ICP Family Programmers: DLL Description

IMPORTANT NOTE:

- Starting from Sep-2016 Softlog Systems manufactures ICP2**(G3)**, ICP2-GANG**(G3)** and ICP2-COMBO**(G3)** programmers additionally to existing ICP2, ICP2-GANG and ICP2-COMBO
- Starting from Jul-2018 Softlog Systems manufactures ICP2-Portable**(G3)** programmer additionally to existing ICP2-Portable
- Due to nearly full compatibility all of them are referred below as ICP2, ICP2-GANG, ICP2-COMBO and ICP2-Portable respectively. If difference is applied then they are referred as “G3 products” and “non-G3 products”

1 Installation

1.1 Important Note

IMPORTANT: recompile your application every time you use new ICP DLL version

1.2 Latest Software

Visit our site and get the latest software: http://www.softlog.com → Support

1.3 Install "ICP for Windows"

Run "ICP for Windows" setup file "IcpWin_setup_xxx.exe".
This manual presumes that you are familiar with "ICP for Windows" software

1.4 Install "ICP DLL"

Run DLL setup file "IcpDll_setup_dll_xxx.exe"

The following files will be installed to the default directory "C:\Softlog\IcpDll":

###	*File Name*	*Description*
1.	IcpDll.dll	ICP DLL
2.	IcpWinComLine.exe	ICP Command line utility. See document "ICP Command Line.pdf" for details
3.	c_icpexp.h	Header file with ICP DLL functions
4.	fr_exp.h	Header file with ICP DLL definitions
5.	fr_icp2.h	Header file with ICP2 hardware definitions
6.	DLL Description.pdf	This document
7.	ICP Command Line.pdf	ICP Command line utility description
8.	ICP2 - FTM Functions.pdf	Description of ICP FTM (final test machine) functions
9.	\Lib_Borland\IcpDll.lib	LIB file for Borland C Builder
10.	\Lib_Microsoft\IcpDll.lib	LIB file for Microsoft C++
11.	\VB_Microsoft\c_icpexp.bas	Header file with ICP DLL functions for Microsoft Visual Basic

2 ICP Firmware Options

DLL/Command Line Support (D) should present in order to use DLL functions. Order Softlog programmers with "D" suffix, for example ICP2-D or ICP2GANG-D or activate "D" support remotely. For more details see page 2 of http://softlog.com/userfiles/file/Downloads/ICP2(G3)%20Family%20Programmers.pdf or contact Softlog Systems

Use the following sequence to validate your ICP firmware options:
- connect the programmer unit to the PC
- run "ICP for Windows"
- if programmer is connected and communication is OK then press "Details" button on "About" screen
- if incorrect COM is selected then open "Communication→RS-232/USB/LAN COM", select the detected COM and press OK, then press "Details" as specified above

3 General Sequence of Operations

Step	*Function*	*Description*	*Usage*
1.	IcpStartApplication()	Starts ICP application	Should be called once before other ICP DLL functions can be called, **mandatory**. Next call can be only done after application is closed by IcpEndApplication()
2.	IcpInitCom()	Initializes COM port	Once, **mandatory** if communication is required. Next call can be done after COM is released by IcpReleaseCom()
3.	IcpLoadHexAndSerFile()	Loads hex and/or serialization files	Once or repeated, **mandatory in PC-driven** mode
4.	IcpDoAction()	Executes action according to ACTION_LIST	Once or repeated, not mandatory
5.	IcpReleaseCom()	Releases COM port	Once or repeated, not mandatory
6.	IcpEndApplication()	Closes the application	Once, **mandatory**

4 Configuration File (*.cfg)

Function IcpStartApplication() uses ICP configuration file ("icp01.cfg") as a parameter. The following procedure is recommended for preparing "icp01.cfg: file:

- Run "ICP for Windows" software
- Select programmer: "Programmer → Select Programmer"
- Select COM: "Communication → RS-232/USB/LAN COM"
- Select device to be programmed: "Device → Select by Name"
- Set desired voltages, etc.: "Options → Voltages"
- Save configuration file: "File → Save Configuration"
- Validate that interactive programming works correctly
- Exit the software
- Copy "icp01.cfg" to your project directory and rename. NOTE: "icp01.cfg" is usually located in directory "C:\Softlog\IcpWin"

  Note: 64-bit Windows OS sometimes places icp01.cfg and other "ini" files from "C:\Program Files" into a virtual storage location, for example:

  C:\Users\<Your name>\AppData\Local\VirtualStore\Program Files (x86)\Soft-Log\IcpWin

5 Quick Start

- Install ICP DLL as described in chapter 1 to directory "C:\Softlog\IcpDll"
- Prepare "icp01.cfg" file as described in chapter 4
- Create project directory, for example **"C:\MyProject"**
- Run Visual C++ and create a new project, for example **Project1**

- Copy ICP DLL files from "C:\Softlog\IcpDll" as follows:
  - to **"C:\MyProject"**:
    - "c_icpexp.h"
    - "fr_exp.h"
    - "IcpDll.lib" (from "C:\Softlog\IcpDll\Lib_Microsoft")
  - to **"C:\MyProject\Debug"**:
    - Icp01.cfg
    - IcpDll.dll
- Define "IcpDll.lib" as a part of the project:
  - Visual Studio C++ 6.0:
    - select menu item “Project->Settings”
    - select tab Link
    - select mode “All Configurations” in combo box “Settings For”
    - fill field “Object/library modules” with string “IcpDll.lib”
    - press OK
  - Visual Studio C++ 2008:
    - select menu item “Project->Properties”
    - select tree items “Linker->Input”
    - select mode “All Configurations” in combo box “Configuration”
    - fill field “Additional Dependencies” with string “IcpDll.lib”
    - press OK
- Write a simple PC-driven application – see example in chapter 17.1
- Select **Debug** mode, compile and run the application
- After debugging is done copy files "Icp01.cfg" and "IcpDll.dll" from directory **"C:\MyProject\Debug"** to **"C:\MyProject\Release"**
- Select **Release** mode, compile and run the application

6 Base Functions

6.1 IcpStartApplication

Description: Starts application and loads configuration file
Prototype: int DLL_FUNC IcpStartApplication (char *aFileCfg)
Parameters: aFileCfg - ICP configuration file to be loaded, usually "icp01.cfg"
Example: int Stat = IcpStartApplication ("c:\MyProject\Icp01.cfg");

