Phyton™

ChipProg Device Programmers

User's Guide

ChipProg-48

ChipProg-40

ChipProg-G4

ChipProg-ISP

ChipProg Device Programmers

© 2008 Phyton, Inc. Microsystems and Development Tools

Printed: November 2008 in (whereever you are located)

Table of Contents

Part VII References   **108**

1   Introduction

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 has an universal software – ChipProgUSB – that controls all the USB hosted models 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 released soon.
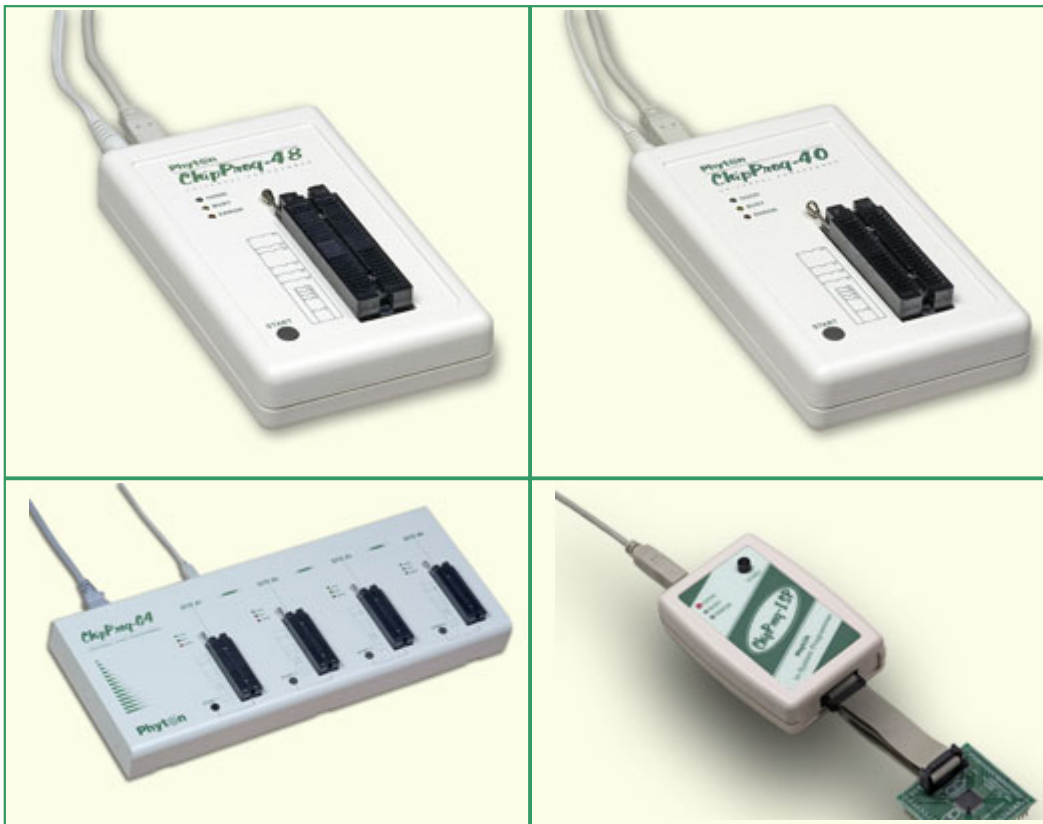
The **ChipProg-48** and **ChipProg-40** programmers are intended for engineering and small volume manufacturing. These models allow operating on the devices before they are installed in the equipment (parallel programming) as well as on the devices already installed in the user's equipment (the method 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 **Script Files**.

The ChipProgUSB software runs on the IBM PC hardware platform under the control of several Windows™ versions (see the **System requirements**).

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 package or **Package**	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.%CPN%> -40, ChipProg-48 and ChipProg-G4 are equipped with 40- and 48-pin ZIF sockets allowing operation on any DIP-packed devices with different numbers of leads and different widths and also connecting additional adapters for programming devices in other packages.
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 EEPROM** and **Identifier 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 98/2000/XP/Vista OS for the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers
- Windows 2000/XP/Vista for the ChipProg-G4 programmer
- 256MB of RAM
- At least one USB port
- A hard drive with at least 200MB of free space

2   ChipProg Family Brief Description

The Phyton family of the ChipProg programmers at the moment of making this Help Manual included:

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

New ChipProg programmer models will be added soon.

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
PC interface	USB, 2.0	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
Capability to support new target devices	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	No
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. An unlimited number of 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. The self-testing procedure automatically executes at any time the programmer is powered on.

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 **ChipProg-48** model.

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
Hardware characteristics
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. Can program target devices in the programmer ZIF socket as well as the devices installed in the equipment (ISP mode).
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

1. The programmer has a 40-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 40.
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 through 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. The self-testing procedure automatically executes at any time the programmer is powered on.

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. 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.

2.4   ChipProg-G4

The **ChipProg-G4** is a 4-site gang programmer based on four **ChipProg-48** tools enclosed in one case 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 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.
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. 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 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	/Error
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 adapters.chm 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
Hardware characteristics
Software 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** to read all the manual basic topics included in the chapters **Graphical User Interface** and **Operating with ChipProg programmers** 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
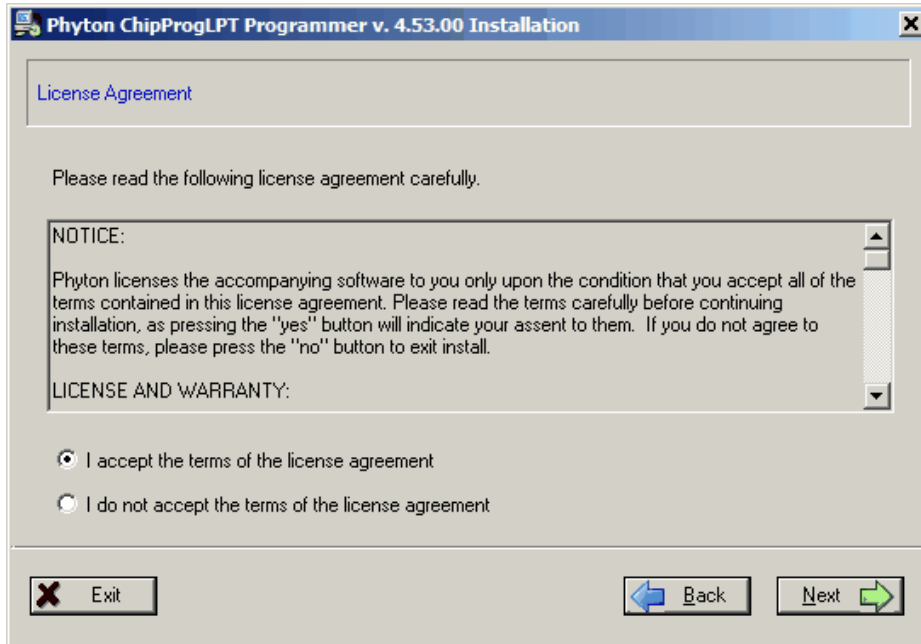
Choose the folder to install
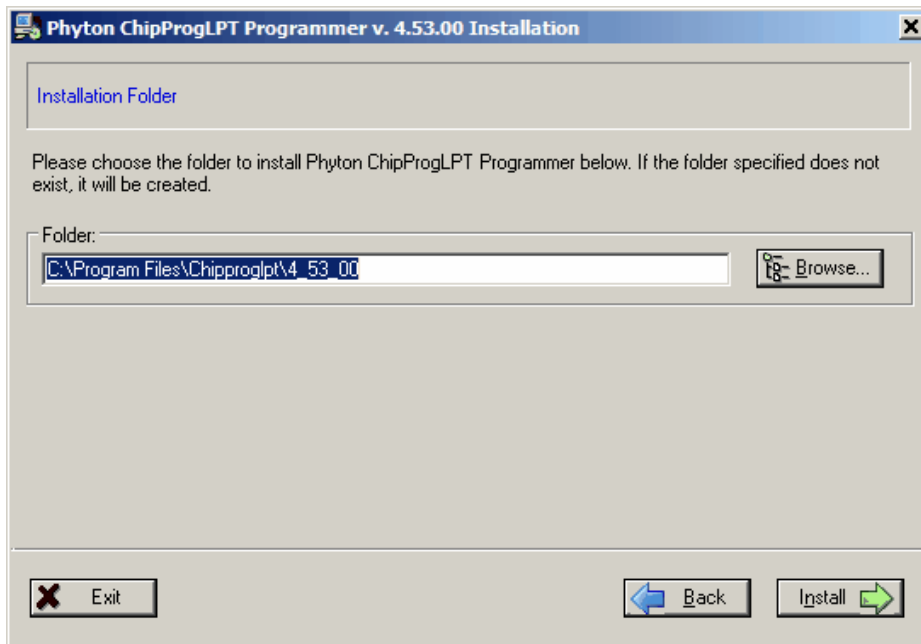
Wait for installation...
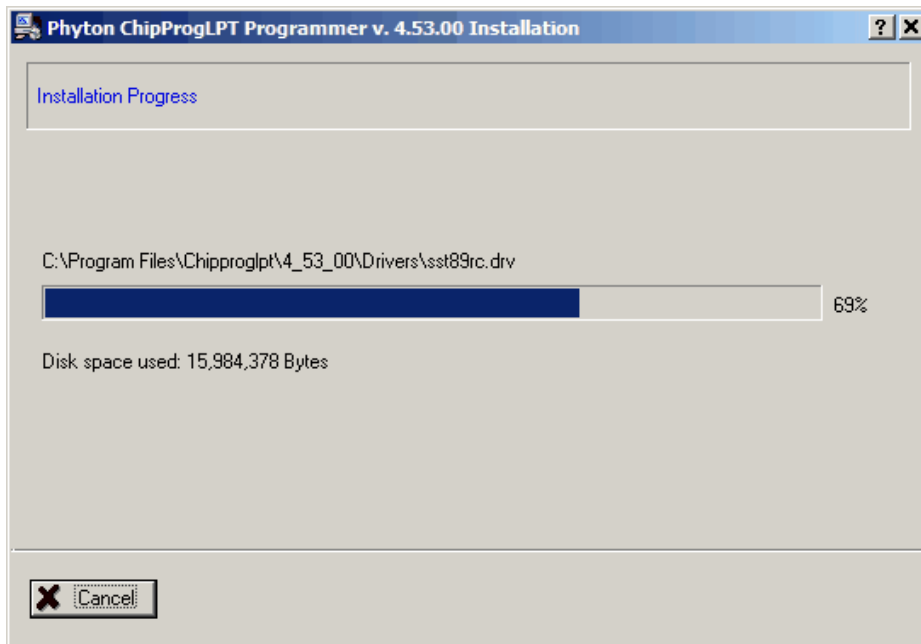
Phyton ChipProgUSB folder

