  Options file name to load/save configuration
} ACI_Config_Params;
```

FileName	This is the name of the file that configures the programmer.

See also: ACI_LoadConfigFile, ACI_SaveConfigFile

6.3.3 ACI_Device_Params

```
typedef struct tagACI_Device_Params
{
  UINT   Size;             // (in)        Size of structure, in bytes
  CHAR   Manufacturer[64]; // (in || out) Device Manufacturer
  CHAR   Name[64];         // (in || out) Device Name
```

Manufacturer	The manufacturer of the device being programmed
Name	The device part number as it is displayed in the programmer's device list

See also: ACI_SetDevice, ACI_GetDevice

6.3.4 ACI_Layer_Params

```
typedef struct tagACI_Layer_Params
{
  UINT   Size;                  // (in)  Size of structure, in bytes
  UINT   BufferNumber;          // (in)  Number of buffer of interest, the first
buffer number is 0
  UINT   LayerNumber;           // (in)  Number of layer of interest, the first layer
number is 0
  DWORD  LayerSizeLow;          // (out) Low 32 bits of layer size, in bytes
  DWORD  LayerSizeHigh;         // (out) High 32 bits of layer size, in bytes
  DWORD  DeviceStartAddrLow;    // (out) Low 32 bits of device start address for this
layer
  DWORD  DeviceStartAddrHigh;   // (out) High 32 bits of device start address for
this layer
  DWORD  DeviceEndAddrLow;      // (out) Low 32 bits of device end address for this
layer
```

```
  DWORD  DeviceEndAddrHigh;      // (out) High 32 bits of device end address for this
layer
  DWORD  DeviceBufStartAddrLow;  // (out) Low 32 bits of device memory start address
in buffer for this layer
  DWORD  DeviceBufStartAddrHigh; // (out) High 32 bits of device memory start address
in buffer for this layer
  UINT   UnitSize;               // (out) Size of layer unit, in bits (8, 16 or 32)
  BOOL   FixedSize;              // (out) Size of layer cannot be changed with
ACI_ReallocBuffer()
  CHAR   BufferName[64];         // (out) Buffer name
  CHAR   LayerName[64];          // (out) Layer name, cannot be changed
  UINT   NumBuffers;             // (out) Total number of currently allocated buffers
  UINT   NumLayers;              // (out) Total number of layers in a buffer
} ACI_Layer_Params;
```

BufferNumber	The ordinal number of the memory buffer, content of which is required by the ChipProg memory buffers begin from #0.
LayerNumber	The ordinal number of the layer in the memory buffer, the content of which is required by the ACI_GetLayer function. The layer numbers begins from #0.
LayerSizeLow, LayerSizeHigh	Here the function returns the range of the memory layer's addresses in bytes.
DeviceStartAddrLow, DeviceStartAddrHigh	Here the function returns the device's start address for the selected memory layer. This address is the device's property and strictly depends on the device type - usually this value is zero. Do not mix it up with the start address of a programming operation that can be shifted by a certain offset value.
DeviceEndAddrLow, DeviceEndAddrHigh	Here the function returns the device's end address for the selected memory layer. This address is the device's property and strictly depends on the device type. Do not mix it up with the end address of a programming operation editable in the setup dialog. The selected layer's address range can be defined as a difference between the end address and the start address plus 1.
DeviceBufStartAddrLow, DeviceBufStartAddrHigh	Here the function returns the start address for the selected - usually this value is zero.
UnitSize	This structure member specifies formats of the data in memory layer: 8 for the 8-bit devices, 16 - for 16-bit devices and 32 for 32-bit devices.
FixedSize	This flag, if TRUE, disables resizing the memory layer by the ACI_ReallocBuffer function. There is one restriction on use of this flag: since the layer #0 is always resizeable the **FixedSize** is always FALSE for the layer #0.
	The name of the memory buffer as it was defined in the ChipProg interface or by the ACI_CreateBuffer
LayerName	Reserved name of the memory buffer's layer that cannot be changed by the ACI.DLL user.

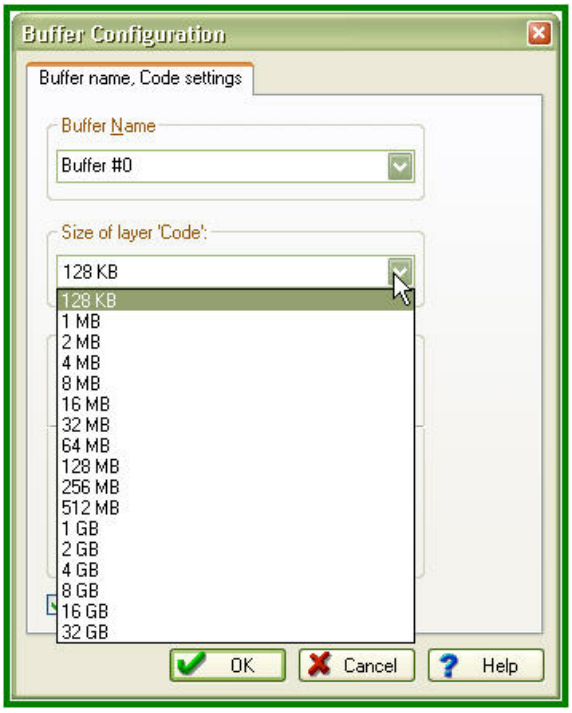
BufferNumber	The ordinal number of the memory buffer, content of which is required by the ChipProg memory buffers begin from #0.
NumBuffers	The number of the assigned memory buffers.
NumLayers	The number of layers in the programmer's memory buffers. This is a pre-defined device-specific value that is the same for all memory buffers.

See also: ACI_GetLayer

6.3.5 ACI_Buffer_Params

```
typedef struct tagACI_Buffer_Params
{
  UINT   Size;             // (in)  Size of structure, in bytes
  DWORD  Layer0SizeLow;    // (in/out) Low 32 bits of layer 0 size, in bytes
  DWORD  Layer0SizeHigh;   // (in/out) High 32 bits of layer 0 size, in bytes
                           //       Layer size is rounded up to a nearest value
supported by programmer.
  LPCSTR BufferName;       // (in)  Buffer name
  UINT   BufferNumber;     // For ACI_CreateBuffer(): out: Created buffer number
                           // For ACI_ReallocBuffer(): in: Buffer number to realloc
  UINT   NumBuffers;       // (out) Total number of currently allocated buffers
  UINT   NumLayers;        // (out) Total number of layers in a buffer
} ACI_Buffer_Params;
```

Layer0SizeLow, Layer0SizeHigh	This structure member represents the buffer layer #0's size in Bytes. This size lays in the range between 128K Bytes to 32G Bytes (may be changed in the future). The ChipProgUSB allows assigning buffers with fixed sizes only (see the list on the picture below). Any intermediate value will be automatically rounded up to one of the reserved buffer sizes. For example if you enter '160000' the programmer will assign a 1MB buffer layer.
BufferName	ACI_CreateBuffer function this structure member represents the name of buffer that will be created. This structure member used with the ACI_ReallocBuffer function will be ignored.
BufferNumber	After calling the ACI_CreateBuffer function this structure member returns the created buffer's number. After calling the ACI_ReallocBuffer function - the number of the buffer, size of which should be changed (re-allocate).
NumBuffers	This structure member represents the current number (quantity) of memory buffers being opened.
NumLayers	This structure member represents the number (quantity) of layers in memory buffers. This value is the same for all opened buffers.



See also: ACI_CreateBuffer, ACI_ReallocBuffer

6.3.6 ACI_Memory_Params

```
typedef struct tagACI_Memory_Params
{
  UINT   Size;             // (in)  Size of structure, in bytes
  UINT   BufferNumber;     // (in)  Number of buffer of interest, the first buffer
number is 0
  UINT   LayerNumber;      // (in)  Number of layer of interest, the first layer
number is 0
  DWORD  AddressLow;       // (in)  Low 32 bits of address, in layer units (natural to
device address)

  PVOID  Data;             // (in || out) Buffer to data to read to or write from
  DWORD  DataSize;         // (in)  Size of data to read or write, in layer units,
max. 16 MB (0x1000000)
  DWORD  FillValue;        // (in)  Value to fill buffer with, used by ACI_FillLayer()
only
} ACI_Memory_Params;
```

BufferNumber	The ordinal number of the buffer to read from or to write into. The buffer numerical order begins from zero.
LayerNumber	
AddressLow,	The start address in the memory layer to read from or to write into

AddressHigh	represented in the units specified by the chosen device manufacturer - Bytes, Words, Double Words. This structure member is ignored in case of use with the ACI_FillLayer function.
Data	Being used with different ACI functions this structure member has different meanings. In case of use with the ACI_ReadLayer function it represents the pointer to the data read out from the ChipProg buffer's layer. In case of use with the ACI_WriteLayer - the pointer to the data to be written to the ChipProg **Data** is ignored if it is used with the function.
DataSize	This structure member represents the data format given in memory units specified by the device manufacturer (Bytes, Words or Double Words). The program ignores the **DataSize** if it is used with the ACI_FillLayer function.
FillValue	This is the data pattern that fills an active ChipProg buffer's layer by means of the ACI_FillLayer function. If, for example, the **FillValue** is presented in the DWORD format then the 8-bit memory layers will be filled with the lower byte of the **FillValue** pattern, the 16-bit layers - with the lower 16-bit word and the 32-bit layers - with a whole **FillValue** pattern.

See also: ACI_ReadLayer, ACI_WriteLayer, ACI_FillLayer

6.3.7 ACI_Programming_Params

```
typedef struct tagACI_Programming_Params
{
  UINT   Size;                    // (in)        Size of structure, in bytes
  BOOL   InsertTest;              // (in || out) Test if device is attached
  BOOL   CheckDeviceId;           // (in || out) Check device identifier
  BOOL   ReverseBytesOrder;       // (in || out) Reverse bytes order in buffer
  BOOL   BlankCheckBeforeProgram; // (in || out) Perform blank check before
programming
  BOOL   VerifyAfterProgram;      // (in || out) Verify after programming
  BOOL   VerifyAfterRead;         // (in || out) Verify after read
  BOOL   SplitData;               // (in || out) Split data: see ACI_SP_xxx constants
  BOOL   DeviceAutoDetect;        // (in || out) Auto detect device in socket (not
all of the programmers provide this feature)
  BOOL   DialogBoxOnError;        // (in || out) On error, display dialog box
  UINT   AutoDetectAction;        // (in || out) Action to perform on device
autodetect or 'Start' button, see ACI_AD_xxx constants
  DWORD  DeviceStartAddrLow;      // (in || out) Low 32 bits of device start address
for programming operation
  DWORD  DeviceStartAddrHigh;     // (in || out) High 32 bits of device start address
for programming operation
  DWORD  DeviceEndAddrLow;        // (in || out) Low 32 bits of device end address
for programming operation
  DWORD  DeviceEndAddrHigh;       // (in || out) High 32 bits of device end address
for programming operation

  DWORD  DeviceBufStartAddrHigh;  // (in || out) High 32 bits of device memory start
address in buffer for programming operation
```

} ACI_Programming_Params;

InsertTest	This is the command to check the device insertion before starting any programming operations on the device. The procedure will check if every chip leads have good contact in the programming socket.
CheckDeviceId	This is the command to check a unique internal device identifier before the device programming.
	This is the command to reverse the byte order in 16-bit words when programming the device, reading it or verifying the data. This structure member does not effect the data value in the ChipProg memory buffers - these data remain the same as they were loaded.
BlankCheckBeforeProgram	device is blank every time before executing the command.
VerifyAfterProgram	This is the command to verify the data written into the device every time after executing the Program command.
VerifyAfterRead	This is the command to verify the data written into the device every time after executing the Read command.
SplitData	This is the command to split data in accordance with the value of the constants **ACI_SP_xxx*** in the aciprog.h file (see below). This allows 8-bit memory devices to be cascaded in multiple memory chips to be used in the systems with 16- and 32-bit address and data buses.
DeviceAutoDetect	This is the command to scan all the device's leads in a process of the device insertion into the programming socket. If the **DeviceAutoDetect** is TRUE the programmer will check whether all of the device's leads are reliably gripped by the programmer socket's sprung contacts. Only when the reliable device insertion is acknowledged, the program launches a chosen programming operation, script or a batch of single operations programmed in the AutoProgramming dialog.
DialogBoxOnError	If this structure member is TRUE then any error occurred in a process of any programming operation will generate error messages and will open associated dialogs. If this attribute is FALSE the error messages will not be issued.
AutoDetectAction	If the **DeviceAutoDetect** is TRUE then values of the **ACI_AD_xxx**** constants in the aciprog.h file define a particular action triggered either on manual pushing the **Start** button or upon auto detecting reliable insertion of the device into the programmer's socket (see below).
DeviceStartAddrLow, DeviceStartAddrHigh	This structure member defines a physical start address of the device to perform a specified programming operation (function). For example: "...read the chip content beginning the address 7Fh". Not all the functions use this parameter.

AutoDetectAction value	What to do (action)
ACI_AD_EXEC_FUNCTION	Launch the programming operation (function) currently highlighted in the .
ACI_AD_EXEC_AUTO	Launch a batch of single operations programmed in the AutoProgramming dialog.
ACI_AD_EXEC_SCRIPT	Perform the script specified in the Script File dialog.
ACI_AD_DO_NOTHING	Do not act (ignore). Then it is possible to resume operations only by executing either the ACI_ExecFunction or ACI_StartFunction.

DeviceEndAddrLow, DeviceEndAddrHigh	This parameter defines a physical end address of the device's to perform a specified programming operation (function) to. For example: "...program the chip till the address 0FFh". Not all the programmer functions use this parameter.
DeviceBufStartAddrLow, DeviceBufStartAddrHigh	This structure member defines the buffers layer's start address to perform a specified programming operation (function) from. For example: "...read the chip and move the data to the buffer beginning the address 10h". Not all the programmer functions use this parameter.

This is the bit definition from the aciprog.h header file:

*** // ACI Data Split defines**