6.2 IcpInitCom

Description: Initializes RS-232/USB/LAN COM port
Prototype: int DLL_FUNC IcpInitCom (int aOverCfg, int aComPort, int aBaudRate)
Parameters: aOverCfg: 0-gets communication parameters from *.cfg file
1-overrides *.cfg file with aPort and aBaudRate
aPort: 0-COM1, 1-COM2,...
aBaudRate: see enum BAUD_RATES (file fr_exp.h)
Example: Stat = IcpInitCom(0, 0, 0); //use settings from *.cfg file

6.3 IcpLoadHexAndSerFile

Description: Loads HEX and serialization files
Prototype: int DLL_FUNC IcpLoadHexAndSerFile (char *aFileHex, char *aFileSer)
Parameters: aFileHex - pointer to char string that contains HEX file name
aFileSer - pointer to char string that contains serialization file name

Example 1: Stat = IcpLoadHexAndSerFile ("c:\MyProject\1.hex", "c:\MyProject\1.ser");

Example 2: Stat = IcpLoadHexAndSerFile ("c:\MyProject\1.hex", ""); //hex file only

Example 3: Stat = IcpLoadHexAndSerFile ("", "c:\MyProject\1.ser"); //serialization file
//only (standalone operation)

6.4 IcpTestConnection

Description: Description: communicates with the programmer as follows:
- if ICP2 then with ICP2
- if GANG/COMBO then with channel 2
- if GANG/COMBO(single channel) then with the selected channel
No LEDs are activated on the programmer (blind connection)

Prototype: int DLL_FUNC IcpTestConnection(void)

Parameters: None

Example:
```
int err= IcpTestConnection();
if (err != AUTO_OK) //(see errorcodes in chapter 0)
  <means communication error>
```

6.5 IcpDoAction

Description: Executes programming, verification and other actions specified in enum ACTION_LIST – see 16.2.1

Prototype:
```
int DLL_FUNC  IcpDoAction(   int aAction,
                             unsigned int aMemorySpace,
                             unsigned int aPmUserRange,
                             unsigned int aPmAddrBeg,
                             unsigned int aPmAddrEnd,
                             unsigned int aSaveResult,
                             char* aReadFile  );
```

Parameters: aAction: one of values of ACTION_LIST. Note: ICP software automatically removes memory spaces which do not exist in the selected device

aMemorySpace: sum of memory spaces - see MEMORY_SPACES in 16.2.2

aPmUserRange:	0-use full PM range, 1-override with aPmAddrBeg and aPmAddrEnd
aPmAddrBeg:	start address of PM (if aPmUserRange=1)
aPmAddrEnd:	end address of PM (if aPmUserRange=1)
aSaveResult:	1-operation result will be written to file "auto01.res"
aReadFile:	hex file to be saved after read (if aAction=ACT_READ)

Example 1 (PC-driven mode): Program (and verify) entire chip

```
int Stat = IcpDoAction(  ACT_PROG,          //aAction
                         ALL_SPACE,         //aMemorySpace
                         0,                 //aPmUserRange
                         0,                 //aPmAddrBeg
                         0,                 //aPmAddrEnd
                         0,                 //aSaveResult
                         “”);               //aReadFile
```

Example 2 (PC-driven mode): Verify locations 0x0020-0x003F of PM

```
Stat = IcpDoAction(      ACT_VER,           //aAction
                         PM_SPACE,          //aMemorySpace
                         1,                 //aPmUserRange
```

```
                    0x0020,             //aPmAddrBeg
                    0x003F,             //aPmAddrEnd
                    0,                  //aSaveResult
                    “”);                //aReadFile
```

Example 3 (PC-driven mode): Read entire chip to file "c:\prj\hex1.hex"

```
Stat = IcpDoAction(  ACT_READ,          //aAction
                     ALL_SPACE,         //aMemorySpace
                     0,                 //aPmUserRange
                     0,                 //aPmAddrBeg
                     0,                 //aPmAddrEnd
                     0,                 //aSaveResult
                     "c:\prj\hex1.hex"); //aReadFile
```

Example 4 (Standalone mode): Program (and verify) all chip(s)

```
int Stat = IcpDoAction(  ACT_STA_PROG,  //aAction
                         ALL_SPACE,     //aMemorySpace (Note 1)
                         0,             //aPmUserRange
                         0,             //aPmAddrBeg
                         0,             //aPmAddrEnd
                         0,             //aSaveResult
                         “”);           //aReadFile
```

Note 1: aMemorySpace parameter does not affect standalone operation since memory space is selected according to environment settings saved in ICP2 internal flash memory

6.6 IcpReleaseCom

Description: Releases serial communication port
Prototype: int DLL_FUNC IcpReleaseCom (void)
Parameters: None

Example: Stat = IcpReleaseCom();

6.7 IcpEndApplication

Description: Terminates ICP DLL application
Prototype: int DLL_FUNC IcpEndApplication (void)
Parameters: None

Example: Stat = IcpEndApplication();

6.8 IcpReadDllVersion

Description: Gets DLL version string
Prototype: int DLL_FUNC IcpReadDllVersion (char *dllSoftwareVer)
Parameters: dllSoftwareVer – pointer to DLL version string

Example:
```
char DllVersion[80];
Stat = IcpReadDllVersion (DllVersion);
```

6.9 IcpEnableProgressWindow

Description: Enables/disables progress window (progress bar)
Prototype: int DLL_FUNC IcpEnableProgressWindow(int aEnable);
Parameters: aEnable: 0-disable, 1-enable

6.10 IcpDllSelectProg

Description: Selects programmer type and mode by overwriting value from *.cfg file,
useful to switch between ICP2-GANG/COMBO channels and standalone/PC-driven modes

Prototype: int DLL_FUNC IcpDllSelectProg(int aProg, int aMode, int aCh)
Parameters: aProg - programmer according to enum PROG_TYPE – see 16.2.3
aMode - mode of operation according to enum PROG_MODE – see 16.2.3
aCh - current channel for "GANG/COMBO-single channel" programmer, 0 for channel 1
Return Value: A) AUTO_OK if parameters are correct
B) -1: parameter 1 is not correct
-2: parameter 2 is not correct
-3: parameter 3 is not correct

Example: Select channel 2 of ICP2-GANG/COMBO programmer for PC-driven operation,
then restore to GANG

```
IcpDllSelectProg(PROG_GANG4_SINGLECHAN, SPM_PCDRIVEN,1);
... - execute desired PC-driven operations with channel 2
IcpDllSelectProg(PROG_GANG4, SPM_STANDALONE, 0); //restore GANG standalone
```