At the end the installer will create a folder with ChipProgUSB tools' and documents' shortcuts:
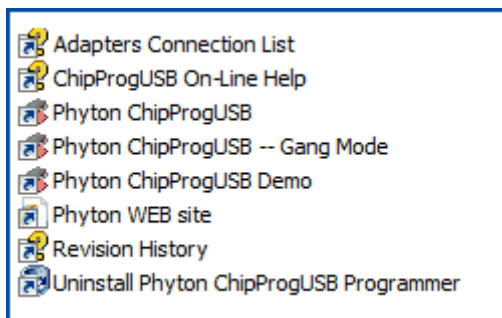
Phyton Programming Adapter List - opens the adapters.chm file that list all the Phyton programming adapters with their short descriptions and wiring diagrams.

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

Phyton ChipProgUSB - invokes the ChipProgUSB executable file and starts operations for the ChipProg-48, ChipProg-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 the ChipProg-G4 hardware. For all other ChipProg tools, e.g. for the ChipProg-48, ChipProg-40 and ChipProg-ISP programmers you must install the USB drivers after the software installation has been completed.

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:
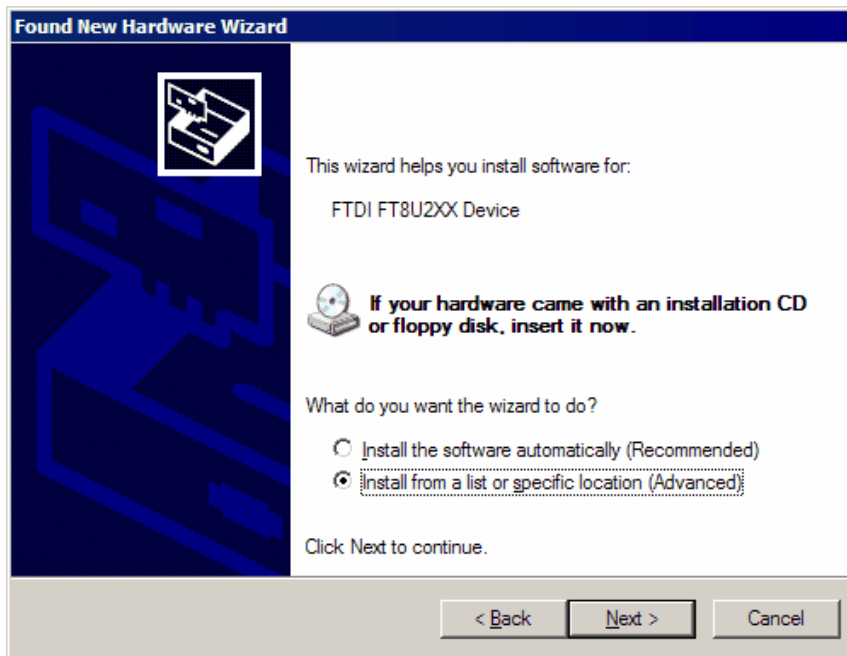
Select the **'Install from a list or specific location'** option and click the **Next** button. The screen below will appear:
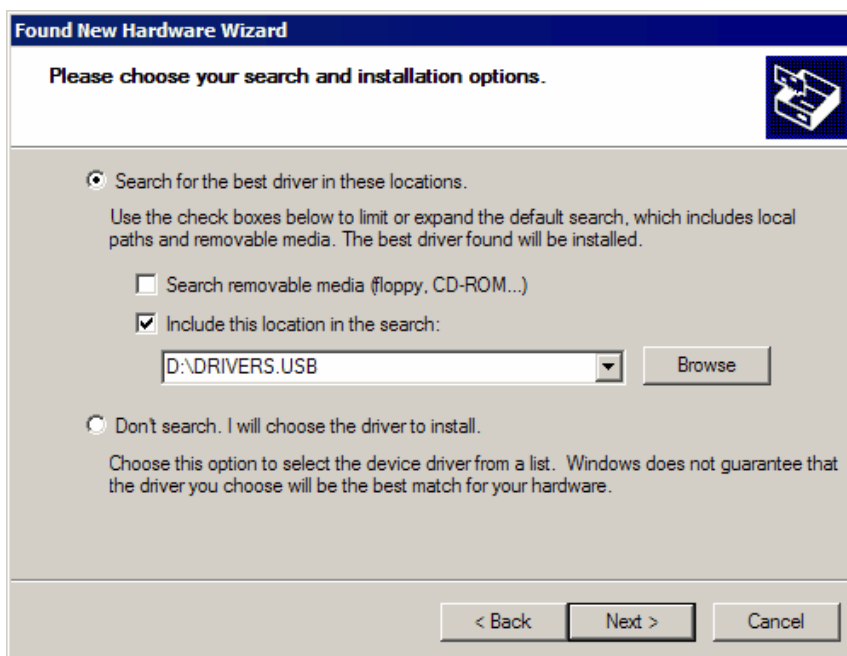
Browse to the **DRIVERS.USB** folder on the Phyton CD and click the **Next** button (certainly the drive letter can be other than D:). This will start the drivers installation.
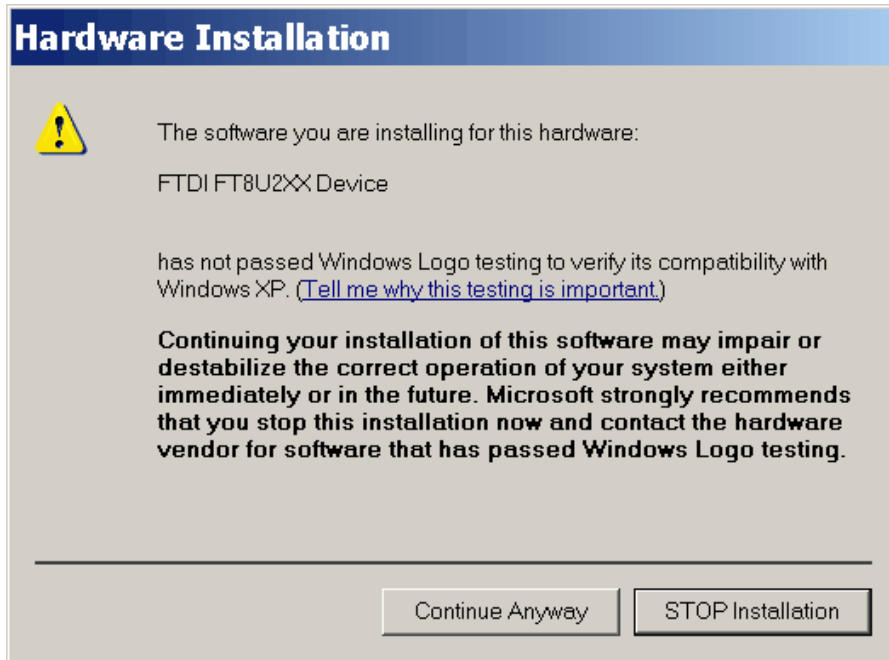
Click the **Continue Anyway** button to complete the installation; you will soon get the last prompt:
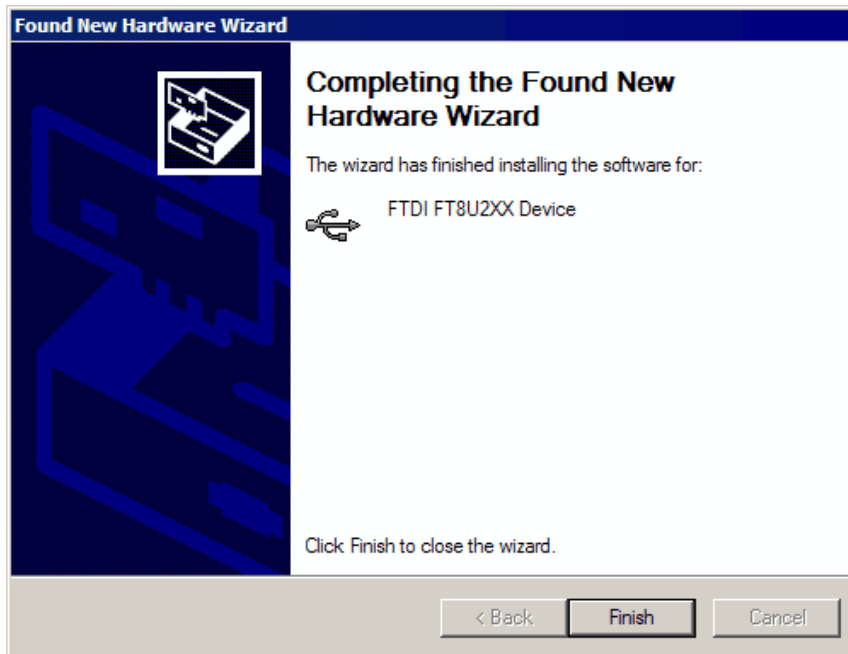
Click the **Finish** button. Now you can use the ChipProg connected to your computer.