```
#define ACI_SP_NONE             0
#define ACI_SP_EVEN_BYTE        1
#define ACI_SP_ODD_BYTE         2
#define ACI_SP_BYTE_0           3
#define ACI_SP_BYTE_1           4
#define ACI_SP_BYTE_2
#define ACI_SP_BYTE_3           6
```

**** // ACI Device Auto-Detect or 'Start' button action**

```
#define ACI_AD_EXEC_FUNCTION    0 // Execute the function currently selected in the list
#define ACI_AD_EXEC_AUTO        1 // Execute Auto Programming
#define ACI_AD_EXEC_SCRIPT      2 // Execute the script chosen in the programmer Script File
dialog
#define ACI_AD_DO_NOTHING       3 // Do nothing
```

See also: ACI_SetProgrammingParams, ACI_GetProgrammingParams

6.3.8 ACI_ProgOption_Params

```
typedef struct tagACI_ProgOption_Params
{
  UINT   Size;              // (in)  Size of structure, in bytes
  LPCSTR OptionName;        // (in)  Name of the option. For lists, it should be in
the form "List array name^List Name", e.g. "Configuration Bits^Oscillator"
  CHAR   Units[32];         // (out) Option measurement units ("kHz", "V", etc.)
  CHAR   OptionDescription[64]; // (out) Description of the option
  CHAR   ListString[64];    // (out) For ACI_PO_LIST option: Option string for Value.
ListIndex
  UINT   OptionType;        // (out) Option type: see ACI_PO_xxx constants
  BOOL   ReadOnly;          // (out) Option is read-only
  union                     // (in || out) Option value
  {
    LONG   LongValue;       // (in || out) Value for ACI_PO_LONG option
    FLOAT  FloatValue;      // (in || out) Value for ACI_PO_FLOAT option
    LPSTR  String;          // (in || out) Pointer to string for ACI_PO_STRING option
    ULONG  CheckBoxesValue; // (in || out) Value for ACI_PO_CHECKBOXES option
    UINT   StateIndex;      // (in || out) State index for ACI_PO_LIST option
    LPBYTE Bitstream;       // (in || out) Pointer to bitstream data for
ACI_PO_BITSTREAM option
```

```
  } Value;

  UINT  VSize;              // For ACI_SetProgOption():
                            //    in: Size of Bitstream if OptionType ==
ACI_PO_BITSTREAM
                            // For ACI_GetProgOption():
                            //    in: Size of buffer pointed by Bitstream if
OptionType == ACI_PO_BITSTREAM
                            //    in: Size of buffer pointed by String if OptionType
== ACI_PO_STRING
                            //   out: Size of buffer needed for storing Bitstream
data if OptionType == ACI_PO_BITSTREAM.

                            //   out: Size of buffer needed for storing String if
OptionType == ACI_PO_STRING, including the terminating NULL character.
                            //        Set Value.String to NULL to get buffer size
without setting the string
  UINT  Mode;               // (in) For ACI_SetProgOption(): SEE ACI_PP_MODE_...
constants
} ACI_ProgOption_Params;
```

	The name of the programming option - for example "Vcc". For the **ACI_PO_LIST - type** options, where the options are grouped into a list, you should specify both the list name and the option name in the following way: For example, **Configuration_bits^Generator**. There are no restrictions on use of uppercase and lowercase characters in the option names.
Units	After executing the ACI_GetProgOption function this structure member returns an abbreviation of the units, in which the programmer represents or measures the **OptionName**. For example, for the Vcc structure member, Units = "V".
OptionDescription	After executing the ACI_GetProgOption function this structure member returns the option description.
ListString	After executing the ACI_GetProgOption function for the **ACI_PO_LIST** - type options this structure member returns a string that describes the current option's value or status. For example, **XT - Standard Crystal** for the option **Configuration bits^Generator**.
OptionType	After executing the function this structure member returns the option's presentation format - for example: integer, floating point, list, bitstream, etc.. See the **ACI_PO_xxx*** constant description in the aciprog.h header file below.
ReadOnly	Setting **ReadOnly**=TRUE disables modification of the option specified by the ACI_GetProgOption function.
Value	Use of this union depends on the **ACI_PO_LONG***

Option type	
ACI_PO_LONG	The option is in the
ACI_PO_FLOAT	**Value.FloatValue**
ACI_PO_STRING	The option is represented as a string, the pointer on which is in the **Value.String**. See the note below.
ACI_PO_CHECKBOXES	The option represents a 32-bit integer word, in which you can individually toggle each bit that represents a particular

	<table><tr><td></td><td>flag in the option setting dialog. The option is in the Value. CheckBoxesValue. See, for example, the Fuse setting dialog for the ATtiny45 MCU implemented as an array of .</td></tr><tr><td>ACI_PO_LIST</td><td>It represents a list of alternative choices - only one of them can be selected at a time, so the parameter changes its value in a range 0, 1, 2 to N. The option is in the Value. CheckStateIndex. See, for example, the Oscillators setting dialog for the PIC12F509 MCU implemented as an alternatively chosen radio buttons</td></tr><tr><td>ACI_PO_BITSTREAM</td><td>Stream of bits. This option type is not in use yet but can be used for future applications.</td></tr></table>
VSize	Size of the buffer assigned for storing the string if the option type is the ACI_PO_STRING. See the note below.
Mode	**ACI_PP_xxx**** constants in the aciprog.h<) header file: <table><tr><td></td><td>**Use of the parameter Value**</td></tr><tr><td>ACI_PP_MODE_VALUE</td><td>1) For measuring (getting): use the in order to get an actual **Option** value; 2) For setting: use the Value to set a particular **Option** value.</td></tr><tr><td>ACI_PP_MODE_DEFAULT_VALUE</td><td>1) Being used with the ACI_GetProgOption function it commands to put the default Option value into the **Value** 2) Being used with the ACI_SetProgOption function the Value will be ignored; the **Option** will be set to the default level defined in the ChipProg hardware.</td></tr><tr><td>ACI_PP_MODE_MIN_VALUE</td><td>1) Being used with the ACI_GetProgOption function it commands to put the minimal **Option** value into the **Value**. 2) Being used with the ACI_SetProgOption function the will be ignored; the **Option** will be set to the minimal level defined in the ChipProg hardware, if it is possible for the **Option** of this type.</td></tr><tr><td>ACI_PP_MODE_MAX_VALUE</td><td>1) Being used with the ACI_GetProgOption function it commands to put the maximal **Option** value into the **Value**. 2) If it is used with the ACI_SetProgOption function the **Value** will be ignored; the **Option** will be set to the maximal level defined in the ChipProg hardware, if it is possible for the **Option** of this type.</td></tr></table>

This is the bit definition from the aciprog.h header file:

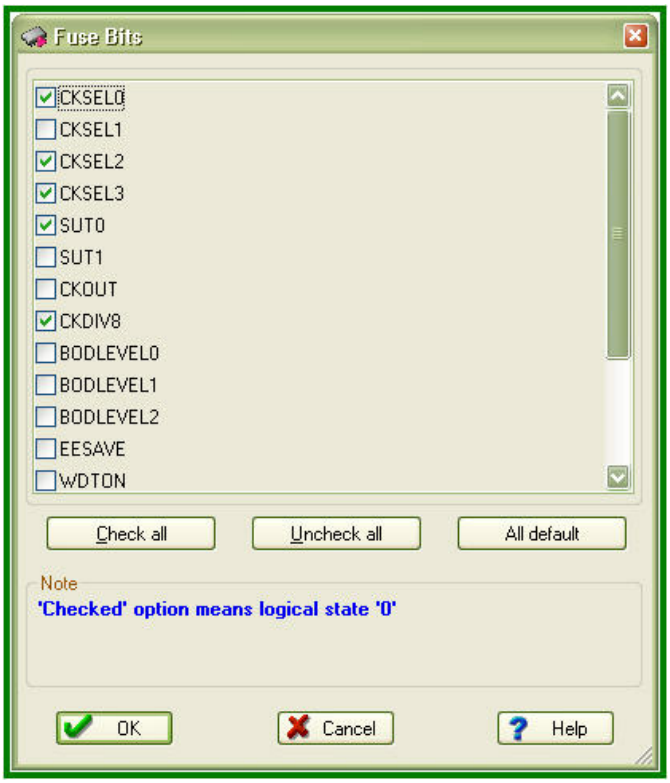
```
#define ACI_PO_LONG                0  // Signed integer option
#define ACI_PO_FLOAT               1  // Floating point option
#define ACI_PO_STRING              2  // String option
#define ACI_PO_CHECKBOXES          3  // 32-bit array of bits
#define ACI_PO_LIST                4  // List (radiobuttons)
#define ACI_PO_BITSTREAM           5  // Bit stream - variable size bit array
```

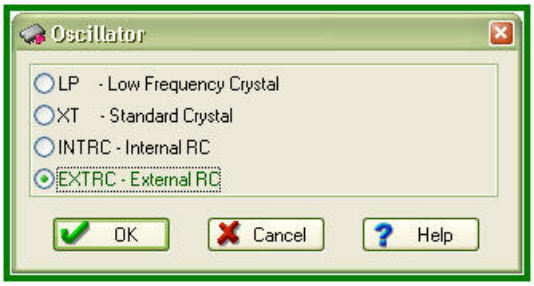
****// ACI Programming Option Mode constants for ACI_GetProgOption()/ACI_SetProgOption()**

```
#define ACI_PP_MODE_VALUE          0  // Get/set value specified in Value member of the
ACI_ProgOption_Params structure
#define ACI_PP_MODE_DEFAULT_VALUE  1  // Get/set default option value, ignore Value member
        2  // Get/set minimal option value, ignore Value member
```

#define ACI_PP_MODE_MAX_VALUE 3 // Get/set maximal option value, ignore Value member

Note for use of the ACI_GetProgOption
In order to get the buffer size necessary for storing the Option ACI_PO_STRING make the first call of the ACI_GetProgOption function with the Value.String= NULL. Then the function will return the VSize equal to the buffer size, including zero at the string's end. In your program assign the buffer of this size, put the Value.String into the buffer pointer and call the ACI_GetProgOption again.





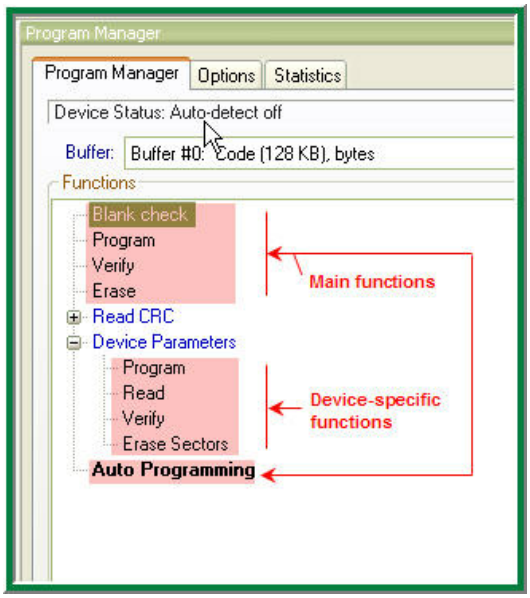
See also: ACI_GetProgOption,

6.3.9 ACI_Function_Params

```
typedef struct tagACI_Function_Params
{
  UINT   Size;                // (in)  Size of structure, in bytes
```