6.11 IcpCalcPmRange

Description: Calculates best of full PM range

Prototype:

```
int DLL_FUNC IcpCalcPmRange(unsigned int *aStartAddr,
                            unsigned int *aEndAddr,
                            int aWhichRange );
```

Parameters: aStartAddr-pointer to start address
aEndAddr-pointer to end address
aWhichRange: 0-get full range, 1-get best range

Return Value: 0-OK, -1: parameter is not correct

Example: Calculate optimum (best) PM range and execute programming

```
unsigned int start_addr, end_addr;
IcpCalcPmRange (&start_addr, &end_addr, 1); //get the best range

Stat = IcpDoAction (ACT_PROG,       //aAction
                    ALL_SPACE,      //aMemorySpace
                    1,              //aPmUserRange
                    start_addr,     //aPmAddrBeg
                    end_addr,       //aPmAddrEnd
                    0,              //aSaveResult
                    "");            //aReadFile
```

6.12 IcpSingleWordRead (read Device ID or special area)

Description: Reads Device ID or data from special areas

Prototype:

```
int DLL_FUNC IcpSingleWordRead (    int aLoadConfig,
                                    int aAddrOrIncValue,
                                    unsigned short *aWord);
```

Parameters: aLoadConfig: 1-access configuration memory, 0-access PM.
see “c_icp_exp.h” for more details
aIncValue: address or increment (in words) from beginning of the accessed area
see “c_icp_exp.h” for more details
aWord: read word (Device ID)

Example:
```
unsigned short dev_id;
Stat = IcpSingleWordRead(1, 6, &dev_id); //for all PIC12F/16F families

Stat = IcpSingleWordRead(0, 0x3FFFFE, &dev_id); //for all PIC18F families

Stat = IcpSingleWordRead(0, 0xFF0000, &dev_id); //PIC24/dsPIC33: DEVID
Stat = IcpSingleWordRead(0, 0xFF0002, &dev_id); //PIC24/dsPIC33: DEVREV
```

6.13 IcpSingleWordWrite (write special area)

See “c_icp_exp.h” for details (device dependent)

6.14 IcpSetChipEraseBeforeProg

Description: Enables/disables bulk erase before programming (overrides settings from *.CFG file)

Prototype: void DLL_FUNC IcpSetChipEraseBeforeProg(int aEraseOn)

Parameters: 1-enable bulk erase, 0-disable bulk erase

WARNING: bulk erase must be enabled for most of devices to keep endurance

6.15 IcpSetRowEraseBeforeProg

Description: Enables/disables row/block erase before programming (overrides settings from *.CFG file)

Prototype: void DLL_FUNC IcpSetRowEraseBeforeProg(int aEraseOn)

Parameters: 1-enable row/block erase, 0-disable row/block erase

NOTE: Row erase is supported for limited devices - contact Softlog Systems for details

6.16 IcpSetHardEraseBeforeProg

Description: Enables/disables smart hardware erase (T_DIO_2 pin) before programming (overrides settings from *.CFG file)

Prototype: void DLL_FUNC IcpSetHardEraseBeforeProg(int aEraseOn)

Parameters: 1-enable hardware erase, 0-disable hardware erase

Note: Hardware erase is supported for limited devices (SAM E/S/V) - contact Softlog Systems for details

7 Gap Eliminator™

In addition to the critical data they carry, HEX files may also contain multiple empty areas (gaps). These gaps may come at the beginning, in the middle, or at the end of the HEX file. Thus, when programming a microcontroller, the empty bytes of a HEX file are also burned onto the microcontroller. Gap Eliminator™ feature solves this problem. Before a production run, it automatically analyzes the HEX file and effectively removes multiple gaps (up to five) from the PM (flash) and DM (EEPROM). This significantly reduces programming time and drives major cost savings for mass production operations

7.1 IcpSetGapElimination

Description: Enables/disables Gap Eliminator™ for PM(flash) or/and DM(EEPROM)

Prototype: int DLL_FUNC IcpSetGapElimination(int aArea, int aEnable);

Parameters: aArea: PM_SPACE, DM_SPACE or PM_SPACE+DM_SPACE
aEnable: 1-enable, 0-disable

Return Value: 0-OK, -1: incorrect area is specified

Example: IcpSetGapElimination(PM_SPACE+DM_SPACE, 1) ; //enable Gap Eliminator™ for //both PM and DM

8 Standalone Operation: General Functions

8.1 Delay between PC-driven and Standalone Modes

IMPORTANT: delay of at least 1000ms (1500ms is recommended) should be added between PC-driven and standalone mode functions - see example 17.2

8.2 IcpDoAction – see 6.5

See IcpDoAction() with parameters ACT_STA_PROG, ACT_STA_GET_RES and ACT_STA_START_PROG

8.3 IcpDllGetInfoSingle

```
enum INFO_ICP_OPTIONS {
  INFO_OPT_DLL        = 0x0001, //1-DLL support enabled
  INFO_OPT_DSPIC      = 0x0002, //1-dsPIC support enabled
  INFO_OPT_KEE        = 0x0004, //1-Keeloq support enabled
  INFO_OPT_SEC        = 0x0008, //1-security feature support enabled
  INFO_OPT_PIC32      = 0x0010  //1-PIC32 support enabled
};

#define INFO_ID_SIZE          3  //firmware ID length
#define INFO_GANG_QUAN    64 //maximum number of GANG/COMBO channels

typedef struct //info from one channel of ICP2/ICP2-GANG/ICP2-COMBO
{
  int iValid;                                 //1-information in this structure is valid
  unsigned short iProgType;                   //programmer type according to PROG_TYPE
  unsigned short iFirmVer;                    //firmware version (H.L)
  unsigned char iIcpId[INFO_ID_SIZE];         //firmware ID (ICP serial number)
  unsigned short iBootVer;                    //bootloader version (H.L)
  unsigned short iIcpOpt;                     //ICP enabled options according to INFO_ICP_OPTIONS above
  unsigned short iFirmDevDb;                  //firmware device database version (H.L.)
  unsigned short iFirmPrjDb;                  //firmware project database version (H.L.)
  unsigned short iDllDevDb;                   //DLL device database version (H.L.)
  unsigned short iDllPrjDb;                   //DLL project database version (H.L.)

  int iSecMode;                               //security mode (1-secure environment now)
  int iEnvStat;                               //environment status according to enum PRJ_VAL – see 16.2.5
  char iEnvHexFileName[SEC_ID_SIZE];//HEX file name inside the environment
  unsigned short iEnvHexFileCs;               //HEX file checksum (ICP2 legacy CS)
  char iSecIdName[SEC_ID_SIZE];               //Security ID name
  int iSecCntValue;                           //value of non-volatile security counter
  int iSecCntInteg;                           //security counter integrity (0-OK)

  //Added 18-Nov-17
  int iProgG3;          //1=G3, 0=non-G3 products

  //Added 28-Feb-18
  unsigned char iG3PcbType;    //G3 programmer: PCB Type
  unsigned char iG3ProductId;  //G3 programmer: Product ID (name)
```

```
  //Total number of ***bytes*** below must be ***128*** for compatibility
  unsigned char iReserved[128]; //reserved for future members

} ICP_INFO;
```