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" green 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 ChipProgUSB 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   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 programming site. To assign the number push an appropriate **Start** 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" green LED on the programmer is on. 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. Use of the passive USB hubs for connecting the ISP programmers is not allowed.

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-ISP 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:

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 ChipProgUSB dialog, message box and menu has its own context-sensitive help, which can be invoked for the active dialog or menu by pressing **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:

- Your name, the name of your company, your contact telephone number and your e-mail address.
- 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

Phyton Inc., Microsystems and Development Tools

7206 Bay Parkway, 2nd floor
Brooklyn, New York 11204
USA

Web address: www.phyton.com

E-mail contacts:

General inquiry: info@phyton.com

Sales: sales@phyton.com

Technical Support: support@phyton.com

Tel: **1-718-259-3191**

Fax: **1-718-259-1539**

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 ChipProg 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 ChipProgUSB 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 **Buffer** 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. All check boxes and radio buttons in the dialogs work in the following way: a double-click on the check box or radio button is equivalent to a single click on the box or button, followed by a click on the **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).
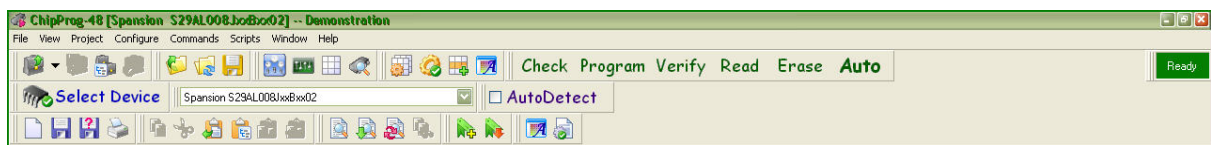
The top line, shown right under the ChipProg main window title, includes the Main menu submenus. A second line under the Main menu line displays icons and buttons of most frequently used commands on files and target devices (Open project, Load file, Save file... Check, Program, Verify, etc.). There is 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.

```
Buffer #0 - Code (1 MB), words: 00000000 [00000000]
Code
Addr | Load | Save | Configure Buffer | Setup | View | Modify | Block
File: None
Checksum: FFF80000
00000000:  FFFF FFFF FFFF FFFF FFFF FFFF
```
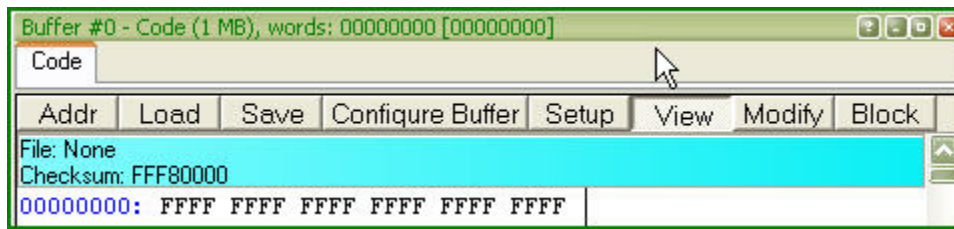
4.3   Menus

The 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 **.opt**. 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** sub command of the **Configuration Files** command. The extension of this file is **.ses**. The default file name is **UPROG.ses.**

- The **History** 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	Command	Description
	New	Opens the **Project Options** dialog.
	Open	Opens the **Open Project** dialog 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 **Project Options** dialog for reviewing or changing the project parameters.

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** dialog.
Remove 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 **Project Open History** 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 ChipProgUSB 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	Opens the **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	Moves the selected project or branch down the tree within the same level of hierarchy. The branch moves together with all branches that "grow" from it, and all their projects.
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	Writes the repository to the disk file and opens the selected 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** tab, 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.
Manufacturer	The box lists the device manufacturers in alphabetic order.
Search mask:	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 **Buffer Configuration** 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: **Start:** **End:**	The Start and End addresses define the range of buffer addresses for which the program calculates the checksum.
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.)	By checking this box you enable logging of all the events associated with the device programming, e.g. the target device replacement, loaded 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	A pair of toggled radio buttons, checking the top one sets the mode of appending new records to a specified log file and checking the second overwrites the old log every time the 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

Element of 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-G4 gang programmer is in use).
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,

Mapping Hot Keys 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.
Window Toolbar Location	Sets the toolbar location for the selected window.
Grid	Turns on/off the display of the vertical and horizontal grids in some window types, and permits adjusting the column width (when the vertical grid is allowed).
Additional Line Spacing	Provides additional line spacing, which will be added to the standard line spacing. Supply a new value or choose from the list of most recently used values.
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 **Colors** tab of the **Environment** 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** list.
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

The **Key Mapping** tab of the **Environment** dialog opens a sub dialog for assigning hot keys for all commands in the ChipProgUSB. The **Menu Commands Tree** column displays a tree-like expandable diagram of all commands. The **Key 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** button in that dialog.
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 **Editor Options** dialog sets up all common options applicable to every **Source** window opened.