```
  LPCSTR FunctionName;        // (in)  Name of a function to execute. If a function is
under a sub-menu, use '^' to separate menu name from function name, e.g. "Lock
Bits^Bit 0"
                              //       To execute Auto Programming, set FunctionName
to NULL, empty string or "Auto Programming".
  UINT   BufferNumber;        // (in)  Buffer number to use
  BOOL   Silent;              // (in)  On error, do not display error message box,
just copy error string to ErrorMessage
  CHAR   ErrorMessage[512];   // (out) Error message string if ACI_ExecFunction()
fails
} ACI_Function_Params;
```

FunctionName	The name of the ChipProg function - one of those listed in the window **Functions** of the ChipProgUSB Program Manager tab. They are divided in two group (see the picture below): the main functions applicable to a majority of the target devices (**Blank Check, Erase, Read, Program, Verify**) and the device-specific lower level functions accessible through expandable sub-menus (for example, **Program Device Parameters**, **Erase Sectors**, **Lock Bits > Program Lock Bit 1, EEPROM > Read**, etc.). For such device-specific functions the FunctionName should be specified in the following way: **<List name>^<Function name>** (for example, **Device Parameters^Program**). To launch the **AutoProgramming** batch set the **FunctionName** either to NULL, a blank string, or the "Auto Programming". There is no restrictions in use of uppercase and lowercase characters in the function names.
BufferNumber	The ordinal number of the buffer the function operates with.
Silent	If this parameter is TRUE, then the error message dialog will be suppressed, the function execution will be terminated returning the **ACI_ERR_FUNCTION_FAILED** code, and the error message will be copied to the **ErrorMessage**.
ErrorMessage	The destination of the error message that will be issued if the function fails.



See also: ACI_ExecFunction, ACI_GetStatus

6.3.10 ACI_PStatus_Params

```
typedef struct tagACI_PStatus_Params
{
  UINT   Size;               // (in)  Size of structure, in bytes
  UINT   SiteNumber;         // (in) For the Gang mode: site number to get status of, otl
  BOOL   Executing;          // (out) The function started by ACI_StartFunction() is
executing
  UINT   PercentComplete;    // (out) Percentage of the function completion, valid id
Executing != FALSE
  UINT   DeviceStatus;       // (out) Device/socket status, see the ACI_DS_XXX
constants
  BOOL   NewDevice;          // (out) New device inserted, no function has been
executed yet. Valid if DeviceAutoDetect is ON.
  BOOL   FunctionFailed;     // (out) TRUE if last function failed
  CHAR   FunctionName[128];  // (out) Name of a function being executed if
Executing != FALSE. If a function is under a sub-menu, function name will be like
this: "Lock Bits^Bit 0"
  CHAR   ErrorMessage[512];  // (out) Error message string if FunctionFailed != FALSE
} ACI_PStatus_Params;
```

SiteNumber	If the ChipProgUSB was launched an the Gang mode (with the command line key / gang) and controls either the gang programmer or a cluster of single programming machines then before starting the ACI_GetStatus function the **SiteNumber** parameter must contain an ordinal number of the programming site (socket) for which the status is inquired. The site numbers begin from #0.
Executing	This parameter is TRUE while the ChipProg operation, launched by the

	ACI_StartFunction
PercentComplete	While the **Executing** == TRUE this parameter represents a percentage of the function completion - from 0 to 100.
DeviceStatus	This structure member defines insertion of the device into the programmer ZIF socket if the device insertion auto detection function is enabled. See the description of the **ACI_DS_XXX*** constants in the aciprog.h

Status	Description
ACI_DS_OK	The device is in the socket and the device's leads are reliably gripped by the programmer ZIF socket's sprung contacts.
ACI_DS_OUT_OF_SOCKET	There is no device in the programmer's ZIF socket.
ACI_DS_SHIFTED	The device's leads are reliably inserted into the socket but the device is incorrectly positioned in the socket (shifted or inserted upside down). The same status may indicate that the device type selected in the Select Device does not correspond to the type of the chip in the programmer's socket.
ACI_DS_BAD_CONTACT	The device's leads are not reliably gripped by the socket's sprung contacts. In most cases this is an intermediate situation while an operator is inserting the chip to the socket or is removing it.
ACI_DS_UNKNOWN	It is impossible to detect the status due to the device insertion auto detection feature is disabled or this feature is not supported by this programmer at all.

NewDevice	This structure member is a flag that acknowledges replacing a programmed device in the programmer's socket by a new, presumably a blank device. It works only when the device insertion auto detection function is enabled. The **NewDevice** == FALSE while the already programmed chip is still into the socket and has not been replaced by a new one. After removing the programmed device from the socket the **NewDevice** toggles to TRUE.
FunctionFailed	This is an indicator of the function execution's result. It sets to FALSE when the ACI_StartFunction launches a programming operation and remains the FALSE while the operation is in progress. If the programming operation fails and the parameter **Executing** becomes FALSE the **FunctionFailed** flag toggles to TRUE.
FunctionName	This is either the name of the programming operation (function) being currently executed or the name of the failed function, if the **FunctionFalied** == TRUE.
ErrorMessage	The destination of the error message if the function fails, i.e. the **FunctionFalied** == TRUE.

This is the bit definition from the aciprog.h header file:

```
*// ACI Device Status
                0  // Device detected, pin contacts are ok
#define ACI_DS_OUT_OF_SOCKET    1  // No device in the socket
#define ACI_DS_SHIFTED          2  // Wrong device insertion is detected (shifted or inserted
upside down)
#define ACI_DS_BAD_CONTACT      3  // Bad pin contact(s)
#define ACI_DS_UNKNOWN          4  // Unknown (Auto Detect is probably off)
```

See also: ACI_ExecFunction, ACI_StartFunction, ACI_GetStatus

6.3.11 ACI_File_Params

```
typedef struct tagACI_File_Params
{
  UINT    Size;              // (in)  Size of structure, in bytes
  LPCSTR  FileName;          // (in)  File name
  UINT    BufferNumber;      // (in)  Buffer number
  UINT    LayerNumber;       // (in)  Layer number
  UINT    Format;            // (in)  File format: see ACI_PLF_... and ACI_PSF_xxx
constants
  DWORD   StartAddressLow;   // (in)  Low 32 bits of start address for ACI_FileSave
().
                             //       For ACI_FileLoad(): Ignored if Format !=
ACI_PLF_BINARY
  DWORD   StartAddressHigh;  // (in)  High 32 bits of start address for ACI_FileSave
().
                             //       For ACI_FileLoad(): Ignored if Format !=
ACI_PLF_BINARY
  DWORD   EndAddressLow;     // (in)  ACI_FileSave(): Low 32 bits of end address
  DWORD   EndAddressHigh;    // (in)  ACI_FileSave(): High 32 bits of end address
  DWORD   OffsetLow;         // (in)  Low 32 bits of address offset for ACI_FileLoad
()
  DWORD   OffsetHigh;        // (in)  High 32 bits of address offset for ACI_FileLoad
()
} ACI_File_Params;
```

FileName	The name of the file to be loaded to the ChipProg buffer.
BufferNumber	The ordinal number of the destination buffer. Buffer numbers begins from zero.
LayerNumber	The ordinal number of the memory layer in the buffer. Layer numbers begins from zero.
Format	The loadable file's format. See the description of the **ACI_PLF_XXX*** constants in the aciprog.h header file (see below).
StartAddressLow, StartAddressHigh	1) Being used with the ACI_FileSave function this parameter specifies the first (start) address in the source memory layer, from which the file will be saved. 2) Being used with the ACI_FileLoad function, but only when it loads a file in the binary format (Format == ACI_PLF_BINARY), this parameter specifies the first (start) address of the destination memory layer, in which the file will be load into. Binary images do not carry any addresses for the file loading.
EndAddressLow, EndAddressHigh	Being used with the ACI_FileSave
OffsetLow, OffsetHigh	The address offset that shifts the file position in the destination memory layer. The offset can be negative as well as positive.

This is the bit definition from the aciprog.h header file:

```
*// ACI File formats for ACI_FileLoad()
#define ACI_PLF_INTEL_HEX          0  // Standard/Extended Intel HEX
#define ACI_PLF_BINARY             1  // Binary image
#define ACI_PLF_MOTOROLA_S         2  // Motorola S-record
#define ACI_PLF_POF                3  // POF
#define ACI_PLF_JEDEC              4  // JEDEC
#define ACI_PLF_PRG                5  // PRG
#define ACI_PLF_OTP                6  // Holtek OTP
#define ACI_PLF_SAV                7  // Angstrem SAV
#define ACI_PLF_ASCII_HEX          8  // ASCII Hex
#define ACI_PLF_ASCII_OCTAL
```

See also: ACI_FileLoad, ACI_FileSave.

6.3.12 ACI_GangStart_Params

```
typedef struct tagACI_GangStart_Params
{
  UINT   Size;                // (in)  Size of structure, in bytes
  UINT   SiteNumber;          // (in)  Site number to start auto programming at
  UINT   BufferNumber;        // (in)  Buffer number to use
  BOOL   Silent;              // (in)  On error, do not display error message box. Use
} ACI_GangStart_Params;
```

SiteNumber	The number of the device programmer socket in the gang programmer or in a programming cluster comprised of multiple ChipProg programmers for which the ACI_GangStart function is launched. The site (socket) numbers begin from #0.
BufferNumber	The ordinal number of the memory buffer, content of which is required by the ACI_GangStart function. Numbers of ChipProg memory buffers begin from #0.
Silent	If this parameter is TRUE, then the error message dialog will be suppressed, the function execution will be terminated returning the **ACI_ERR_FUNCTION_FAILED** code. Use the ACI_GetStatus

See also: ACI_GangStart, ACI_GetStatus

6.4 Examples of use

The ChipProgUSB software includes a few examples of use the **Application Control Interface** functions and structures. The examples reside in the subdirectory **ACI\Programmer ACI Examples** in the directory where the ChipProg program is installed.

The examples are written in the C language and are represent the projects that can be compiled by the

Microsoft Visual Studio® 2008. The project sources can be also compiled by other C/C++ compilers, sometimes with minor adjustments. After building the project you get the Windows consol application executable file.

In order to adjust the example project (or a part of it) for use in your application you have to set correct paths to the ACI functions called by the main() function. This includes paths to the ChipProg executable file, to the file that is supposed to be loaded to the programmer's memory buffer or to be saved from the buffer. You also have to specify your target device. See an example of the main() function's fragment below.

```
/*+          main                    °          01.07.09 17:37:24*/
......

  // Launch the programmer executable
  if (!Attach("C:\\Program Files\\ChipProgUSB\\4_72_00\\UProgrnt2.exe", "", FALSE)) return -1;

  // Select device to operate on
  if (!SetDevice("Microchip", "PIC18F242")) return -1;

  // Load .hex file to buffer 0, layer 0
  if (!LoadHexFile("C:\\Program\\test.hex", 0, 0)) return -1;