Description: Get environment and ICP general info from a single channel
Prototype: int DLL_FUNC IcpDllGetInfoSingle(ICP_INFO *aInfo)
Parameters: aInfo - pointer to ICP_INFO structure

Example: Validate that environment is OK and HEX file checksum is 0x1234

```
ICP_INFO MyInfo;
IcpDllGetInfoSingle(&MyInfo);
if (! MyInfo.iValid) //invalid environment
  return 1;

if (MyInfo.iEnvHexFileCs != 0x1234)
   return 2;

return 0; //OK
```

Note: Use IcpDllGetIcpInfoToPcTmpStruct() followed by multiple IcpDllReadInfoOneByOne() to avoid using structures – see c_icpexp.h for details

8.4 IcpReadStaResOneCh

Description: Reads result of standalone operation for one channel. Should be called in loop for all channels after previous programming is done
Prototype: int DLL_FUNC IcpReadStaResOneCh(unsigned int aCh);
Parameters: aCh: channel number, range 0...63. NOTE: aCh is 0 for ICP2 (not GANG)
Return Value: A) -1 if channel is not enabled
B) -2 if channel number is out of range (>63)
C) according to AUTO_ERROR_LEVEL

Example: Program 8 channels of ICP2-GANG/ICP2-COMBO and get results

Step 1: Execute standalone programming for 8 channels

```
#define CH_NUM  8  //8 channels
int Res[CH_NUM];
int Stat = IcpDoAction(   ACT_STA_PROG,      //aAction
                          0,                 //aMemorySpace
                          0,                 //aPmUserRange
                          0,                 //aPmAddrBeg
                          0,                 //aPmAddrEnd
                          0,                 //aSaveResult
                          "");               //aReadFile
```

Step 2: Analyze result

```
if (Stat==AUTO_OK)  {
  ; //do nothing, all channels OK
}
else {
  for (int i=0; i<CH_NUM; i++)
    Res[i]= IcpReadStaResOneCh[i]; //save all results
{
```

9 Standalone Operation: Create/Transfer Environment

Simultaneous GANG programming can be done in standalone mode only. An environment should be created and transferred to the programmer unit before programming takes place

An environment can be created and transferred by 3 methods:

- Off-line by using "ICP for Windows" ("Environment → Save Environment As" and "Environment → Transfer Environment to Programmer")
- On-line by using DLL functions below
- Combination of the both methods above, i.e. multiple environments can be created off-line and then transferred by DLL function

IMPORTANT: A transferred environment is saved in **non-volatile** flash memory of the programmer and automatically reloads after power-up. Don't transfer the same environment multiple times to keep endurance of internal ICP family programmer flash (10K cycles minimum, 100K typical)

9.1 IcpSaveEnvironment

See also: IcpTransferEnvironmentToIcp()

Description: Creates and saves current workspace into an environment file (extension *.pj2)
Prototype: int DLL_FUNC IcpSaveEnvironment (const char* aFileName);
Parameters: aFileName - pointer to char string containing file name

Example:
```
IcpStartApplication ("icp01.cfg");
IcpInitCom (0, 0, 0);
IcpLoadHexAndSerFile("1.hex","");
IcpSaveEnvironment ("File1.pj2");
IcpEndApplication();
```

9.2 IcpTransferEnvironmentToIcp

See also: IcpSaveEnvironment()

Description: Transfers an environment file (extension *.pj2) to the programmer
Prototype: int DLL_FUNC IcpTransferEnvironmentToIcp (const char* aFileName);
Parameters: aFileName - pointer to char string containing file name

Example: IcpTransferEnvironmentToIcp ("File1.pj2");

10 Standalone Operation: RAM Buffer

Every ICP2/ICP2-GANG/ICP2-COMBO channel unit has a volatile RAM buffer (256 bytes) which can be used for different purposes, for example to overwrite non-volatile contents of the environment

WARNING: be careful with RAM buffer manipulation since it changes your programming data

10.1 IcpRamUsage

See also: IcpRamBufWrite() and IcpRamBufRead()

Description: Communicates with ICP2 and defines functionality of the RAM buffer
Prototype: int DLL_FUNC IcpRamUsage(int aPmStartAddr, int aPmQuan, int aDmStartAddr, int aDmQuan, int aUsage)
Parameters: aPmStartAddr - start address (**in bytes**) of PM to be overwritten by the RAM buffer
aPmQuan - length of overwritten block (PM) **in bytes**
aDmStartAddr - start address (**in bytes**) of DM (EEPROM) to be overwritten by the RAM buffer
aDmQuan - length of overwritten block (DM) **in bytes**
aUsage - functionality of RAM buffer according to enum RAM_BUF_USAGE – see 16.2.4

Return Value: A) -1: incorrect parameter(s)
B) according to AUTO_ERROR_LEVEL

Note: This function does not execute immediately changes the programmed unit, it just informs the ICP2 channel to overwrite PM/DM during programming

Example: Overwrite 256(dec) bytes of PM buffer starting from address 1000(dec)
IcpRamUsage (1000, 256, 0, 0, RAM_BUF_PM);

10.2 IcpRamBufWrite

See also: IcpRamUsage() and IcpRamBufRead()

Description: Writes up to 256 bytes into RAM buffer of ICP2 unit
Prototype: int DLL_FUNC IcpRamBufWrite(unsigned char *aRamBuf, int aStartAddr, int aQuan)
Parameters: aRamBuf - pointer to user's buffer to be copied from
aStartAddr-start address of the RAM buffer
aQuan - number of bytes to be written
Return Value: A) -1: incorrect parameter(s)
B) according to AUTO_ERROR_LEVEL