Element of dialog	Description
Backspace Unindents	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	If the box is checked then <%CM%> creates a *.BAK file each time a file is saved in the **Source window**.
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 **Source** 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	Sets the maximum number of available undo steps (512 by default). If this does not suffice, you can set a value of up to 10000 steps. However, larger values increase the editor's memory requirement.
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 **Editor** in the **Windows** list and press the **Define Font** button.

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. (*Command* 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	Opens the **Edit Command** dialog to change the command parameters. For built-in commands, you can only reassign the hot keys (the **Command Description** and **Script Name** 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**	Opens the specialized dialog box where you can assign two key combinations to a couple of hot keys.

The script source files for commands will reside only in the KEYCMD subdirectory of the ChipProgUSB system folder. Enter the file name only, without the path or extension.

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.
Erase	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.
Signed Values	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` and the value of 66h will be assigned to SP.

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 **Scripts** menu, it searches the current folder and its installation folder for all *. CMD files that contain '///' in the first line (remember that '//' denotes the beginning of the single-line comment) and inserts the text following '///' into the **Scripts** menu.

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

Button	Command	Description
	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 **Script files** 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

This menu gives access to the help system. See also, How to Get On-line Help.

Command	Description
Contents	Opens the contents of the help file.
Search for Help on	Opens the dialog for searching the tool's help system for the content, index and keywords.
Phyton Adapters	Opens the HTML file, which includes adapters' part numbers, their short descriptions and wiring diagrams.
Visit Phyton website	Open the **www.phyton.com** site in your default Internet browser.
Check for updates	Opens the **Update Checking** dialog that directly links your computer to the Phyton download webpage.
Send e-mail message to Phyton	Opens the default email client to compose a message to Phyton.
About ChipProg	The box displays: the ChipProgUSB and the ChipProg Windows shell software version numbers; the selected target device type and the device manufacturer.

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 tab

The Option tab

The Statistics tab

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.commands while

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).
Read	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.
Execute	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	In this field the ChipProgUSB displays the current operation progress bar and the operation status (OK, failed, etc.).

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.
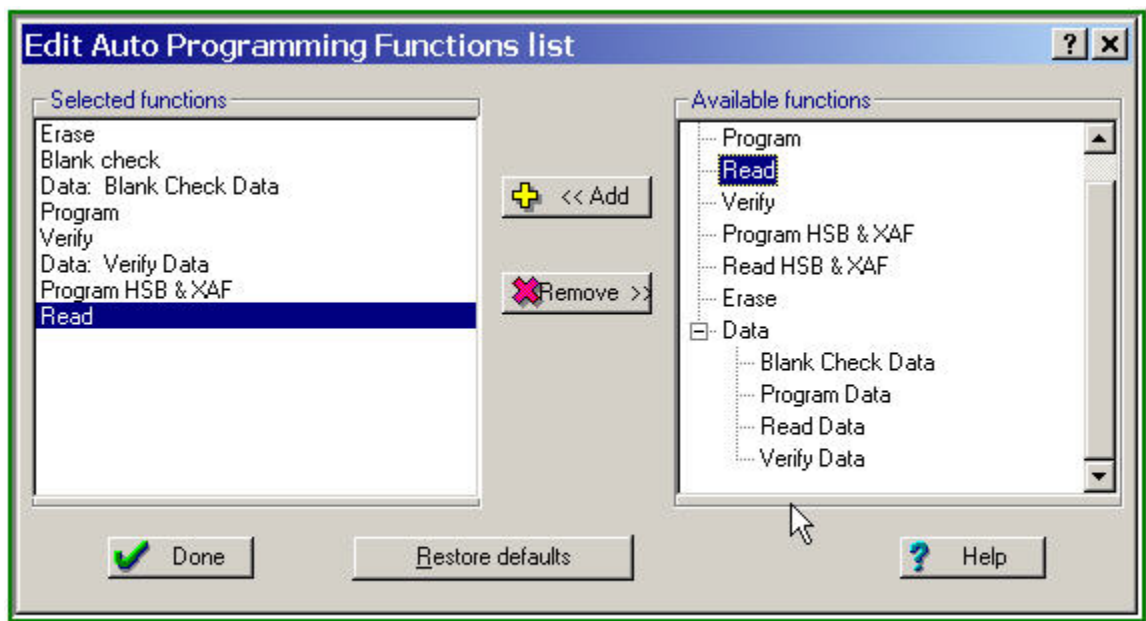
Adding a Command to the Auto Programming Function List

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**

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**. 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=4, etc.
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**). Getting such statistics is quite helpful when you need to program a series of same type devices. It is important to remember that the statistical counters are affected by executing the Auto Programming only, as execution of other functions makes no effect on the statistics.

Element of dialog	Description
Clear statistics **Device Programming Countdown**	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 will count the number of the programmed devices down.
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.

4.4.2   The Device and Algorithm Parameters window

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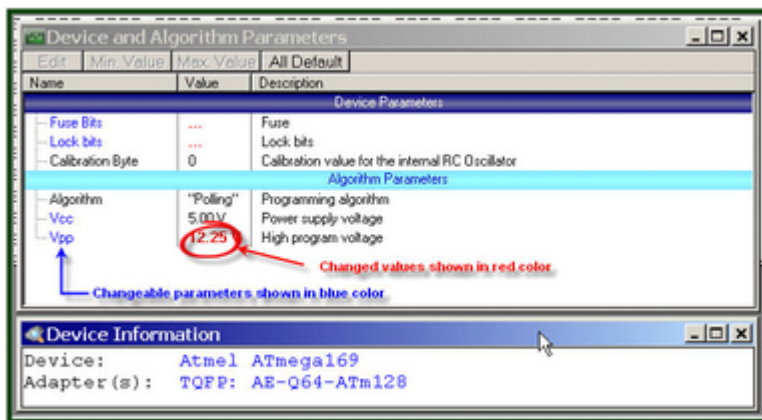
Device and Algorithm parameters

The parameters displayed into this window are split in two groups: **Device Parameters** 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** window's toolbar includes a few buttons positioned on the top of the window:

Toolbar button	Description
Edit	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.
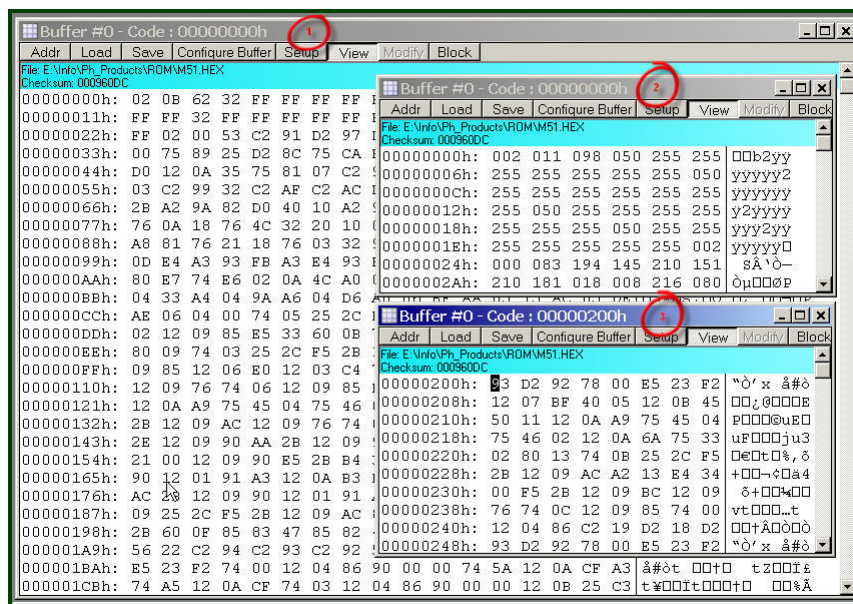
4.4.3   Buffer Dump Window

The **Buffer Dump** window displays the contents of the memory buffer.