....
```

All examples uses the ACI.DLL file that must be either in the same folder where the example executable file resides or in the folder specified in the variable PATH. In the supplied examples the ACI.DLL file is already copied into the folders where the MS Visual Studio creates executable files.

Example Descriptions

Each example has a comment header briefly describing the program purposedo. Additionally, some comments are inserted in the code texts. All examples begin from executing the ACI_Launch() function that activate the programmer.

AutoProgramming.c

This is the simplest and frequently used example of the ChipProg external control. The program launches the programmer, selects the PIC18F242 target device, loads the test.hex file into the programmer buffer, sets default programming options and then executes a preset AutoProgramming batch of functions: Erase, Blank Check, Program, Verify.

LongProgramming.c

This example shows how to monitor a process of the AutoProgramming procedure if it may last quite a long time. The program acts as the the example above. The programming launches by the ACI_StartFunction then it keeps checking percentage of the operation completion by means of the ACI_GetStatus function. If the operation fails the programmer issues an error message, otherwise it allows continuing operations.

ProgrammingOptions.c

This example shows how to get, print out and change options settable in the Device and Algorithm

Parameters Editor window. First, the program checks the device insertion into the programmer's socket by calling the ACI_GetStatus(&Status) function. Then, after detecting correct and reliable insertion of the device into the programmer's socket, the program reads the current set of options by the ACI_GetProgOption function and print them out. Then it changes the Vpp value from the default to 10.5V and disables the device Brown-out Reset feature.

SaveMemory.c

This example shows how to save a binary image of the device in a file. First, the program checks the device insertion into the programmer's socket by calling the ACI_GetStatus(&Status) function. Then, after detecting correct and reliable insertion of the device into the programmer's socket, the program reads data from a specified range of the SST89V564RD device's memory and saves them in the file test. bin.

Checksum.c

This example shows how to calculate a checksum of the data read out from a device. First, the program checks the device insertion into the programmer's socket by calling the ACI_GetStatus(&Status) function. After detecting correct and reliable insertion of the device into the programmer's socket the program figures out a real size of the SST89V564RD device's flash memory by executing the ACI_ExecFunction function then it assigns the buffer 'buf' in the host computer's memory in order to accommodate the data read out from the device, moves the data to this buffer and calculates the checksum of the buffer's content.

7 Script Files

The program ChipProgUSB can execute so-called **script files** in a way similar to how DOS executes the batch files. Use of script files is to automate usage of the ChipProg**Console window** or other special windows generated by the script itself, including displaying graphical data in special windows; to create user's custom menus, etc. The script language is similar to C program language; almost all C constructions are supported, except for structures, conjunctives and pointers. There are also many built-in functions available, such as printf(), sin() and strcpy(). The extension of script source file is **. CMD**.

When the ChipProgUSB program starts, it searches for the script with the reserved name **START.CMD**. So, if you wish the ChipProgUSB program would automatically perform some operations immediately after you launch the program, you can create a special script. The ChipProgUSB program begins searching for the **START.CMD** in the current directory on the disc, then it searches for this script in the directory where the ChipProgUSB.exe file resides. If the **START.CMD** is not located then a default ChipProg shell will open.

The **scrips** controls and associated dialogs and windows are concentrated under the **Script menu**. The major dialog that controls scripts is the **.**

See also:

Simple example of a script file

How to write a script file

How to start a script file

How to debug a script file

Description of Script Language

Script Language Built-in Functions

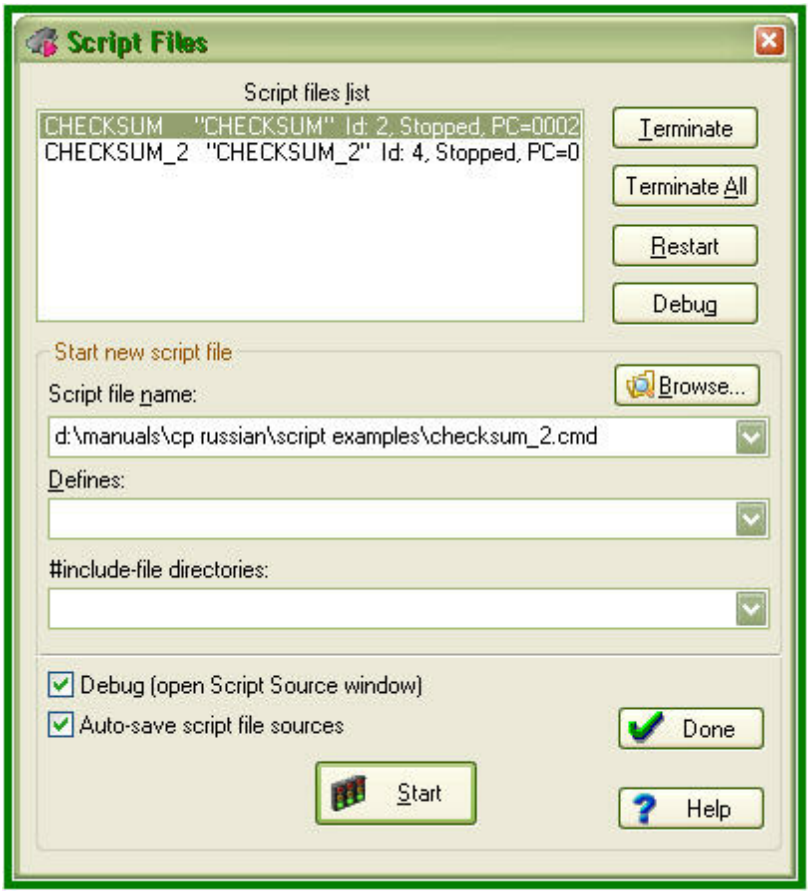
Script Language Built-in Variables

Difference Between the Script Language and the C Language

Alphabetical List of Script Language Built-in Functions and Variables

7.1 The Script Files Dialog

This dialog is used for controlling the , it allows to **start, stop** and **debug** scripts.



In the upper window of this dialog you see the list of loaded script files with the current state of each file. Any script can be in one of the following states:

State of File	Description
Stopped	Execution of the script file is temporarily stopped.
Running	The script file is being executed.
Waiting	The script is waiting for an event. This state is initiated by calling certain wait functions in the script file text (for example, **Wait**).
Cancelled	The script execution is terminated, but the script file is not yet unloaded from the memory.

To select a script file, highlight its name in the window. The four buttons on the right of the list control the highlighted script:

Button	Description
Terminate	Unloads the selected script file if it can be unloaded. Otherwise, it sets up the flag for the selected script that then goes to the **Canceled** state.
Terminate All	Unloads all script files visible in the window.
Restart	Restarts a highlighted script file.

Debug	Switches to the Debugger mode for the highlighted script file. This command stops execution of the script and opens it in the source window of the script for debugging. If the script is in the wait state, then execution will immediately stop after the script returns from the **Waiting** status.

When you use several script files simultaneously and unload or restart some of them, remember that script files can share global data and functions. If one script accesses data or the functions belonging to another one that is already unloaded, then the script interpreter will issue error messages and the active script file will be also be unloaded (terminated).

Element of dialog	Description
Script File Name	Specifies a name of the script file to be loaded. You can either typed in the file name with a full path to the box or to take it from the drop-down history list or browse it from a computer disc.
Browse	Opens the **Load/Execute Script File** dialog for locating and loading script files into the **Script File Name** box.
Defines	Defines the processor text variables for compilation. For more information, see below the **Processor text variables**.
#include-file Directories	Specifies the directories in which the script file will search for the files specified in the **#include <file_name>** directive(s). To specify more than one directory, separate them by semicolons. The current directory is scanned as well.
Debug (open Script Source window)	If this box is unchecked, a script file automatically starts execution upon the file loading. If the box is checked, then upon loading a script file, the program immediately opens the window for debugging the script. See also How to Debug a Script File. If this box is checked when you click the ChipProgUSB automatically saves the source texts of all script files visible in the **Script Source** windows.
Start	Starts the script file specified in the **Script File Name** box.

Processor text variables

The content of the text box is equivalent to the #define directive in the C language. For example, if you type **DEBUG** in this text box, the result will be as if the **#define DEBUG** directive is placed in the first line of the script source text.

You can specify values for variables. For example, **DEBUG=3** is equivalent to **#define DEBUG 3**.

You can list several variables in a line and separate them with semicolons. For example:

```
DEBUG;Passes=3;Abort=No
```

Also, see Predefined Symbols at the Script File Compilation.

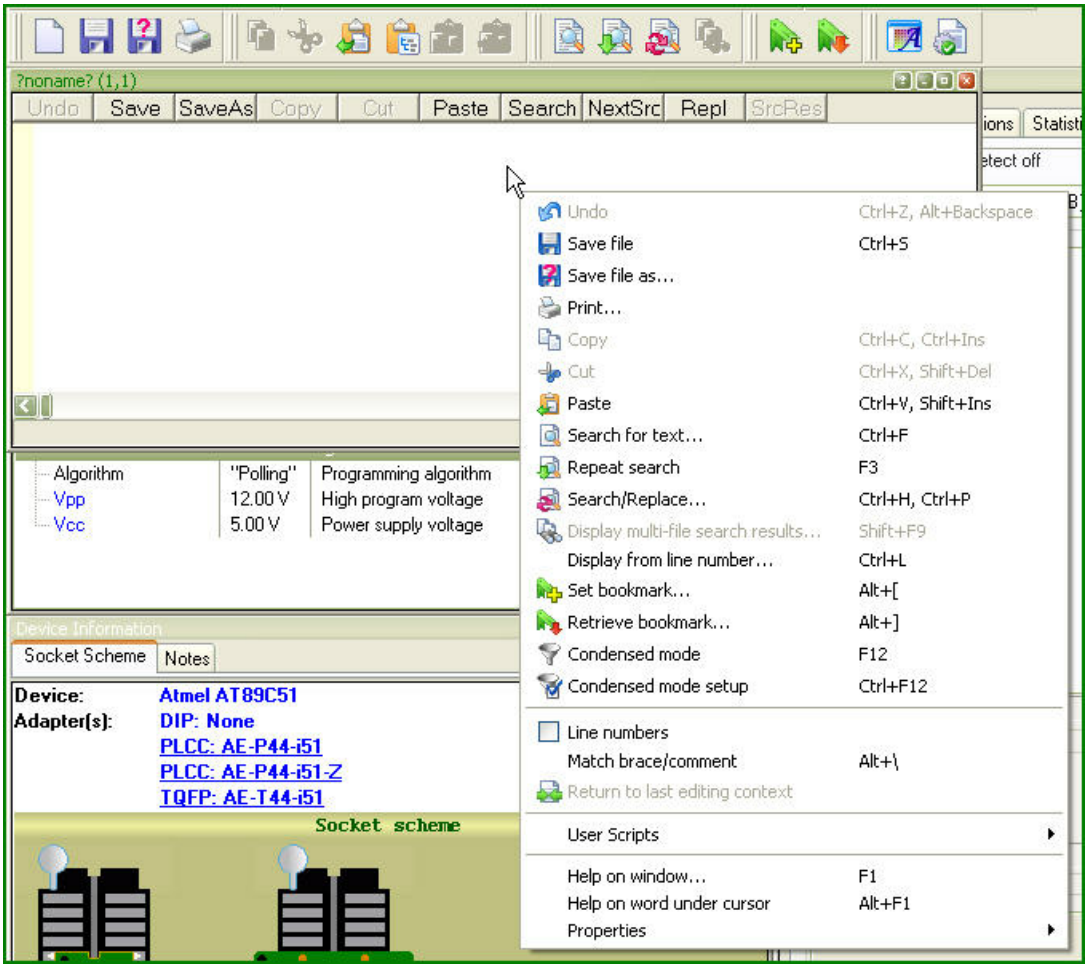
7.2 How to create and edit script files

A **script file** is similar to a source program text written in programming language (C, for example), e.g. a script file can be created and edited either in the **Editor window** by the ChipProgUSB built-in editor or by any other editor. You can allocate script files in your work directory or in the directory where the

ChipProgUSB program is installed.

Normally the Editor toolbar that collects all the edit function buttons is hidden. To create a customized editor toolbar right click on the blank area of the main toolbar, select the Customize line in the drop-down menu and check the boxes of the editor functions which you would like to make visible.

To open a new script file for editing open the **Script menu > Editor window > New**. This will open a blank window below. Right clicking within the window pops up the **Editor command menu** that includes the buttons which you can bring up to the local **Editor toolbar**. Here the toolbar is shown above the window.



Now you can compose your script right in the window.

Note that you should not use the punctuation characters (braces, dash, etc.) in the script file name.

When you complete the file composing click on the **Save** button on the window local toolbar or on the **Editor toolbar** and the program will prompt you to name the script file and to specify its location.

7.2.1 The Editor Window

Commands of this menu refer to the currently active **Edit** window.

Button	Command	Description

	**New**	Opens the **Editor window** for a new script file.
	**Open...**	Opens the **Open file** dialog to load a script file for editing. The file name and path can be either entered or browsed here.
	**Save**	Saves the file from the currently active window to a disc.
		Opens the **Save file as...** dialog.
	**Print**	Opens the standard **Print** dialog for the default printer. You can print an entire file or a selected text block.
	Properties..	The common properties for open files.

7.2.2 Text Edit

Commands of this menu refer to the currently active **Edit** window.

Button	Command	Description
	**Undo**	Undoes the last text editing action executed in this window. For example, if the last action deleted a line, then the deleted line will be restored. The number of steps provided by the Undo function is set in the of the **Configure > Editor Options > General tab**.
	**Copy**	Copies the marked block to the clipboard. The text format in the clipboard is standard and the copied block is accessible to other programs.
	**Cut**	Removes the marked block to the clipboard..
	**Paste**	Copies the block from the clipboard, starting at the cursor position.
	Clipboard History/ Repository	Opens the **Clipboard History/Repository** dialog.
	Append to Clipboard	
	Cut & Append to Clipboard	Cuts the marked block of text and appends it to the block in the clipboard.
	Fast Copy	Copies a block to a specified position in the same window.
	Fast Move	Moves a block from one position in the window to another position in this window.
	Block Off	Unmarks a marked text block.
	**Search**	Opens the **Search for Text** dialog.
	**Next Search**	Repeats search with the parameters used in the previous search.

	**Replace**	Opens the **Replace Text** dialog.
	**Display Multi-file Search Results**	Re-opens the last multi–file search results in the **Multi-File Search Results** dialog.
	Display from line number...	Opens the **Display from Line Number** dialog for you to specify a line number. Source text will be displayed from this line.
	**Set bookmark...**	Opens the **Set Bookmark** dialog
	**Retrieve bookmark**	Opens the **Retrieve Bookmark** dialog to retrieve a local bookmark.
	**Condensed mode**	Toggles the Condensed display mode on and off.
	**Condensed mode setup**	Opens the **Condensed Mode Setup**
	Line numbers on/off	Toggles the availability of the line numbers on and off.
	Return to last editing context	Activates the most recently edited **Source** window, and places the cursor in its final position during the edit.

7.2.2.1 The Search for Text Dialog

This dialog sets complex criteria and parameters for searching text in files. This dialog and the **Replace Text** dialog have a number of common parameters, which function in the same way in both dialogs. To specify file names, you can use one or several wildcards. Also, the names may contain paths. You can search in more than one file at once by using parameters of the **Multi-File Search** area.

	Description
String to Search for	Specifies the text string to search for.
Case Sensitive	This box is unchecked by default. Checking this box specifies that the case of the string is to be matched.
Whole Words Only	This box is unchecked by default. If checked, the editor will search only for whole words: the string will be found only if it is enclosed between punctuation or separation characters (spaces, tabulation symbols, commas, quotation marks, etc.).
Regular Expressions	This box is unchecked by default. Checking of this box specifies that the search string is a regular expression.
Global	Search the entire file for the string. Enabled by default.
Selected Text	Search the string in the selected block.
From Cursor	Search from the current cursor position.
Entire Scope	Search from the beginning or end of the file (depending on the search direction). Enabled by default.
Perform Multi-File Search	If this box is checked the editor will search in all project files (see the notes below). If the box is unchecked, then the search will be performed in the current window only.
Search All Source	If this box is checked the editor will search in all the source files included in

Files in Project	the project.
Include Dependency Files	If this box is checked the editor will search in all the source files included in the project and all files on which the source files depend, whether explicitly or implicitly. For C language, these are the header files (*.h).
Search Wildcard(s)	Check this box to search for one or several wildcards specifying the files to be searched. Separate wildcards with semicolons. No quotes are required to denote Windows-style long names. Example: `*.txt;*.c;c:\prog\*.h`. This option and the **Search All Source Files in Project** option act independently of each other: you can search in all files of the project AND in other files that comply with the specified wildcard(s).
Search Subdirectories	If this box is checked the editor will search in subdirectories of all the directories specified by the **Search All Source Files in Project** option and by wildcards.
Starting Path	Begin search from the directory specified in this text box. This directory serves as the common path and is useful when there are several wildcards such as the following ones: `c:\prog\text\source\*.txt;c:\prog\text\source\*.doc` In this case, make use of wildcards (`*.txt;*.doc`) and common path (`c:\prog\text\source`).

Notes

1. When you search in the file opened in the **Source**
2. Multi-file search is performed in all source files of the project. Upon finishing, the **Multi-File Search Results** dialog remains open.

7.2.2.2 The Replace Text Dialog

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

7.2.2.3 The Confirm Replace Dialog

This dialog requires your permission to replace a found string. You can turn the prompt on/off by checking/ clearing the **Prompt at Replace** box in the dialog.

Button	Function
Yes	Replace the found string.
No	Cancel this replacement. If the procedure is started with the **Change All** button for all occurrences in the search area, then the search-and-replace process will continue.
Non-Stop	From this moment, replace all found strings in this file without prompt.
Cancel	Cancel the search-and-replace process.
	Stop search in this file and switch to the next one.
Replace in All Files	Replace all occurrences in all other files without confirmation.
Move cursor to the Yes/No Buttons	When this box is checked the cursor will be automatically placed on the **Yes** button on each inquiry for confirmation.

7.2.2.4 The Multi-File Search Results Dialog

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam velit risus, placerat et, rutrum nec, condimentum at, leo. Aliquam in augue a magna semper pellentesque. Suspendisse augue. Nullam est nibh, molestie eget, tempor ut, consectetuer ac, pede. Vestibulum sodales hendrerit augue. Suspendisse id mi. Aenean leo diam, sollicitudin adipiscing, posuere quis, venenatis sed, metus. Integer et nunc. Sed viverra dolor quis justo. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis elementum. Nullam a arcu. Vivamus sagittis imperdiet odio. Nam nonummy. Phasellus ullamcorper velit vehicula lorem. Aliquam eu ligula. Maecenas rhoncus. In elementum eros at elit. Quisque leo dolor, rutrum sit amet, fringilla in, tincidunt et, nisi.

Donec ut eros faucibus lorem lobortis sodales. Nam vitae lectus id lectus tincidunt ornare. Aliquam sodales suscipit velit. Nullam leo erat, iaculis vehicula, dignissim vel, rhoncus id, velit. Nulla facilisi. Fusce tortor lorem, mollis sed, scelerisque eget, faucibus sed, dui. Quisque eu nisi. Etiam sed erat id lorem placerat feugiat. Pellentesque vitae orci at odio porta pretium. Cras quis tellus eu pede auctor iaculis. Donec suscipit venenatis mi.

Aliquam erat volutpat. Sed congue feugiat tellus. Praesent ac nunc non nisi eleifend cursus. Sed nisi massa, mattis eu, elementum ac, luctus a, lacus. Nunc luctus malesuada ipsum. Morbi aliquam, massa eget gravida fermentum, eros nisi volutpat neque, nec placerat nisi nunc non mi. Quisque tincidunt quam nec nibh sagittis eleifend. Duis malesuada dignissim ante. Aliquam erat volutpat. Proin risus lectus, pharetra vel, mollis sit amet, suscipit ac, sapien. Fusce egestas. Curabitur ut tortor id massa egestas ullamcorper. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec fermentum. Curabitur ut ligula ac ante scelerisque consectetuer. Nullam at turpis quis nisl eleifend aliquam. Sed odio sapien, semper eget, rutrum a, tempor in, nibh.

7.2.2.5 Search for Regular Expressions

The text editor supports "regular expressions," which can be used to search for special cases of text strings. Regular expressions contain the control characters in the search argument string:

?	Means any one character in this place. Example: if you specify *?ell* as the search string, then "bell," "tell," "cell," etc. will be found.
%	Means the beginning of line. The characters following '%' must begin from column 1.

Example: - find the word "Counter," which begins at the first column.

**$** The end of line. The characters preceding the '$' should be at the last positions of the line. Example: Counter$ - find the word "Counter" at the line end.

@ Match the next character literally; '@' lets you specify the control characters as usual letters. Example: @? - search for the question mark character.

\xNN\xA7 - find the character with the hexadecimal code of A7.

+ Indefinite number of repetitions of the previous character. For example, if you specify 1T+2, then the editor will find the lines containing "1" followed by "2", which are separated with any number of repetitions of the letter T.

[c1-c2] Match any character in the interval from c1 to c2. Example: [A-Z]

[~c1-c2] Match any character whose value is outside the interval from c1 to c2. Example: [~A-Z] means any character except for the uppercase letters.

text1|text2 The "|" character is the logical "OR" and the editor will look for either **text1** or **text2**. Example: LPT|COM|CON means search for "LPT" or "COM" or "CON."

7.2.2.6 The Set/Retrieve Bookmark Dialogs

Bookmarks help you to return to a marked cursor position in a source file.

You can set and retrieve up to 10 local bookmarks. Every local bookmark has an individual numbered button assigned to it.

To open the **Set Bookmark** dialog, press . To open the dialog, press **Alt+]**. To set/retrieve a bookmark, press its numbered button. The number of the bookmarked line, the bookmark position in the line (in brackets) and the text of the line are shown at the right of the button.

Local bookmarks are stored in the configuration file and you can work with them in the next session.

7.2.2.7 Condensed Mode

In the Condensed mode, only lines that satisfy a specified criterion are displayed in the window. There are two available criteria:

- the line must contain a given sub-string;
- the first non-space character in a line must be at a specified position (column).

Examples: (a) with the sub-string criterion and the sub-string set to "counter," only the lines containing the word "counter" will be displayed; (b) with the second criterion and the position set to four, only the lines in which text begins at column 4 will be displayed.

The Condensed mode brings the lines having some common feature to "one place." If you attentively follow a rule to begin the declaration of data at position 2, procedures at position 3, and interrupt handlers at position 4, then the Condensed mode will help you to find a necessary declaration. If you comment certain lines with the same or similar comments and use the Condensed mode with sub-string, you will be able to benefit from your composing style. In the Condensed mode, you can move the cursor just as in the normal mode.

How to control

The criterion for display is set in the **Main menu > Script > Text Edit >** **Condensed Mode Setup** dialog. To toggle the Condensed mode on/off, use the **Edit** menu command, the **Condensed Mode** command of the local menu or the **F12** hotkey. To exit the Condensed mode, press **Esc**. When you exit, the cursor returns to the position at which it was before the mode was turned on. To exit the mode and remain in the line from which you moved the cursor while in the mode, press **Enter** or begin editing the line.

7.2.2.8 The Condensed Mode Setup Dialog

This dialog sets up the parameters for the Condensed mode of the **Source** window.

The **Display Lines of Text** area has radio buttons for switching between two alternative criteria for condensing text in the **Source** window: **Containing String** and **Where First Non-blank Column Is:**

1. If you check the **Containing String** radio button the Source window will display only the lines with text that match the sub-string specified in the text box at the right. Additionally, you can specify that the case should be matched the case, that whole words only should be used, and that the sub-string is a regular expression.

2. If you check the **Where First Non-blank Column Is** radio button, the Source window will display the lines where text begins from the position specified in the **Column** box. Then you should select one of four options by checking an appropriate radio button:

- - the first non-space character should be exactly in the specified column. For example, if you specify position number 2, the window will display only the lines whose text begins in column 2.
- **Not Equal to** - the first non-space character should be in any column except the position specified here. For example, if you specify position number 2, the window will not display all the lines beginning in this column. All other lines will be displayed.
- **Less than** - display only the lines in which text begins at a position less than the specified one.
- **Greater than** - display only the lines in which text begins at a position greater than the specified one.

When you have completed setup click **OK** to switch the **Source** window to the Condensed mode.

7.2.2.9 Automatic Word Completion

It is normal for words (labels, names of variables) to be repeated within a limited part of a file. In such cases, the **Source** window helps you finish typing the whole word.

If the cursor is at the end of line that is being composed, then upon typing a letter, the editor scans the text above and below the current line. If a word beginning with the letters that you have just typed is found in these lines, then the editor will "complete" this word for you by writing the remaining part of the word from the current cursor position. If this word suits you, press **Alt+Right** (**Alt+<right arrow>**) and the editor will append the remaining part of the word to the text as if you have typed it yourself. If the word doesn't suit you, just continue typing and the editor will accept whatever you type. At any point during the typing, you may press **Alt+Right**

You can press **Alt+Right** at any time and not only when the editor offers you to complete a word. In this case, the editor will open a list of words that begin with the typed letters. If the list does not include an applicable word, just ignore the prompt. The right pane of the Source window, if it is open, also displays the word completion list.

How to control

To disable automatic word completion, uncheck the **Automatic Word Completion** box in the **Main menu > Configure>Editor Options>** General **tab.** When the box is checked, a number placed in the **Scan Range**

7.2.2.10 Syntax Highlighting

When the **Source** window displays the source text, it marks different C language constructions with different colors. This feature improves readability. The following constructions are highlighted separately:

- Punctuation and special characters: **( ) [ ] { } . , : ;** and so on.
- Comments that begin with **//** are highlighted. Comments enclosed in the **/* */** character pairs are highlighted, if the opening and closing pairs are placed in the same line.
- Strings enclosed in double or single quotation marks.
- Keywords of the C language (**for**, **while**, and so on).

- Type names of the C language (**char**, **float**
- Library function names of the C language (**printf**, **strcpy**, and so on).

How to control

Main menu > Configure>Editor Options> General tab>Syntax Highlighting flag In addition, you can change the color for each construction. To do the latter, use any of the following items: **Main menu > Configure> Environment > Colors tab.**

7.2.2.11 The Display from Line Number Dialog

Use this dialog to display the source file in the active **Source** window starting with a specified line. Enter the line number or select any previous number from the **History** list. The number of the first line is 1.

7.2.2.12 The Quick Watch Function

The **Quick Watch** function works as follows: if you roll the mouse pointer over a variable name in the **Source** window or the **Script Source**window, a small box containing the value of the variable will be opened. This box disappears upon moving the mouse off the object.

7.2.2.13 Block Operations

Block operations apply an editing action to more than one character at once. The **Source** window supports persistent blocks and performs a full range of operations with standard (stream), vertical (column) and line blocks of text.

Non-persistent blocks In this mode, once a block is marked, you have to immediately carry out an operation with it (delete, copy, etc.), because any movement of cursor takes the marking off the block. If a block is marked, then any entered text will replace the block with the typed text.

Persistent blocks In this mode, the block remains marked until the marking is explicitly removed (hot key **Shift+F3Ctrl+X**). The Paste operation for persistent blocks has specifics. Two additional block operations are available for persistent blocks: fast copy and fast move. These operations do not use the clipboard and require fewer manipulations of the keyboard.

To enable the persistent block mode check the namesake box on the **Main menu > Configure>Editor Options> General tab.**

Standard blocks The standard (stream) block contains a "text stream" that begins from the initial line and column of the block and ends at the final line and column.

The **Standard blocks** is enabled by default.

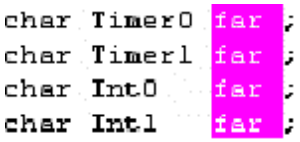
Line blocks The line block contains whole lines of text. To mark a line block, put the cursor anywhere in the first line and press **Alt+Z**; then put the cursor anywhere in the last line of the block and press **Alt+Z** once more (the latter is not necessary if the block is to be immediately deleted or copied to the clipboard).

Line blocks are always available.

Vertical blocks The vertical block contains a rectangular text fragment. Characters within the block, which goes beyond the end of the line, are considered to be spaces. Vertical blocks are convenient in cases like the following example of source text:

```
char Timer0 far ;
char Timer1 far ;
char Int0 far ;
char Int1 far ;
```

Assume the word "far" is to be moved to the place right after the word "char" in each line. The stream blocks are of little help here. However the task can be easily done with one vertical block. Mark the persistent vertical block containing the word "far" in each line, place the cursor on the first letter of word "Timer0" and press **Shift+F2**



Checking/Clearing the **Vertical Blocks** box toggles between the vertical block and the stream block modes in the the **Main menu > Configure>Editor Options> General tab.** The standard blocks are enabled by default; i.e. the **Vertical Blocks** box in the **Editor Options** dialog is unchecked by default. The line blocks are always accessible, irrespective of the status of the **Vertical Blocks** box.

To mark a block, either move the mouse while pressing its left button or use the arrow keys of the keyboard while pressing the **Shift** key. To unmark the block, press **Shift+F3**.

Copying / moving blocks

A marked block can be copied or moved within the same **Source** window in two ways: directly (fast copying, fast moving) and through the clipboard (Copy/Cut-n-Paste). Copying and moving blocks between the windows, or to another application should always be made through the clipboard.

Note. The result of copying the stream or vertical non-persistent block depends on the INSERT mode. If the mode is enabled, then the block is inserted into the text, starting at the cursor position; otherwise the copied block overwrites the text on an area of equivalent size.

Fast copying (moving) the blocks in the same window directly (without the clipboard) is convenient because it requires pressing of keys only once per operation. Mark a persistent block, then place the cursor at the destination position and press **Shift+F1** to copy, or **Shift+F2** to move.

7.3 How to start and debug script files

Starting scripts

Scripts can be started and restarted in several ways. The easiest one uses the commands of the **Script Files dialog**:

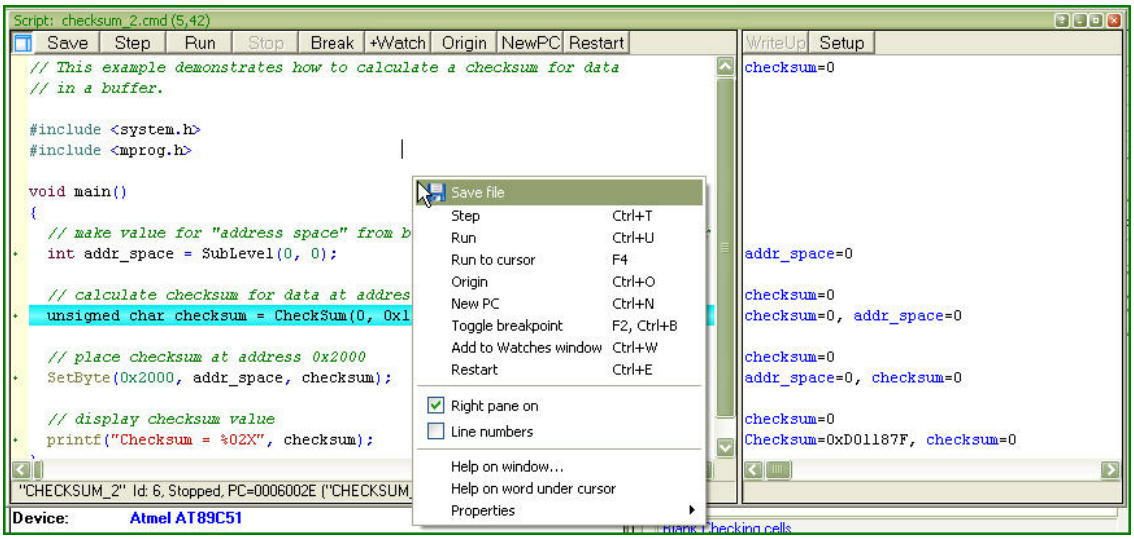
- to start a new script enter the file name into **Start new script file** box and click the **Start button** in the bottom part of the dialog box;
- to restart a stopped script highlight its name in the dialog window that displays all the loaded scripts and click the **Restart button**.

A script can be also started by means of the **StartCommandFile() function** executed by another running script.

Debugging scripts

A script can be started for an immediate execution (read above) and can be launched in the **Debug mode**

that usually is necessary while you master the script and need to check if it properly works and make necessary corrections in it. To start the script debugging highlight its name in the **Script Files dialog** window and click the **Debug button -** the program opens the window with the script file's editable text. right one is the **AutoWatches pane**. If you check the **Debug box** then every time when you start a script it will automatically switch to the , stop the script execution and open the window with the script file.



Syntax constructions and the lines, which correspond to the current PC value (blue strip) and the breakpoints (red strips), are highlighted in the script file text (for more information, see Syntax Highlighting).

Local menu and toolbar

Command	Window Toolbar	
Step	**Step**	Executes one operator of the script.
Run	**Run**	Starts continuous execution of the script in the window. Then the script execution can be broken either by hitting a set breakpoint or by the command **Stop**
Run to Cursor		
	Stop	Stops the running script.
Origin	**Origin**	Displays the source text from the line whose address corresponds to the script file Program Counter. This operation is not available when source text lines do not exist for the program addresses.
New PC	**New PC**	Sets the script file's Program Counter value to the

		address corresponding to the line where the caret is positioned.
		Sets up or clears the breakpoint at the address corresponding to the line where the caret is positioned. When you execute the **Run** or **Run to cursor** command the program execution will be stopped at the breakpoint.
Add to Watches Window	**+Watch**	Opens the **Watches** window (if not yet opened) and places the name at the caret position into it.
Restart	**Restart**	Restarts the highlighted script.

Note. To get help on a function or variable, point to the function or variable with the cursor and click. For more information, see **How to Debug a Script File** and **Script Files.**

For customizing the ChipProg**User window** and the **I/O Stream window.**

7.3.1 The AutoWatches Pane

The ChipProgUSB program displays a visible portion of the script in the **Script** window. The names of variables, called which belong to the visible script lines, are listed together with their current values in the right pane of the window. When you scroll through the **Script** window the contents of the **AutoWatches** pane automatically refreshes.

AutoWatches can be presented in the pane in the binary, hexadecimal, decimal or ASCII formats. To set the format you need to click the **Setup** button on the pane local toolbar or right click on the pane space to open the local menu.

7.3.2 The Watches Window

While the **AutoWatches** pane of the Script window displays values of the script variables visible in the current window scope you may want to monitor changing other explicitly specified script variables and expressions. To do so the ChipProgUSB allows opening the **Watches** windows. For each variable, the window displays its name, value, type and address, if any.

A newly opened **Watches** window has one **Main** tab. You can add custom tabs (with the **Display Options** command of the local menu) or rename any existing tabs. The tabs operate independently of each other; each tab is functionally equivalent to a separate **Watches** window. However, if needed, you can open several **Watches** windows.

Each of the above windows has the **+Watch** button on its toolbar. Clicking this button opens a dialog for adding a selected object to the **Watches** window.

Grids in the Watches window

For better readability the Watches window can be divided in cells by vertical and horizontal grid lines. Enable the grids to be visible within the **Watches** window by checking the corresponding boxes in the **Configure** menu > **Environment** > **Fonts** tab.

Local menu

The window local menu contains the following commands, most of which are duplicated by corresponding buttons on the window toolbar.

Command	Description
Add Watch	Adds one or more objects to the window. Opens the **Add Watch** dialog to choose an object by name. Also, you can enter an expression as a name.
Delete Watch	**Watches** window.
Delete All Watches	Deletes all watches from the window.
Modify	Opens the **Modify dialog** to set a new value for a selected variable. Alternatively, just enter the new value.
Move Watch Up	Moves a selected watch up the list.
Move Watch Down	Moves a selected watch down the list.
Display Options	Opens the **Display Options**dialog to change the display settings for a selected object and also to add/delete tabs to/from the window.

7.3.2.1 The Display Watches Options Dialog

Use this dialog to set the display options for the selected variable or in the **Watches window.**

Element of dialog	Description
Watch Expression	Contains a selected expression. The drop–down list contains the previously used expressions.
Display Format	Specifies the format for displaying a selected expression (binary, hexadecimal, decimal or ASCII).
Pop-up Description	Contains check boxes that let you choose formats for displaying pop-up SFR descriptions.
Display Bit Layout	If this box is checked the SFR bits will be displayed in the pop-up layout descriptions.
Display Bit Descriptions	Checking this box enables displaying the pop-up descriptions for the SFR bits, if any.
Auto-size Name Field	When this box is checked and when the vertical grid is visible (see note below), the window automatically adjusts the **Name** column width to fit

	the longest record in the column.
Tabs	Lists all the tabs present in the window.
Add Tab	Opens the **Add New Tab to Watches Window** dialog for entering a new tab's name. The window adds this new tab upon pressing **OK**.
Remove Tab	Removes the tab selected in the **Tabs** list.
Edit Tab Name	Opens the **Edit Watch Window Tab Name** dialog for editing the tab name.
Global Debug/ Display Options	Opens the **Debug Options** dialog.

Note. To make grids visible in the **Watches** window open the **Configure menu**, the **Environment** dialog, the **Fonts** tab and check corresponding boxes in the **Grid** field.

7.3.2.2 The Add Watch Dialog

Use this dialog box to add symbol names (for example, a variable name or an expression) to the **Watches** window. The dialog contains a list of the symbol names defined in or known to the program.

Element of dialog	Description
Name or expression to watch:	Enter into this box the symbol name or expression to be added. You can specify several names and expressions either manually (separated with semicolons) or by selecting in the list with the **Ctrl** key pressed.
History	The list of previous names and expressions.

7.3.3 The User Window

The **User** window is a window that can be created by means of the built-in **OpenUserWindow** function executed from the script itself. The **User windows** enable:

- drawing graphical objects (indicators, LEDs, buttons, arrows, etc. by means of the built-in graphical output functions;
- displaying texts in the window;
- **User windows** (see **WaitWindowEvent**).

With this capability, you can organize window operations in the interactive mode. For more information, see **Script Files**.

All functions working with windows (including the **User** window) obtain the window identifier (handle) as a parameter. Therefore, you can have several windows of this type opened at the same time.

The **User** windows do not have a local menu. They only have toolbars with 16 buttons (**0**...**F**), and each button can be programmed to perform a certain function. Pressing a button generates the **WE_TOOLBARBUTTON** event.

7.3.4 The I/O Stream Window

The **I/O Stream window** is a window that can be created by means of the built-in **OpenUserWindow**

The functions, which operate with windows (including the **I/O Stream** window), receive the window identifier (handle) as a parameter. Therefore, several windows of this type can be open at the same time.

When the text display function sends text to this window, the window displays the text from the current cursor position. To begin the next line, this function outputs '\n' (the line feed character).

The window features two text display modes: with the automatic line advance (Wrap) and without it. In the automatic line feed mode, every text line that does not fit in the window is wrapped to the next line. In the other mode, if the line does not fit in the window, its end will lie beyond the window border and will be invisible. The **Wrap** button in the toolbar toggles the window between these modes. The **Clear**

Windows of this type do not have a local menu.

8 References

8.1 Command line keys

The ChipProgUSB can be launched from the command line with addition of optional keys (parameters) that vary the program default configuration and/or automatically executes some function.

The command line mnemonic is: **UPROGNT2.EXE /Key1 /Key2 ...** , where each **'/Key'** parameter specifies a certain function as described in the table below. (more than one key can follow the executable file; each **'/KeyX'** parameter should be separated by a space symbol from each other and from the executable file). The **'/KeyX'** parameters below are case-insensitive. See the command line example below.

Keys	Description
/S<file>	Opens the program with a pre-loaded **Session configurationSession configuration** file is loaded instead of the default one. The default session file **UPROG.ses** resides in the ChipProgUSB folder.
/D<file>	Opens the program with a pre-loaded **Desktop configuration** file, the name of which is specified in the <> brackets. This **Desktop configuration** file is loaded instead of the default one. The default session file **UPROG.dsk** resides in the ChipProgUSB folder.
/O<file>	Opens the program with a pre-loaded **Option configuration** file, the name of which is specified in the <> brackets. This **Option configuration** file is loaded instead of the default one. The default session file resides in the ChipProgUSB folder.
/C"<device_name>"	Opens the program with a pre-selected device type, which is specified in the <> brackets. If the **device_name** does not exist in the ChipProg database the program will immediately open the **Select device** dialog.
/L<file>	Opens the program with a pre-loaded file, the name of which is specified in the <> brackets. A full path to the file should be specified. If the specified file cannot be found the program will immediately opens the **Load File** dialog.
/F<format>	This key works together with the **'/L<file>'** and specifies the format of the file to

	be loaded.
/A	Opens the program that is configured and ready for **AutoProgramming**. The list of the functions preset for AutoProgramming is defined by the loaded **Configuration Files**.
/I	This key is used when some external application controls the programmer and the ChipProgUSB window should be invisible. The command line with the **'/I'** key starts the ChipProgUSB program but it works in the background.
/I1	This key is similar to the **'/I'** key with the following addition - the program does not write error messages to the console buffer.
/I2	This key is similar to the **'/I1'** key with the following addition - the program copies error text to the system clipboard.
/ES<script_file>	The ChipProgUSB executes a **script file**, the name of which is specified in the <> brackets.
/GANG	ChipProgUSB program in the **multi-programming** mode.
/M	The Demo mode.

Note. The file names above must follow the parameter without a blank space.

Here is an example of the command line that controls the ChipProg:

"C:\Program Files\Chipprogusb\4_57_00\uprognt2.exe" /Lc:\work\program.hex /FH /A /I2.

The command above starts the **C:\Program Files\Chipprogusb\4_57_00\uprognt2.exe** program in the hidden mode, loads the file **c:\work\program.h** in the **Intel HEX format**, starts **AutoProgramming**, completes the application that controls the programmer and if there is an error then copies it to the computer clipboard.

8.2 Errors Messages

Enter topic text here.

8.2.1 Error Load/ Save File

5005 "Error reading file"

5004 "CRC mismatch, loading terminated"

5003 "Invalid .HEX file format"

5043 "Address out of range"

5078 "End address should be greater than start address"

5151 "Invalid file format"

5007 "Error writing file"

6899 "Cannot load file '%s': buffer #%u does not exist"

6900 "Cannot load file '%s': sub-level #%u does not exist"

7019 "Unable to open project file: '%s'.\n\nAfter start, the programmer attempts to load the most recent project. This error means that the project file does not exist on disk."

8.2.2 Error Addresses

5189 "Device start address (0x%LX) is too large.\nMax. address is 0x%LX."

5190 "Device end address (0x%LX) is too large.\nMax. address is 0x%LX."

5191 "Buffer start address is too large"

4024 "Address %s is out of range (%s...%s)"

4106 "File format does not allow addresses larger than 0xFFFFFFFF"

6626 "Buffer start address must be even"

6627 "Device start address must be even"

6628 "Buffer end address must be odd"

8002 "Buffer named '%s' already exists. Please choose another name for the buffer."

8.2.3 Error sizes

6372 "Buffer size is too small for selected split data option"

6495 "Requested buffer size (%lu) is too large"

6441 "Size of file is greater than buffer size:\nAddr = %08lX, length = %u"

6431 "Source block does not fit into destination sub-level"

6859 "File size is %u bytes that is less than header size (%u bytes), loading terminated. Probably, you have specified an invalid file format."

4107 "Cannot allocate %Lu MBytes for the buffer, maximal buffer size is %Lu MBytes"

5192 "Invalid number: '%s'"

.

8.2.4 Error command-line option

5329 "/%s command-line option: Device name required"

5330 "/%s command-line option: Missing file name"

5331 "/%s command-line option: Missing file format tag"

5332 "/%s command-line option: Invalid file format tag"

5333 "Command line: unable to determine the file format"

4104 "Command-line option /I ignored because /A option is not specified"

8.2.5 Error Programming option

6409 "Invalid programming function or menu name:\n'%s'"

6410 "Invalid programming option name '%s'"

6902 "Invalid '%s' programming option value string: '%s'"

6411 "Programming option '%s' cannot be changed"

6412 "Programming option string is too long.\nMax. length is %u."

6854 "Programming option '%s' has type of '%s'. Use '%s()' script function to get the value of this option."

5188 "Value %.2f is out of range of %.2f...%.2f for programming option '%s'"

6561 "Value %ld is out of range of %ld...%ld for programming option '%s'"

4001 "Not all of the saved auto-programming functions were restored. Check the auto-programming functions list."

8.2.6 Error DLL

6499 "Cannot find bit resource with id 0x%X in DLL:\n'%s'"

6500 "Error handling bit resource with id 0x%X in DLL:\n'%s'"

6502 "Unable to find device '%s' in DLL:\n'%s'"

8.2.7 Error USB

4015 "USB device driver error 0x%04X in '%s'.\n\nCannot recover from this error, exiting.\n\nPlease check if the programmer power is on. If yes, disconnect the USB cable from computer and connect it again, then restart the %s shell."

4016 "All sites reported USB device driver error.\n\nCannot recover from this error, exiting.\n\nPlease check if the programmer(s) power is on. If yes, disconnect the USB cable from computer and connect it again, then restart the %s shell."

4017 "The following site(s):\n\n%s\n\nreported USB device driver error.\n\nThese site(s) will be removed from the gang programming process.\n\nPlease check if the programmer(s) power is on. If yes, disconnect the USB cable from computer and connect it again, then restart the %s shell."

8.2.8 Error programmer hardware

6546 "Source area does not fit into destination address space"

4005 "Attempt to read memory beyond buffer end: Addr = %s, len = %u bytes"

6988 "Unable to establish connection with programmer hardware. Please check if:\n\n"

4006 "Attached programmers have duplicate serial number '%s'"

4010 "This programmer with serial number '%s' has been already assigned the site number = %u"

4011 "This gang programmer with serial number '%s' has been already assigned the site numbers = %u..%u"

4013 "The programmers attached are of different types and cannot be used for gang mode.\n\nExiting."

4014 "ExecFunction() does not work in Gang mode"

4020 "%s reported hardware error 0x%X, error group 0x%X. If problem persists, please contact Phyton."

4000 "The attached programmer with id = %u is not supported"

4102 "Device programming countdown value is zero%s"

8.2.9 Error internal

4025 "Internal Error: Unable to allocate %u bytes for the buffer. Please contact Phyton."

8.2.10 Error confiquration

6503 "No programmer configuration files found (prog.ini)"

5325 "The device type '%s %s' stored in configuration "
"or choosen from script file function 'SetDevice()' is not supported by %s.\n"
"The device '%s %s' will be selected.\n"
"Use 'Configure / Select device' to choose the device "
"you need to operate on."

4002 "The '%s' configuration option has been set to an illegal state due to the data read from file. Setting this option to its default state ('%s')."

8.2.11 Error device

5326 "Device selection error"

8.2.12 Error check box

6852 "Error in check box option specification string: '=' expected"

6853 "Cannot find check box option string '%s'"

8.2.13 Error mix

5195 " Number of repetitions cannot be zero"

5206 "The 'View only' option is on; editing disabled. Click the 'View' button on toolbar to enable editing."

6501 "No power-on tests defined in:\n'%s'"

6903 "'%s' is a sub-menu name, not a function name"

6401 "No more occurences"

6387 "Invalid fill string"

5172 "Checksum = %08lX"

5311 "No more mismatches"

8.2.14 Warning

5339 "Warning: JEDEC file has invalid CRC"

6933 "Warning: no 'file end' record in file"

6845 "Attention! The %s %s device must be inserted into the programmer's socket shifted by %d row(s) relative to the standard position as shown in the Device Information window."

6846 "Attention! Insert device into socket shifted by %d row(s) as shown on the picture."

8.3 Expressions

Expressions in the program are the mathematical constructions for calculating results with the use of one or more operands. It supports various operations on expressions. The following operands are used:

- numbers
- example of expressions

When a number is required, you may use an expression; <%CM%> will accept the value of the expression. For example, when using the **Modify** command in the **Buffer** window, you can enter the new value in the form of a number or arithmetic expression.

Interpreting the expression result

The expression result is interpreted in accordance with the context in which it is used.

In the dialog box, when an address is required, the program tries to interpret the expression's value as the

address. If you enter a variable name, the result of the expression will be the variable's address but not the value of the variable.

If the dialog expects a number to be entered, the expression's value will be interpreted as a number (for example, the **Modify Memory** dialog box of the **Buffer Dump** window). If you enter a variable name there, then the result will be the value of the variable, but not its address.

Nonetheless, you can follow the default rules:

If you need to use the variable's **value**, where an address is expected, then you can write something like `var + 0`. In this case, the variable's value will be used in the expression.

If you need to use the variable **address&** (address) operation, that is, `&var`.

8.3.1 Operations with Expressions

The program supports all arithmetic and logical operations valid for the C language, as well as pointer and address operations:

Designation	Description
( )	Brackets (higher priority)
[ ]	Array component selector
.	Structure component or union selector
->	Selection of a structure component or a union addressed with a pointer
!	Logical negation
~	Bitwise inversion
-	Bitwise sign change
&	Returns address
*****	Access by address
(type)	Explicit type conversion
(sizeof)	(returns size of operand, in bytes)
*****	
/	Division
	Modulus operator (produces the remainder of an integer division)
+	Addition
-	Subtraction
<<	Left shift
>>	Right shift

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
	Not equal to
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
	Logical OR
=	Assignment

The types of operands are converted in accordance with the ANSI standard.

The results of logical operations are 0 (false) or 1 (true).

Allowed type conversions:

- Operands can be converted to simple types (char, int, ... float).
-
- The word "struct" is not necessarily (MyStruct *).

8.3.2 Numbers

By default, numbers are treated as decimals. Integers should fit into 32 bits; floating point numbers should fit into the single precision format (32 bits).

The following formats are supported:

1) Decimal integer.

Example: `126889`

2) Decimal floating point.

Examples: `365.678`; `2.12e-9`

3) Hexadecimal.

<%CM%> understands numbers in C format and assembly format.

Examples: `0xF6D7`; `0F6D7H`; `0xFFFF1111`

4) Binary.

Binary numbers must end with 'B'.

Examples: `011101B`; `1111111111111111000011B`

5) Symbol (ASCII).

'a'; 'ab'; '$B%8'.'.

8.3.3 Examples of Expressions

Examples of expresions

#test#i + #test#j << 2
(unsigned char)#test#i + 2
sizeof(##array) > 200

main

i + j << 2 / :CW0x1200

(unsigned char)i + 2

sizeof(array) > 200

(a == b && a <= 4) || a > '3'

sptr -> Member1 -> a[i]

*p

*((char *)ptr)

8.4 Script Language

The program ChipProgUSB can execute so-called script files in a way similar to how DOS executes the batch files.

The main objective of script files is to automate usage of the emulator. Using script files makes it possible to load programs, set up breakpoints, start program execution, manipulate windows and perform any actions available to you in automatic (batch) mode. It is also possible to display various messages in the **Console window** or other special windows, to create user's custom menus, etc. There is the option of displaying any graphical data in special windows.

The script language is similar to C: almost all C constructions are supported, except for structures, conjunctives and pointers. However, there are some differences. There are also many built-in functions available, such as printf(), sin() and strcpy().

The extension of script source file is **.CMD**.

Simple example of a script file

How to write a script file

How to start a script file

How to debug a script file

Description of Script Language

Script Language Built-in Functions

8.4.1 Simple example

This example shows how to load a file and automatically program it and display the result.