Example: Load RAM buffer with a string below

```
#define OVERWRITE_LEN  20 //20 bytes to be loaded
char MyBuf[] = "ICP2 Programmer";
IcpRamBufWrite(MyBuf, 0, OVERWRITE_LEN);
```

10.3 IcpRamBufRead

See also: IcpRamUsage() and IcpRamBufWrite()

Description: Reads up to 256 bytes from RAM buffer of ICP2 unit
Prototype: int DLL_FUNC IcpRamBufRead(unsigned char *aRamBuf, int aStartAddr, int aQuan)
Parameters: aRamBuf - pointer to user's buffer to be copied to
aStartAddr-start address of the RAM buffer
aQuan - number of bytes to be read
Return Value: A) -1: incorrect parameter(s)
B) according to AUTO_ERROR_LEVEL

Example: Validate that RAM buffer is loaded as expected

```
#define OVERWRITE_LEN  20 //20 bytes to be loaded
char MyBuf[] = "ICP2 Programmer";
char MyVerifyBuf[OVERWRITE_LEN];
IcpRamBufWrite(MyBuf, 0, OVERWRITE_LEN);
IcpRamBufRead(MyVerifyBuf, 0, OVERWRITE_LEN);
Compare MyBuf and MyVerifyBuf
```

10.4 IcpRamByteWr and IcpRamByteRd

IcpRamByteWr and IcpRamByteRd can be used as alternative functions to IcpRamBufWrite and IcpRamBufRead respectively – see “c_icpexp.h” for details

10.5 IcpPcTmpBufWr and IcpTrnasferPcTmpBufToRamBuf

IcpPcTmpBufWr and IcpTrnasferPcTmpBufToRamBuf can be used as alternative functions to IcpRamBufWrite and IcpRamBufRead respectively – see “c_icpexp.h” for details

11 Standalone Operation: Multiple Environments

Depending on programmer model multiple environments can be loaded.
The following environment is selected after power-up:
- ICP2/ICP2-GANG: environment 1
- ICP2-Portable: last used (1 to 6)
- ICP2-COMBO: environment 1 or as selected by hardware control interface (1 to 6)

11.1 IcpSwitchEnv

Description: Communicates with programmer and simultaneously switches environment for all active channels

IMPORTANT: **All environment-related functions after this command will be applied to the selected environment until power-up occurs**

Prototype: int DLL_FUNC IcpSwitchEnv (int aEnv);
Parameters: aEnv should be 0 (environment 1), 1 (environment 2), …, 5 (environment 6)
Note: number of available environments:
- ICP2, ICP2(HC) and ICP2-GANG: 2
- ICP2-COMBO and ICP2-Portable: 6

Return Value: A) -2: operation is not supported (ICP-01)
B) according to AUTO_ERROR_LEVEL

Example 1:
```
IcpSwitchEnv(1); //switch to environment 2
IcpTransferEnvironmentToIcp ("File1.pj2"); //transfer environment file to
                                           //environment 2
```

Example 2:
```
IcpSwitchEnv(5); //switch to environment 6
IcpDoAction(ACT_STA_PROG,0,0,0,0,0,“”); //program device with environment 6
```

12 PC-Driven Operation: Access PC Memory Buffers

WARNING: be careful with memory manipulation since it changes your programming data

Function IcpGetBuffers() allows reading and modification of internal programming buffers in PC-driven mode. It may be useful for applications that require buffer manipulation without (or in addition to) using hex file.

NOTE: be careful when modifying programming buffers

12.1 IcpGetBuffers

```
typedef struct { //use members shown in bold italic
   int      PmWordSize;          // Program Memory word size
   unsigned PmMaxWordVal;        // Program Memory maximal word value
   int      PmAddrUnit;          // Program Memory address unit
   char ***PmBuf***; int PmSize;  // Program Memory (flash)
   char ***IdBuf;*** int IdSize;  // ID Memory
   char ***DmBuf;*** int DmSize;  // Data Memory (EEPROM)
   char *CmBuf; int CmSize;     // Calibration Memory (not supported)
   char ***FuBuf;*** int FuSize;  // Fuses Memory (configuration words)
} iBUFFERS;
```

Description: Provides pointer to the internal programming buffers (in PC memory)
Prototype: void DLL_FUNC IcpGetBuffers (iBUFFERS *aBufs);
Parameters: pointer to structure iBUFFERS
Return Value: none

Example:
```
iBUFFERS MyBuf;
IcpGetBuffers(&MyBuf);
MyBuf.DmBuf[1]=0x55; //modify buffer for EEPROM programming
```

```
MyBuf.DmBuf[2]=0x56; //
IcpDoAction(ACT_PROG,…); //execute PC-driven programming
```

12.2 IcpBufWr

Description: Writes (modifies) internal buffer of selected memory area (1 bytes)

Prototype: int DLL_FUNC IcpBufWr (int aBufType, unsigned int aOffset, unsigned char aData);
Parameters: aBufType: buffer type according to enum MEMORY_SPACES
aAddr: offset in buffer in **bytes**
aData: value to be written
Return Value: A) 0: OK
B) -1: incorrect aBufType
C) -2: incorrect aOffset

Example: IcpBufWr(PM_SPACE, 12, 0x55); //write 0x55 to PM buffer, offset 12(dec)

12.3 IcpBufRd

Description: Reads 1 byte from internal buffer of selected memory area

Prototype: int DLL_FUNC IcpBufRd (int aBufType, unsigned int aOffset, unsigned char* aData);
Parameters: aBufType: buffer type according to enum MEMORY_SPACES
aAddr: offset in buffer in **bytes**
*aData: pointer to a read byte
Return Value: A) 0: OK
B) -1: incorrect aBufType
C) -2: incorrect aOffset

Example:
```
unsigned char my_data;
IcpBufRd(PM_SPACE, 12, &my_data); //my_data = value of PM buffer, offset 12(dec)
```

13 PC-Driven Operation: Enhanced and Low-Level ICSP™

Starting from software 4.9.1 (Jan-2012) most of dsPIC33/PIC24 devices can be operated in both low-level and Enhanced ICSP™ modes. Enhanced ICSP™ provides much faster operation while low-level programming is more suitable for manipulation with small data areas.