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

4.4.3.1 The 'Configuring a Buffer' dialog

The dialog allows configuring the buffer dumps in the most convenient format and name/rename open buffers. 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.
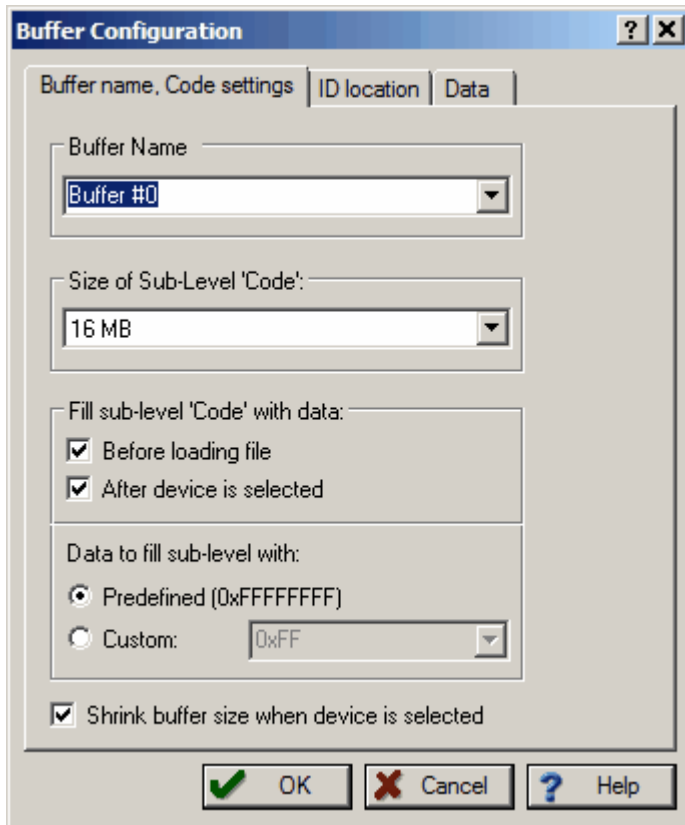
By default each buffer has a minimal size of 128K RAM in a PC and by default the ChipProgUSB program fills the buffer with a predefined value (usually 0FFh). You can customize these buffer settings - check the Custom radio button and type in the pattern to fill the buffer.

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	Is represented by three radio buttons. Here you can select one of the formats for the data displayed: binary, decimal or hexadecimal.
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 **0 00...0 20** and **0 80...0 FF** are non-printable. The options here customize presentations of non-printable ASCII characters in the ASCII pane of the buffer dump window.
Replace characters 0 00...0 20	If the box is checked then all the characters belonging to the range **0 00...0 20** 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	If the box is checked then all the characters belonging to the range **0 80...0 FF** will be replaced with the character dot ('.') or space (' '). The

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 **Buffer Dump** window. The dialog can be invoked only when 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.
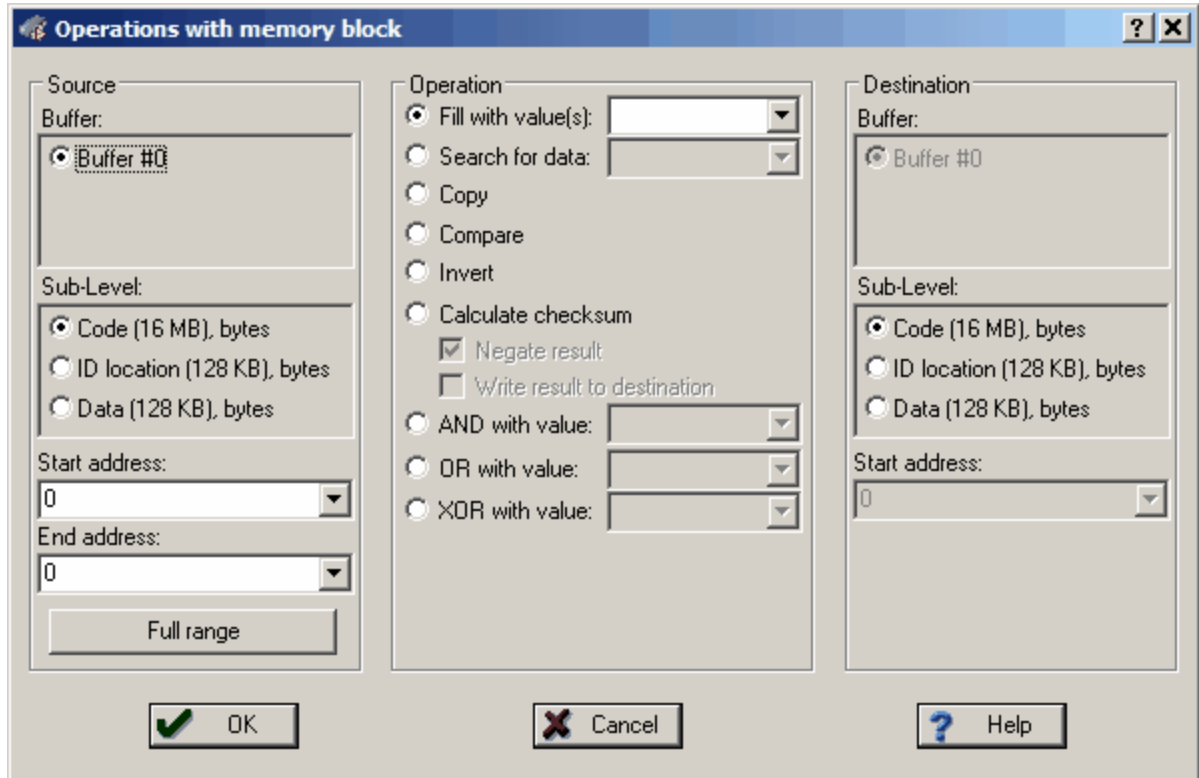
The dialog box splits in three columns. The **Source** parameters, shown in the left column, specify the source memory area for the operations shown in the middle column. The operation's result will be placed in the area specified by the **Destination** shown in the right column. By default the destination 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	Description
Start Address (of the Source)	The start address of the memory area in the selected **Source buffer**, to which the operation will be applied.
End Address (of the Source)	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.

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:	The **format** of the file to be loaded can be selected here by checking one of the radio buttons in the **File Format** 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:	Files in **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 loader supports both Standard and Extended Intel HEX format.
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).
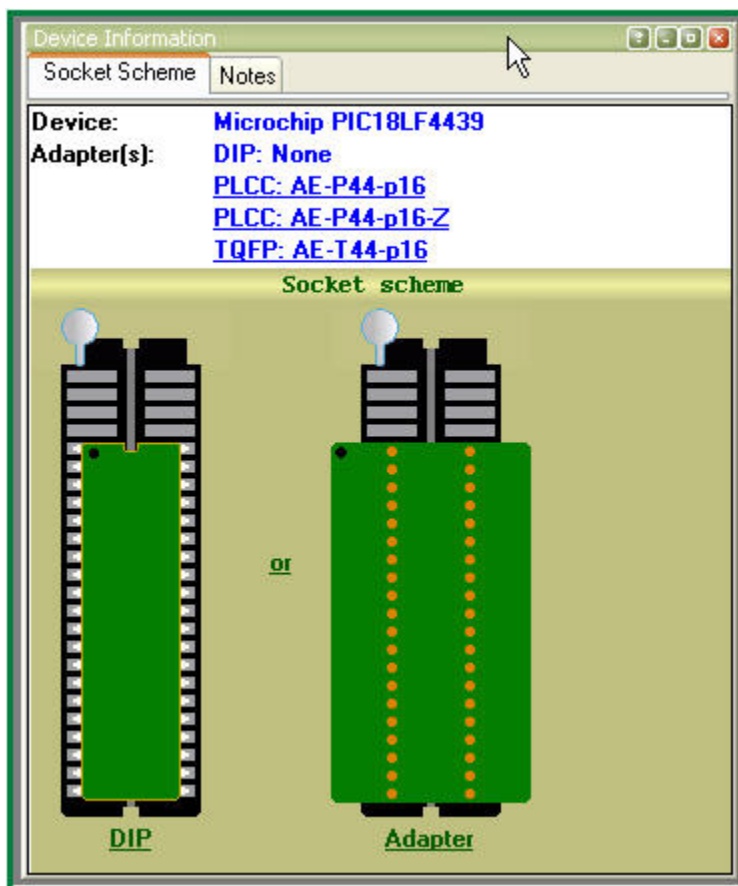
The adapter part numbers are linkable and the links being clicked opens the **adapters.chm** file with a description and wiring diagram of the chosen adapter. The cable adapters for in-system programming are also included into the **adapters.chm** file. There are some peculiarities that such ISP adapters 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			
5	29			
6	9			
7	10			
8	11			
9	12			
10	40			
11	7			
12	13			
13	14			
14	16			
15	17			
16	18			
17	19			
18	20			
19	21			
20	22,30,42	7	1	1
21	24			
22	25			
23	26			
24	27			
25	8			
26	31			