```
#include <system.h>
#include <mprog.h>

void main()
{

  LoadProgram("test.hex", F_HEX, SubLevel(0, 0));                    // load file "test.hex" that is an Intel HEX
file                                                                                                                //
to buffer 0, sub-level 0

  if (ExecFunction("Auto Programming") == EF_OK)                     // perform an automatic programming
  {
   if (ExecFunction("Verify", SubLevel(0, 0), 10) != EF_OK)          // verify 10 times
   {
    printf("Verify failed: %s", LastErrorMessage);                   // display error message if verify failed
    return;                                                          // terminate script
   }
   printf("Verify ok.");                                             // display Ok result
  }
  else
   printf("Programming failed: %s", LastErrorMessage);               // display error message
}
```

8.4.2 Description

The language used for writing the script files is similar to the C language. If you are familiar with the C language, you can skip this chapter and switch to reading about the differences between the script language and the C language.

This manual contains just a few examples of programming in the script language. To find more examples, refer to books on the C language.

General Syntax of Script Language
Basic Data Types
Data byte order
Operations and Expressions
Operators
Functions
Descriptions

Predefined Symbols in the Script File Compilation

8.4.3 Built-in Functions

The script file system provides you with a large set of built-in functions intended for work with lines, files, for mathematical calculations, and access to the processor resources. The **system.h** file contains descriptions of these built-in functions. You should include the **system.h** file in the script file source text with the #include directive:

You can use these built-in functions in the same way you use any function that you have defined.

Buffer access functions

Device programming control functions

String Operation Functions
Character Operation Functions
Functions for File and Directory Operation
Stream File Functions

Script File Manipulation Functions
Text Editor Functions
Control Functions
Windows Operation Functions and Other System Functions
Graphical Output Functions
I/O Stream Window Operation Functions

Event Wait Functions
Other Various Functions

Note. To get help on a function or variable, while editing the script source with the <%CM%> built-in editor, point that function/variable name with the cursor and hit **Alt+F1**.

8.4.4 Built-in Variables

You can access script language built-in variables in the same way as regular global variables. However, some built-in variables are accessible only for reading, and in case of attempt to write to such variable.

The built-in variables are declared in the **system.h** header file.

Programming variables:
InsertTest
ReverseBytesOrder
BlankCheck
VerifyAfterProgram
VerifyAfterRead

ChipEndAddr
BufferStartAddr
LastErrorMessage
DialogOnError

Text editor built-in variables:
InsertMode
CaseSensitive
WholeWords
RegularExpressions
BlockCol1
BlockCol2
BlockLine1
BlockLine2
BlockStatus
CurLine
CurCol
LastFoundString

Miscellaneous variables:
WorkFieldWidth
WorkFieldHeight
ApplName[]
DesktopName[]
SystemDir[]
errno
_fmode
MainWindowHandle
NumWindows
WindowHandles[]
SelectedString[]
LastMessageInt
LastMessageLong

8.4.5 Difference between the Script and the C Languages

The script files are written in a C-type language and you should not expect it to meet standards. Many features are not supported because they are not necessary and complication of the language can cause compiler errors (the script file language compiler is not a simple thing).

- Pointers are not directly supported. But arrays are supported, therefore a pointer can always be built from an array and element number. Note that, for example, string operation functions, such as strcpy, receive a string and a byte number (index) as parameters, which form the pointer. In function

declarations, index is equal to zero by default.

- Pointers to functions are not supported. If necessary, a table call can always be replaced with the **switch** operator.
- Multidimensional arrays are not supported. If it is necessary, you can write a couple of functions, such as:

```
int GetElement(int array[], int index1, int index2);
```

- Structures (and unions) are not supported. In fact, you can always do without structures. Structures may be required for API Windows and user DLLs operations, but as a rule only experienced programmers should do it, such as those who know how to reach structure elements. As a tip, there are functions, such as memcpy, which receive a void "pointer").
- Enumerated types (**enum**) are not supported #define.
  #define half(x) (x / 2), are not supported. The same operations can be done with functions.
- Conditional operators such as **x = y == 2? 3 : 4;**, are not supported; the operator "comma" outside variable declaration is not supported. For example,

int i = 0, j = 1; is supported, but
for (i = 0, j = 1; ...) is not supported.

- **User** functions with a variable amount of parameters are not supported. However, there are many system functions, such as printf, with a variable number of parameters.
- Declaration of **user** function parameters such as **void array[]** is not supported. The system functions such as memcpy, have such parameters.
- Logical expressions are always fully computed. It is very important to remember it, as a situation like

```
char array[10];
if (i < 10 && array[i] != 0)
  array[i] = 1;
```

will cause an error at the execution stage, if i is greater than 9, because the expression of array[i] will be computed. In a standard compiler such an expression is not computed, because the condition of **i > 10** would cancel any further processing of the expression.

- Constant expressions are always computed during execution. For example, **int i = 10 * 22** will be computed not during compilation, but during execution.
- The **const** key word is absent.
- Static variables cannot be declared inside functions.

But

- Variables can be declared anywhere, not just in front of the first executed operator. For example:

```
void main()
{
GlobalVar = 0;
int i = 1;       // will be OK as in C++
}
```

- Nested comments are allowed.
- Expressions like **array = "1234"** are allowed.
- Default parameter values in declared functions, as in C++, are allowed. For example, **void func (char array[],int index = 0);**. Expressions can also serve as default values, for example **void func(char array[], int index = func1() + 1);**.
- Expressions in global variable initializers are allowed. For example:

```
float table[] = { sin(0), sin(0.1) };

void main()

  ...
}
```

8.4.6 Script Language Built-in Functions and Variables

The list below includes all the names of the script language built-in functions and variables:

AllProgOptionsDefault
API
ActivateWindow
AddButton
AddWatch

BackSpace
BlankCheck
BlockBegin
BlockCol1
BlockCol2
BlockCopy
BlockDelete
BlockEnd
BlockFastCopy
BlockLine1
BlockLine2
BlockMove
BlockOff
BlockPaste
BlockStatus
BufferStartAddr
CaseSensitive
CheckSum
ChipEndAddr
ChipStartAddr
ClearWindow
CloseProject
CloseWindow
Cr
CurChar
CurCol
CurLine
Curcuit
DelChar
DelLine
DesktopName[]
DialogOnError
DisplayText
DisplayTextF
Down
Ellipse

Eof
Eol
ExecFunction
ExecMenu
ExecScript
ExitProgram
Expr
FileChanged
FillRect
FindWindow
FirstWord
FloatExpr

ForwardTillNot
FrameRect
FreeLibrary
GetByte
GetDword
GetFileName
GetLine
GetMark

GetProgOptionBits
GetProgOptionFloat
GetProgOptionList
GetProgOptionLong
GetProgOptionString
GetScriptFileName
GetWindowHeight
GetWindowWidth
GetWord
GotoXY
InsertMode

Inspect
InvertRect
LastChar
LastErrorMessage
LastEvent
LastEventInt{1…4}
LastFoundString
LastMessageInt
LastMessageLong
LastString
Left
LineTo
LoadDesktop

LoadLibrary
LoadOptions
LoadProgram
LoadProject
MainWindowHandle
MaxAddr
MessageBox
MessageBoxEx
MinAddr
MoveTo
MoveWindow

OpenEditorWindow
OpenStreamWindow
OpenUserWindow
OpenWindow
Polyline
ProgOptionDefault
Rectangle
RedrawScreen
RegularExpressions
ReloadProgram
RemoveButtons

Right
SaveData
SaveDesktop
SaveFile
SaveOptions
Search
SearchReplace
SelectBrush
SelectFont
SelectPen
SelectedString[]
SetBkColor
SetBkMode
SetByte
SetCaption
SetDevice
SetDWord
SetFileName
SetMark
SetMemory

SetProgOption
SetTextColor

SetToolbar
SetUpdateMode
SetWindowFont
SetWindowSize
SetWindowSizeT
SetWord
SystemDir[]
TerminateAllScripts
TerminateScript
Text
Tof
Up
UpdateWindow
VerifyAfterProgram

WaitEprTrue
WaitGetMessage
WaitSendMessage
WaitWindowEvent
WholeWords
WindowHandles[]
WindowHotkey
WordLeft
WordRight
WorkFieldHeight
WorkFieldWidth
_GetWord
_ff_attrib
_ff_date
_ff_name
_ff_size
_ff_time
_fmode
_fullpath
_printf
abs
acos
asin
atan

atoi
ceil
chdir
chsize
clearerr
close
cos

creat
creatnew
creattemp
delay
difftime

eof
errno
exec
exit
exp
fabs

fdopen
feof
ferror
fflush
fgetc
fgets
filelength
fileno
findfirst
findnext
floor
fmod
fnmerge
fnsplit
fopen
fprintf
fputc
fputs
fread
freopen
frexp
fscanf
fseek
ftell
fwrite
getc
getcurdir
getcwd
getdate
getdfree
getdisk()
getenv
getftime

gettime

inport

isalnum
isalpha
isascii
isatty
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit

lock
locking
log
log10
lseek
ltoa
memccpy
memchr
memcmp
memcpy
memicmp

memset
mkdir
movmem
mprintf
open
outport
outportb
peek
peekb
poke
pokeb
pow
pow10
printf
pscanf
putc
putenv

putw
rand
random
randomize
read
rename
rewind
rmdir
scanf
searchpath
setdisk
setftime
setmem
setmode
sin
sprintf
sqrt
srand
sscanf
stpcpy
strcat

strcmp
strcmpi
strcpy
strcspn

strlen
strlwr
strncat
strncmp
strncmpi
strncpy
strnicmp
strnset
strpbrk
strrchr
strrev
strset
strspn
strstr
strtol
strtoul
strupr
tan
tanh
tell

toascii
tolower
toupper

unlink
unlock
wgetchar
wgethex
wgetstring
wprintf
write

8.5 In-System Programming for different devices

NOTE! Always carefully check connecting your ChipProg programmer to the target. Wrong connecting may and probably will cause destruction of the programmer's and/or the target system's hardware.

Most embedded microcontrollers have different algorithms for the ISP procedure. See the following topics regarding the ISP for popular microcontrollers:

Specific of the in-system programming of the Microchip PICmicro

Specific of the in-system programming of the Atmel AVR microcontrollers

Specific of the in-system programming of the Atmel 8051 microcontrollers

8.5.1 Specific of programming PICmicro

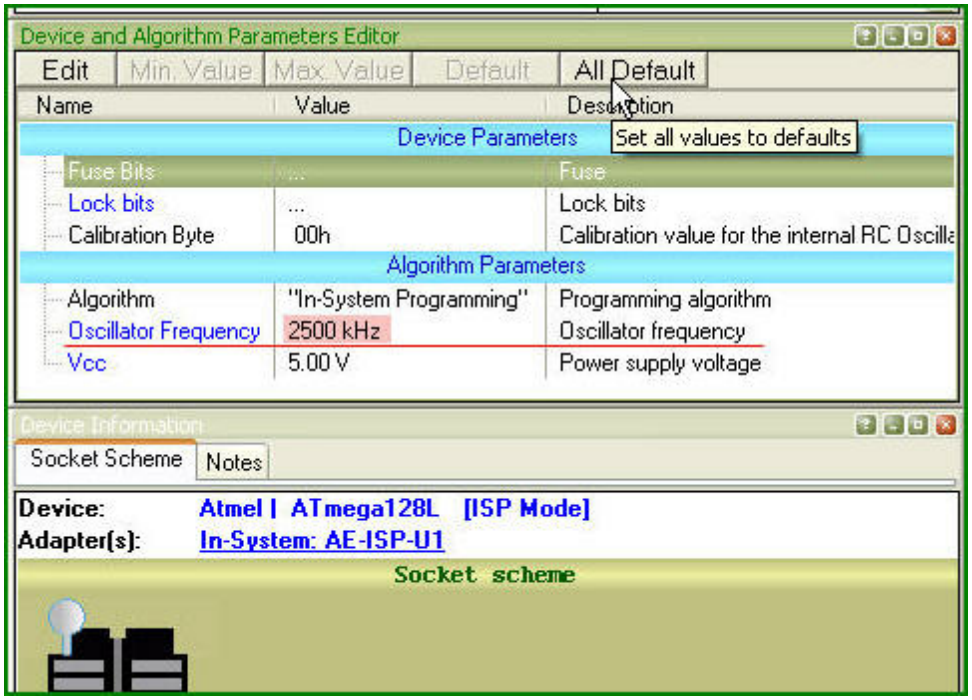
1. Most of the PIC microcontrollers produced by Microchip Technology Corporation require a special HV ISP Programming Mode (High-Voltage in-System Programming Mode). In this mode a relatively high voltage of 13V is applied to the MCLR device pin. The user's equipment to be programmed should be designed in the way tolerating a 13V signal to be applied to the MCLR device pin - in particular this pin should not be connected to the Vcc pin of the device.

2. Though the PIC microcontrollers are capable to work in a certain range on the Vcc voltage (the range varies from 2 to 5V for some PICmicro derivatives) the device being under programming must have the 5V voltage level applied to the Vcc device pin. If in the working mode the target microcontroller works under the Vcc lower than 5V and the target cannot tolerate applying the 5V voltage to the Vcc pin, then, if the user needs to program the PICmicro device in-system, it is necessary to change the schematic to have an ability to connect 5V to the Vcc pin while the target is under the programming. However, verification of the correct programming can be conducted under the voltages allowed by the manufacturer (Vcc min - Vcc max).

8.5.2 Specific of programming AVR microcontrollers

Microcontrollers of the Atmel AVR series can be programmed in-system being under a normal Vcc voltage. Practically all AVR microcontrollers require clocking while they are under in-system programming. ChipProg programmers are capable to send clocks to the target microcontroller but sometimes the systems based on AVR microcontrollers have their own built-in clock generators.

1. If the system **has its own built-in clock generator** then make sure that the clock line of the ChipProg**is not connected** to the clock input pin of the target microcontroller, otherwise it may destroy either the target or programmer hardware. What you need in this case is just to enter a value of the generator clock frequency in the **Algorithm Parameters > Oscillator Frequency** field in the **Device and Algorithm Parameters Editor** window (see on the picture below). By default the **Oscillator Frequency** value is 2.5 MHz. To change it double click the **Oscillator Frequency** line displayed in blue color and enter the Fclk value into the popped up dialog. If the actual clock frequency differs from the value set in the window the correct programming will be impossible.



2. If the target system **does not have its own built-in clock generator** then, the target AVR device needs to get clocks from the built-in generator; thus the clock output wire of the cable-adapter should be connected to an appropriate clock input pin of the target device. By default the Fclk= 2.5 MHz. It can be set in the range of the Fclk allowed for a particular selected target AVR device in the **Algorithm Parameters > Oscillator Frequency** field in the **Device and Algorithm Parameters Editor** window (see the picture above).

8.5.3 Specific of programming Atmel 8051 microcontrollers

Microcontrollers of the Atmel 8051 family (AT89 series) can be programmed in-system being under a normal Vcc voltage. Practically all these microcontrollers require clocking while they are under in-system programming. ChipProg programmers are capable to send clocks to the target microcontroller but sometimes the systems based on the Atmel 8051 microcontrollers have their own built-in clock generators.

1. If the system **has its own built-in clock generator** then make sure that the clock line of the ChipProg cable adapter **is not connected** to the clock input pin of the target microcontroller, otherwise it may destroy either the target or programmer hardware.

2. If the target system **does not have its own built-in clock generator** then, the target device needs to get clocks from the ChipProg built-in generator; thus the clock output wire of the cable-adapter should be connected to an appropriate clock input pin of the target device.

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

Back Cover