IMPORTANT: In order to use Enhanced ICSP™, a pull-down resistor 3.3K-10K Ohm must be placed between T_MOSI (PGD) and GND. If your PCB contains a PGD pull-up resistor then value of the resistor should be about 20% of the pull-up resistor but not less than 1.5K Ohm. For more info contact Softlog Systems: support@softlog.com

IMPORTANT - Enhanced ICSP™ limitations (silicon issue):
- PGEC3/PGED3 programming pair does not work on several devices – check Microchip® errata
- Enhanced ICSP™ may not work if “Windowed WDT” is defined

13.1 IcpSetEnhancedProg

Description: Select Enhanced or low-level ICSP™

Prototype: int DLL_FUNC IcpSetEnhancedProg(int aEnhancedOn);
Parameters: aEnhancedOn=1: select Enhanced ICSP™
aEnhancedOn=0: select low-level programming
Return Value: A) 0: OK
B) -1: Enhanced ICSP™ is not supported (8-bit devices, some 16-bit devices)
C) -2: Low-level ICSP™ is not supported (PIC32 family)

14 Secure Programming

14.1 IcpSecTransferSecEnv

A secure environment can be transferred by 3 methods:

- Off-line by using " ICP Secure Programming" utility (ADMIN or USER)
- On-line by using DLL function below
- On-line or off-line by using "ICP Command Line" utility (switch /u)

IMPORTANT: A transferred environment is saved in **non-volatile** flash memory of the programmer and automatically reloads after power-up. Don't transfer the same environment multiple times to keep endurance of internal ICP2 flash (10K cycles minimum, 100K typical)

Description: Transfers a secure environment file to a **single** channel of the programmer

Prototype: int DLL_FUNC IcpSecTransferSecEnv (const char *aSenName);
Parameters: aSenName - pointer to char string containing file name
Return Value: according to AUTO_ERROR_LEVEL

Example: IcpSecTransferSecEnv ("File1.sen");

14.2 IcpDllGetInfoSingle – see 8.3

15 Direct Hardware Control

ICP DLL provides direct access to ICP2/ICP2-GANG/ICP2-COMBO hardware. The functions are listed below, for more details see document "ICP2 - FTM Functions.pdf"

###	*Function Name*	*Function Description*
1.	Icp2AdConv()	Execute A/D Conversion
2.	Icp2DaVolt ()	Set D/A Levels (not pins) for VDD, VPP or VLIM
3.	Icp2PinState()	Set/Read Pin State
4.	Icp2ClockDataComPar()	Set CLOCK/DATA ("RB6/RB7") Communication Parameters
5.	Icp2SingleClockDataAction()	Execute single CLOCK/DATA Communication
6.	Icp2MultiClockDataAction()	Execute multiple CLOCK/DATA Communications
7.	Icp2SafeOff()	Turn ICP2 pins to safe state (off)

16 More DLL Functions

File "c_icpexp.h" contains many additional DLL functions (write/read to internal ICP EEPROM, calibration, FTB9 operation, etc.). Specify your tasks and contact Softlog Systems for suggestion. Additional DLL functions can be done upon your request