27	32			
28	33			
29	34			
30	5			
31	36			
32	35,3,15,23	14	2	
33	37			
34	38			
35	39			
36	40			
37	41			
38		11		
-	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

The Console window displays messages generated by the ChipProgUSB program that can be divided into two groups: the ChipProg error messages and what-to-do prompts. The window stores messages even if it is closed. You can open it at any time to view the last 256 messages, and get help for any of them. The error messages are shown in red color, others in black.

The window should be large enough to watch several messages. To save screen space you can close the **Console** redirecting all messages to the popping-up message boxes. To do this, go to the **Configure menu** > **Environment** > **Misc** tab and select the **Always Display Message Box** option. Alternatively you can select the **Do not open box if Console window opened** option, redirecting all the messages to the **Console** window.

Click the **Help** button in the box or to invoke the ChipProg context-sensitive **Help** topic associated with the error, or click the **Close** button and continue after correcting a parameter error.

Local menu and Toolbar

The local menu, which can be opened by the right mouse click, includes the **Console** 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
Clear Window	**Clear**	Deletes all the messages from the window
Help on message	**MHelp**	Opens the context-sensitive **Help** topic associated with the error or information in the highligted message
Help on window	No button	Opens the **Console** window **Help** topic
Help on word under cursor	No button	Opens the context-sensitive **Help** topic associated with the word appointed by the cursor

4.4.6   Windows for Scripts

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

- **(Script) Editor** windows
- **Watches** windows
- **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 ChipProg 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 ChipProg-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 ChipProgs. The **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 ChipProg-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
   c) Select the **Blank check** 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 **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, that you want to write to the device;
- edit the file (if necessary);
- configure the device to be programmed (if necessary);
- write the prepared information into the device and verify the programming.

5.3.3.1 How to load a file into a buffer

1. Select the **Main menu** > **File** > **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. If you need to modify source data before writing into the target device, then open the **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** > **Program**.

3. 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.

5. **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 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. Continue until every parameter that was changed in the **Device and Algorithm Parameters window** is successfully programmed.

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 **Program Manager** 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 wait for the message **Verifying … OK** in the **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   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-programming** or **Gang-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. Only the same type of programmers can be used in this mode - either 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 ChipProg-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** shortcut in the 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):
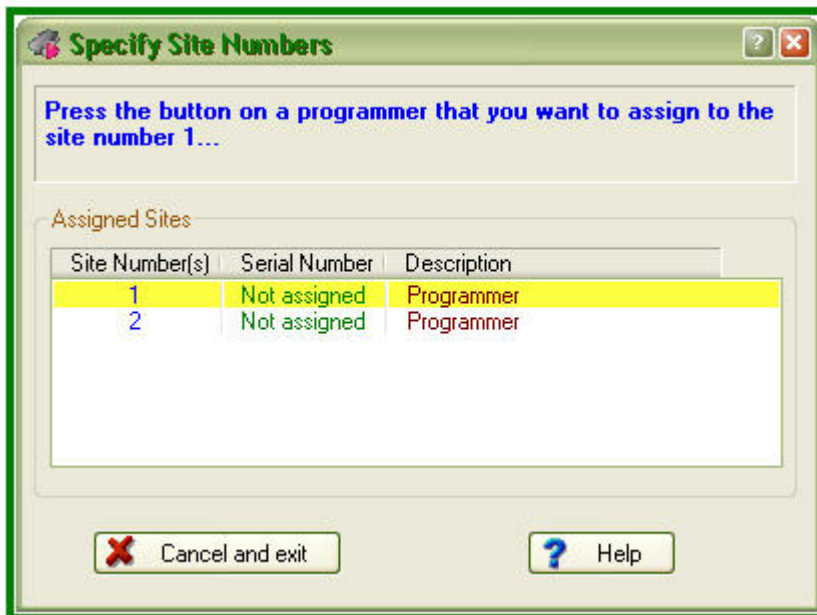
Now you should press the **Start** button on the programmer to which you would like to assign the site #1. Then the ChipProgUSB will prompt to assign the site #2 to another programmer (in case there are more than two programmers in the programming cluster), etc. After assigning numbers to the programmers you will get the **Program Manager** window that differs from the same window that you get when you work with one programmer.

5.4.1   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 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 tab

The **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 ChipProg-48, ChipProg-40 and ChipProg-ISP programmers when they are configured to work in the multi-programming mode. See:

5.4.1.1 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 statisticsand information messages generated by the ChipProgUSB program.
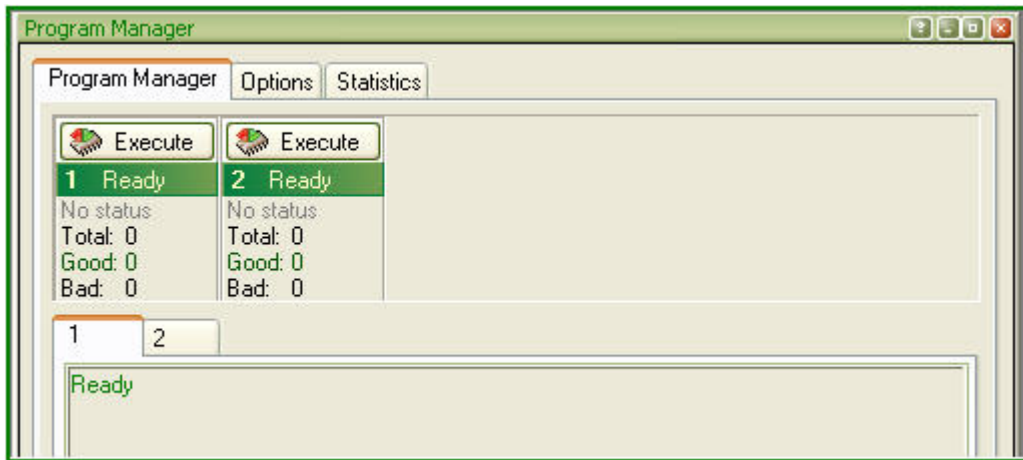
5.4.1.2 The Options tab

The tab serves for setting all programming parameters and options for **multi-programming** mode.
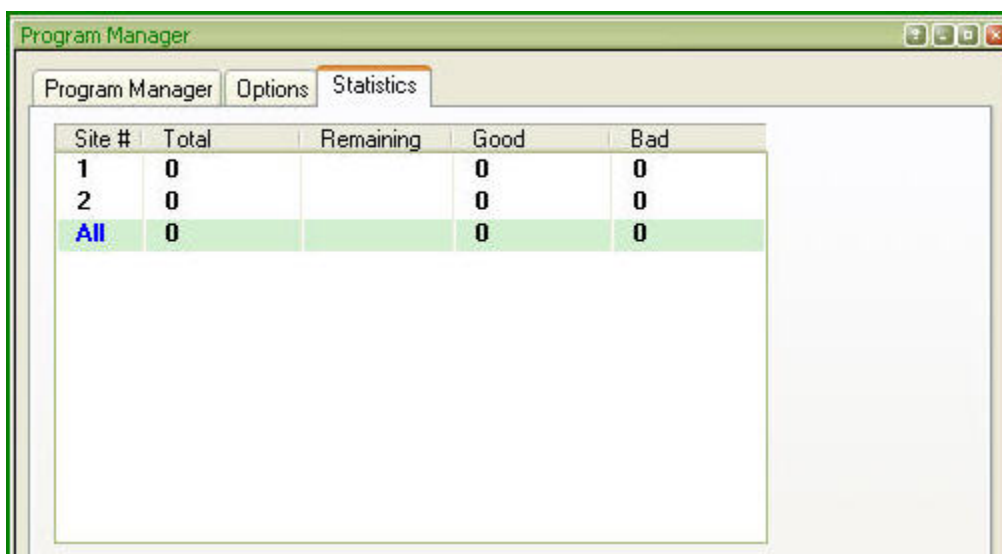
Element of dialog	Description
Buffer:	The field **Buffer** displays the active buffer to which the programming operations (functions) will be applied. A full list of open buffers is available here via the drop-down menu.
Addresses	Here you can set the addresses for the buffer and the target device to 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.
Options:	

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.
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.
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** and **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 ChipProgUSB will always verify the device content right after it has been read out.