Appendix A: Return Values and Definitions

16.1 Return Values

DLL functions return value according to enum AUTO_ERROR_LEVEL below:

```
enum AUTO_ERROR_LEVEL { //return values
AUTO_OK                 = 0,  //operation OK
AUTO_DB_ERR             = 1,  //database error
AUTO_COM_ERR            = 2,  //communication error
AUTO_VDD_ERR            = 3,  //Vdd overload error
AUTO_VPP_ERR            = 4,  //Vpp overload error
AUTO_HEX_ERR            = 5,  //HEX file loading error
AUTO_SER_ERR            = 6,  //serialization file error
AUTO_VER_ERR            = 7,  //verification error
AUTO_ERR_NO_SPACE       = 8,  //no space selected
AUTO_SAVE_ERR           = 9,  //file save error
AUTO_SOCK_ERR           = 10, //socket communication error (obsolete)
AUTO_I2C_ERR            = 11, //UUT I2C communication error
```

```
AUTO_DLL_ERR                = 12, //DLL programming is not supported
AUTO_KEY_ERR                = 13, //key generation error
AUTO_CFG_ERR                = 14, //config. file error
AUTO_COM_NUM_ERR            = 15, //invalid COM number
AUTO_COM_BUSY_ERR           = 16, //selected COM is busy
AUTO_COM_BAUD_ERR           = 17, //invalid baud rate
AUTO_COM_NO_OPEN            = 18, //can't open COM port
AUTO_USER_CANCEL            = 19, //user cancel
AUTO_IN_PROGRESS            = 20, //operation in progress
AUTO_BC_ERR                 = 21, //blank check error
AUTO_OP_NOT_ALLOW           = 22, //operation not allowed for selected programmer
AUTO_FW_INVALID             = 23, //firmware invalid-firmware upgrade needed
AUTO_24LC_ADDR_ERR          = 24, //24LC01 address (offset) is out of range
AUTO_DM_ADDR_ERR            = 25, //DM range error
AUTO_FIRM_ERR               = 26, //firmware version error
AUTO_NO_SUB                 = 27, //no ICP-SUB PCB
AUTO_NO_SUP_KEE             = 28, //no keeloq support
AUTO_NO_SUP_DSPIC           = 29, //no dsPIC support
AUTO_ICP2_REQ               = 30, //ICP2 required
AUTO_DEV_ERR                = 31, //device selection error (unspecified error)
AUTO_PROG_MISMATCH          = 32, //mismatch between selected and detected programmers
AUTO_PRJ_INVALID            = 33, //Invalid environment
AUTO_PRJ_DB_FIRM_PC_MIS     = 34, //mismatch between PC and firmware database
AUTO_PRJ_DB_FIRM_AT45_MIS   = 35, //mismatch between environment and firmware database
AUTO_DLL_SUPPORT_REQIURED   = 36, //obsolete: "GO" pressed on hardware and no DLL/standalone support
AUTO_PRJ_CS                 = 37, //environment CS error
AUTO_STA_IDLE               = 38, //programmer is idle or standalone operation can't be started
AUTO_STA_BUSY               = 39, //standalone operation: programmer busy
AUTO_ENV_ERR                = 40, //environment file error
AUTO_PM_RANGE               = 41, //invalid PM range specified
AUTO_SEC_SUPPORT_REQUIRED   = 42, //Security support required
AUTO_SEC_CNT_INTEG          = 43, //Future: Security feature: integrity error in counter
AUTO_SEC_CNT_ZERO           = 44, //Future: Security feature: counter = 0
AUTO_SEC_NO_FUNC            = 45, //Future: Security feature: function does not exist
AUTO_SEC_PACK_ERR           = 46, //Future: Security feature: packet error
AUTO_SEC_EEPROM_FAIL        = 47, //Future: Security feature: EEPROM error
AUTO_SEC_ANTI_SCAN          = 48, //Future: Security feature: anti-scan activated,
AUTO_SEC_SEC_ID_CMP         = 49, //Future: Security feature: incorrect Security ID
AUTO_SEC_PASSW_CMP          = 50, //Future: Security feature: incorrect password
AUTO_SEC_BATCH_CMP          = 51, //Future: Security feature: incorrect batch
AUTO_SEC_VERS_ERR           = 52, //Future: Security feature: version error
AUTO_SEC_UNKNOWN_ERR        = 53, //Future: Security feature: unknown error
AUTO_NO_ROW_ERASE           = 54, //row erase is not supported
AUTO_INVALID_PARAM          = 55, //invalid parameters
AUTO_MOVLW_RETLW_CALIB      = 56, //no movlw in calibration word
AUTO_NO_USUAL_ENV_TRAN      = 57, //Usual environment can't be sent if a secure one inside
AUTO_SEC_BUF_START_ADDR     = 58, //sec. buf. properties error: incorrect start addr
AUTO_SEC_BUF_END_ADDR       = 59, //sec. buf. properties error: incorrect end addr
AUTO_SEC_BUF_PAGE_START     = 60, //sec. buf. properties error: incorrect page start
AUTO_SEC_BUF_PAGE_SIZE      = 61, //sec. buf. properties error: incorrect page size
AUTO_SEC_BUF_NOT_EVEN       = 62, //sec. buf. properties error: length not even
AUTO_SEC_BUF_NO_DM          = 63, //sec. buf. properties error: no DM in PIC
AUTO_SEC_BUF_LAST_PAGE      = 64, //sec. buf. properties error: last PM page can't be used
AUTO_SEC_BUF_NO_16BIT_SUP   = 65, //sec. buf. properties error: no Script 1 for 16-bit devices
AUTO_SEC_BUF_NOT_MODULO_3   = 66, //sec. buf. properties error: length not modulo 3
AUTO_SEC_EMPTY_MASK         = 67, //Security feature: empty mask for secure environment
AUTO_TEST_COM_NO_SUPPORT    = 68, //ICP2 test command not supported
AUTO_TEST_NACK              = 69, //ICP2 test command returns NACK
AUTO_NO_SUP_P32             = 70, //no PIC32 support
AUTO_PIC32_BUSY_OR_DAMAGED  = 71, //PIC32 is busy or damaged
AUTO_PIC32_CP_OR_DAMAGED    = 72, //PIC32 is code protected or damaged
AUTO_PIC32_PE_ANSWER        = 73, //PIC32 programming executive: no answer
AUTO_PIC32_PE_VERSION       = 74, //PIC32 programming executive: incorrect version
AUTO_SEC_BUF_NO_32BIT_SUP   = 75, //no security support for PIC32
AUTO_CNT_ZERO               = 76, //non-secure (low-endurance) counter is 0
AUTO_SQTP_CONFLICT          = 77, //serialization from PC is not allowed if standalone serialization=ON
AUTO_INVALID_DEVICE_CFG     = 78, //invalid device number in CFG file. Use latest DLL
AUTO_DEV_ID_NO_SUPPORT      = 79, //Device ID read is not supported for the family
AUTO_ROW_PM_RANGE           = 80, //invalid PM range due to row size
AUTO_PE_MISMATCH            = 81, //Programming executive: mismatch between environment and firmware
AUTO_PE_NO_PGD_PULLDOWN     = 82, //No pull-down on PGD line
AUTO_PE_VER                 = 83, //PE verification failed
AUTO_PE_NO_IN_ENV           = 84, //PE does not present in environment
AUTO_PE_CALIB               = 85, //invalid calibration/diagnostic data
AUTO_PC_DRV_STA_CONFLICT    = 86, //conflict between PC-driven and standalone modes
AUTO_CALIB_WORD_1_CORRUPT   = 87, //Calibration word 1 corrupted during programming
AUTO_CALIB_WORD_2_CORRUPT   = 88, //Calibration word 2 corrupted during programming
AUTO_ENV_NUM_OUT_RANGE      = 89, //Specified environment number is out of range
```

```
AUTO_CYBL_ACQUIRE_TIMEOUT     = 90, //Device acquire timeout
AUTO_CYBL_SROM_ACT_TIMEOUT    = 91, //SROM operation timeout
AUTO_CYBL_VIRGIN_DEVICE       = 92, //Device is VIRGIN
AUTO_CYBL_SWD_ACK_FAULT       = 93, //ACK response for SWD transfer is not OK
AUTO_NO_FIRMWARE_CYBL         = 94, //no firmware for CYBL10x6x
AUTO_NO_FIRMWARE_I2C          = 95, //no firmware for I2C
AUTO_NO_FIRMWARE_DSPIC        = 96, //no firmware for dsPIC
AUTO_NO_FIRMWARE_P32          = 97, //no firmware for PIC32
AUTO_G3_REQUIRED              = 98, //G3 (ICP3M) is required for selected device
AUTO_G3_NO_PIC17C             = 99, //PIC17C is not supported by G3 (ICP3M)
AUTO_RESERVED_100             = 100, //reserved
AUTO_DEMO_ERR                 = 101, //demo version
AUTO_OTP_NOT_BLANK            = 102, //OTP area is not blank, no programming is allowed
AUTO_OTP_VER_ERR              = 103, //OTP verification error
AUTO_FBOOT_VER_ERR            = 104, //FBOOT verification error
AUTO_DUAL_PART_ILLEGAL_BUF    = 105, //illegal partition mode in programming buffer
AUTO_DUAL_PART_MISMATCH       = 106, //partition mode mismatch
AUTO_NOT_ALLOWED_IN_DUAL      = 107, //operation is not allowed in dual partition mode
AUTO_DUAL_PART_ILLEGAL_PIC    = 108, //illegal partition mode in PIC
AUTO_FBOOT_BLANK_ERR          = 109, //FBOOT blank check error
AUTO_ENV_SIZE_ERR             = 110, //environment size is too big for connected programmer
AUTO_GANG_COMBO_MISMATCH      = 111, //mismatch between ICP2-GANG and ICP2-COMBO
AUTO_SWD_DEVICE_PROTECTED     = 112, //SWD device is protected
AUTO_DEVICE_PROTECTED         = 112, //Device is protected (same errorcode as for AUTO_SWD_DEVICE_PROTECTED)
AUTO_SECURITY_BIT_VER_ERR     = 113, //Security bit verification error
AUTO_CANT_CONNECT_TO_UUT      = 114, //can't connect to UUT (target)
AUTO_SINGLE_WORD_RD_NO_SUP    = 115, //single word read not supported
AUTO_SINGLE_WORD_WR_NO_SUP    = 116, //single word write not supported
AUTO_ERASE_WRITE_TIMEOUT      = 117, //erase or write timeout
AUTO_UPDI_TINY_CRC_FAULT      = 118, //CRC fail. Execute programming with enabled bulk erase
AUTO_CANT_CONNECT_TO_UPDI     = 119, //can't connect to UPDI UUT (target)
AUTO_CANT_CONNECT_TO_TPI      = 120, //can't connect to TPI UUT (target)
AUTO_FTDI_LATENCY_BIG         = 121, //FTDI latency is too big
};
```