5.4.1.3 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 Device Programming Countdown	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 will count the number of the programmed devices down.
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.

5.5   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	There are two alternative options for powering the targets:
	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 should not be connected with the target. The target system should be tolerant to applying logical signals with the voltage levels exceeding the voltages on the target.
	NOTE! 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 - 5mA.
	Max current load for the Vcc line - 80 .
	Max current load for the Vpp line - 80 .

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 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 programmers. By means of script files you can automate loading files to the programmer buffers, calculating checksum, launching device programming, pausing programming in case of an error, manipulating windows and performing many other operations. It is also possible to display various messages in the **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 **Script Files dialog.**

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

6.1   The Script Files Dialog

This dialog is used for controlling the **Script Files**, 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 **Unload Request** 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

6.2   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.
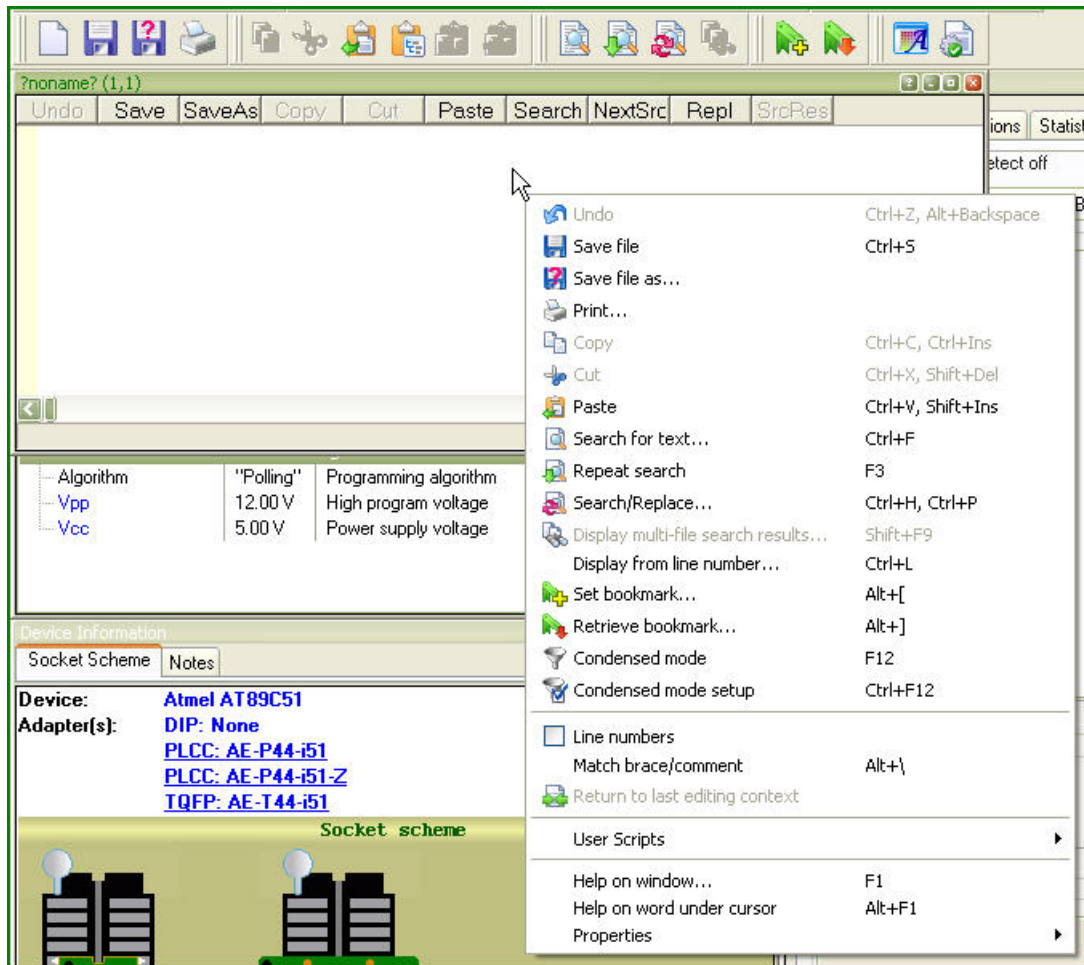
Now you can compose your script right in the window.

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.

6.2.1   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.
	Save As...	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.

6.2.2   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	Copies and appends the marked block of text to the block in the 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 to set a local bookmark.
	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** dialog.
	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.

6.2.2.1 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.

Element of dialog	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 **Source** window only.
Search All Source Files in Project	If this box is checked the editor will search in all the source files included in 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** window, then only the window buffer will be searched, not the file on disk.

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

6.2.2.2 The Replace Text Dialog

This dialog sets the parameters for the search-and-replace operation. This dialog and the **Search for 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.

Element of dialog	Description
Text to Search for	Specifies the text string to look for (search string).
Replace with	Specifies the text string to replace the found one.
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.

Prompt at Replace	This box is checked by default and if it is checked the editor will always pop up the **Confirm Replace** dialog requiring your permission to replace the found text. If unchecked the editor will automatically replace the searched-and found text.
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 and Replace	This box is checked by default and if it 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 **Source** window only.
Search All Source Files in Project	If this box is checked the editor will search in all the source files included in 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, which are 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** window, then only the window buffer will be searched, not the file on disk.

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

6.2.2.3   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 **Replace Text** 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.
Skip this File	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.

6.2.2.4 The Multi-File Search Results Dialog

This dialog displays the multi-file search results. To learn about the multi-file search, see the **Search for Text** dialog.

The **List of Matched Files** shows the files where the search string is found. The file name is on the left and its directory is on the right. The line with green text right under this box displays information about the file selected in the box. "File in memory" means that the file is opened in the **Source** window. General information from FAT means the file is on disk, not loaded. The **Preview** area shows the source line with the found text string.

The **Sort Files by** area includes a radio button with four file sorting options. When the **Consider Directory** box is checked, the files are sorted with respect to their directories.

The **Edit** button opens the selected file in the new **Source** window and places the cursor on the line with the found string. The found string is marked with the background color. To check if there are other occurrences of the sought string in this file, press **Ctrl+R** or use the **Next Search** command of the **Edit** menu.

The **Close** button closes the dialog but the results are not lost. To reopen the dialog use the **Display Multi-file Search Results** button. You can also use the same command of the **Edit** menu or press **Shift+F5**. The files in the **List of Matched Files** box, which are opened in the **Source** window, will be marked with asterisks on the left.

6.2.2.5 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: %Counter - 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	The hexadecimal value of the character. Example: \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] means any letter from A to 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."

6.2.2.6   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 **Alt+[**. To open the **Retrieve Bookmark** 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.

6.2.2.7   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.

6.2.2.8   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:
- **Equal to** - 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.

6.2.2.9   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** to accept the editor's completion suggestion.

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** box defines the number of lines for the editor to scan. The default is 24 lines below and 24 lines above the current line. When this parameter is greater than the total number of lines in the file (for example, 65535), then program composing will become slower because the whole file will be scanned.

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

How to control

You can disable syntax highlighting through the **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.**

6.2.2.11 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.

6.2.2.12 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.

6.2.2.13 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+F3**) or the block is deleted (**Ctrl+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** (fast move the block):

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

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 **Source** 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

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.

6.3 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. The window is split in two panes: the left pane displays the script text, the 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 **Debug mode**, stop the script execution and open the window with the script file.
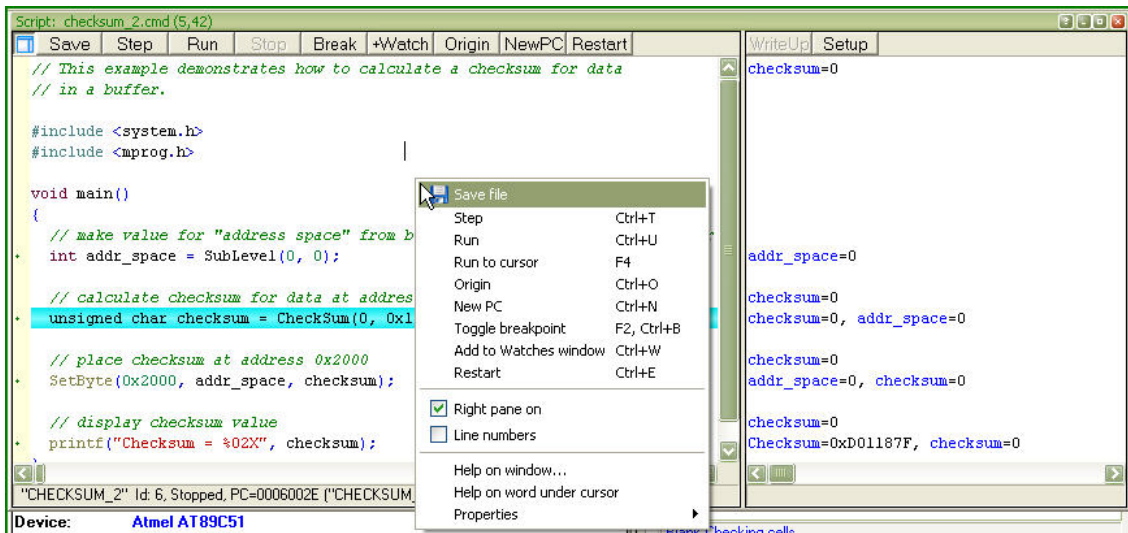
Syntax constructions and the lines, which correspond to the current PC value (blue strip) and the breakpoints (red strips), are highlighted in the script file text (for more information, see Syntax Highlighting).

Local menu and toolbar

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

Command	Window Toolbar	Description
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		Executes the script up to the line where the caret is positioned (the corresponding address). Alternatively, you can double-click the line to carry out this command.

6.3.1   The AutoWatches Pane

The ChipProgUSB program displays a visible portion of the script in the **Script** window. The names of variables, called **AutoWatches,** 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.

The **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.

6.3.2   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	Deletes a selected object from the **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.

6.3.2.1 The Display Watches Options Dialog

Use this dialog to set the display options for the selected variable or expression 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.

6.3.2.2 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.

6.3.3   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;
- responding to the events displaying in the **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.

6.3.4   The I/O Stream Window

The **I/O Stream window** is a window that can be created by means of the built-in **OpenUserWindow** function executed from the script itself. Script files use windows of this type to display I/O streams in the form of text. The most usual examples of I/O streams are displaying the characters inputted from the PC keyboard and text messages outputting by the scripts. Also, you can reassign I/O streams to files and input data from files.

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** button clears the window contents.

Windows of this type do not have a local menu.

7 References

7.1 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 configuration** file, the name of which is specified in the <> brackets. This **Session 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 **UPROG.opt** 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 ChipProg 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	Starts the 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.

7.2 Errors Messages

Enter topic text here.

7.2.1 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."

7.2.2 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"

4019 "Address in device: 0x%08X, Address in buffer: 0x%08X\n"

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."

7.2.3 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'"

.

7.2.4   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"

5334 "/%s command-line option: Invalid address value"

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

7.2.5   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."

7.2.6   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'"

7.2.7   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."

7.2.8   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"

7.2.9   Error internal

6527   "Internal error:\nCORE() for %s %s returned NULL.\nPlease contact your %s distributor."

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

7.2.10   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')."

7.2.11   Error device

5326   "Device selection error"

4018   "Device '%s' is not supported by the %s. Please choose another device."

7.2.12 Error check box

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

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

7.2.13 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"

7.2.14 Warning

5338 "Warning: JEDEC file has no file CRC"

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."

7.3 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**, apply the **&** (address) operation, that is, `&var`.

7.3.1 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)
*	Multiplication
/	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.

7.3.2   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`; `111111111111111000011B`

5) Symbol (ASCII).

Examples: `'a'`; `'ab'`; `'$B%8'.'`.

7.3.3   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)

7.4 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

Script Language Built-in Variables

Difference Between the Script Language and the C Language

Alphabetical List of Script Language Built-in Functions and Variables

7.4.1 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
  InsertTest = TRUE;                                            // set testing of chip presence to "on"
  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
}
```

7.4.2 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
Directives of the Script File Language Preprocessor
Predefined Symbols in the Script File Compilation

7.4.3 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:

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

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
Mathematical Functions
String Operation Functions
Character Operation Functions
Functions for File and Directory Operation
Stream File Functions
Formatted Input-Output 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**.

7.4.4 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
ChipStartAddr
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

7.4.5   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);
void SetElement(int array[], int index1, int index2, int value);
```

- 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.
- Preprocessor macros, such as **#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()
{
  ...
}
```

7.4.6 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
ApplName[]
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

7.5   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

7.5.1   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).

7.5.2   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 cable adapter **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.
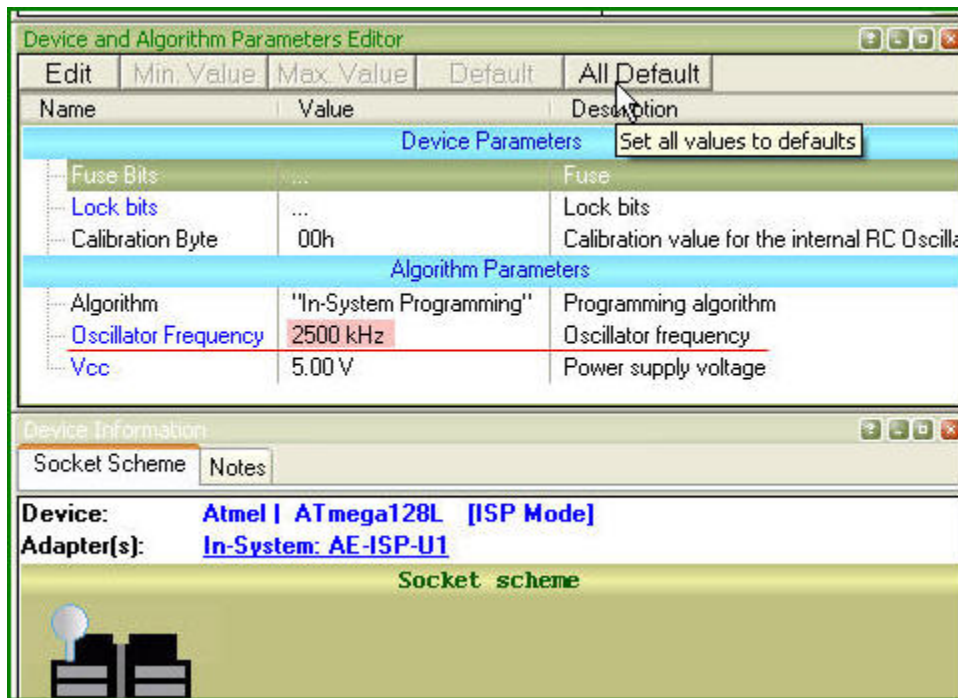
2. If the target system **does not have its own built-in clock generator** then, the target AVR device needs to get clocks from the ChipProg built-in generator; thus the clock output wire of the cable-adapter should be connected to an appropriate clock input pin of the target device. By default the Fclk= 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).

7.5.3   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 -

- Q -

- R -

- S -

Back Cover