16.2 Definitions

16.2.1 Action List

```
enum ACTION_LIST {
  ACT_PROG                = 1, //PC-driven programming
  ACT_VER                 = 2, //PC-driven verification
  ACT_READ                = 3, //PC-driven read
  ACT_BC                  = 4, //PC-driven blank check

  ACT_STA_PROG            = 5, //Standalone programming
  ACT_STA_GET_RES         = 6, //Standalone: communicate and get latest results
  ACT_STA_CLR_RES         = 7, //Standalone: communicate and clear all latest results
  ACT_STA_START_PROG      = 8, //Standalone: start standalone programming (NOTE: should be monitored then for completion)
};
```

16.2.2 Memory Spaces (PC-driven Operation)

```
enum MEMORY_SPACES {
  PM_SPACE                = 0x0001, //program memory (flash)
  ID_SPACE                = 0x0002, //User ID memory
  DM_SPACE                = 0x0004, //data memory EEPROM)
  CM_SPACE                = 0x0008, //calibration memory (not supported)
  FU_SPACE                = 0x0010, //configuration word
  BM_SPACE                = 0x0020, //boot memory
  OTP_SPACE               = 0x0040, //OTP memory

  ALL_SPACE               = PM_SPACE | ID_SPACE | DM_SPACE | FU_SPACE | BM_SPACE, //all excluding OTP

  LAST_SPACE = 0x0080
};
```

16.2.3 Programmer Type and Mode

```
enum PROG_TYPE {
  PROG_ICP01              = 0, //ICP-01
  PROG_ICP2               = 1, //ICP2 or ICP2(HC)
  PROG_GANG4              = 2, //ICP2-GANG/ICP2-COMBO (multichannel)
  PROG_GANG4_SINGLECHAN   = 3, //ICP2-GANG/ICP2-COMBO (single channel)
  PROG_PORTABLE           = 4, //ICP2-Portable
  PROG_LAST
```

```
};

enum PROG_MODE {
  SPM_PCDRIVEN                  = 0, //PC-driven mode
  SPM_STANDALONE                = 1, //Standalone mode
  SPM_LAST
};
```

16.2.4 Usage of RAM Buffer

```
enum RAM_BUF_USAGE {
  RAM_BUF_NO                    = 0, //not used (default)
  RAM_BUF_PM                    = 1, //override PM
  RAM_BUF_DM                    = 2, //override DM (EEPROM)
  RAM_BUF_PM_DM                 = 3, //override PM and DM (128 per each, for future use)
  RAM_BUF_SEC_COM               = 4  //internal use only: security feature related communication between PC and ICP
};
```

16.2.5 Environment Status

```
enum PRJ_VAL {
  prjUNKNOWN                    = 0, //unknown state (for PC only)
  prjLOADING                    = 1, //environment loading/validation in progress
  prjINVALID                    = 2, //invalid environment
  prjVALID                      = 3, //valid environment
};
```

16.2.6 Additional Definitions

See files “fr_exp.h” and “fr_icp2.h” for more definitions

17 Appendix B: Software Examples

Contact Softlog Systems for more examples

17.1 Software Example: PC-Driven Mode, Single Channel

```
// This example shows how to use ICP2 in PC-driven mode
// Be sure that COM port and other settings in *.cfg file meet your hardware setup

#include "fr_exp.h"
#include "c_icpexp.h"

#define CFG_FILE       "icp01.cfg"
#define HEX_FILE       "hex1.hex"
#define SER_FILE       "ser1.ser"

#define MY_ACTION      ACT_PROG       //action = PC-driven programming
#define MEM_SPACE      ALL_SPACE      //selected operation will apply to all memory spaces

#define PM_USER        0              //0-full PM range, 1-user defined
#define PM_START       0              //not valid if PM_USER=0
#define PM_END         0              //not valid if PM_USER=0

#define DM_USER        1              //1-user defined DM (EEPROM) range
#define DM_START       0              //DM start
#define DM_END         0x3F           //DM end

int FullCycle(void)
{
int ret; //errorcode

  ret = IcpStartApplication (CFG_FILE);
  if(ret)
    return ret;

  ret = IcpInitCom (0, 0, 0); //init COM from *.cfg file
  if(ret) {
    IcpEndApplication();
    return ret;
  }

  //At this point application and USB/RS-232 COM port are successfully open

  //Load HEX and serialization (SQTP) files (SQTP file is optional, define it as "" if not required)
  //-----------------------------------------------------------------------------------------------
  ret = IcpLoadHexAndSerFile (HEX_FILE, SER_FILE);
  if(ret) {
```

```
    IcpEndApplication(); //it also closes opened COM
    return ret;
  }

  //Enable progress window (optional)
  //---------------------------------
  IcpEnableProgressWindow (1);

  //Maximize speed of PC operation (optional)
  //-----------------------------------------
  IcpSetSleepOnWaitingRS232(0); //no sleep

  //Set DM (EEPROM) memory range (optional)
  //---------------------------------------
  IcpSetDmRange (DM_USER, DM_START, DM_END);

  //Execute programming - this operation can be done several times
  //--------------------------------------------------------------
  ret = IcpDoAction (  MY_ACTION,
                       MEM_SPACE,
                       PM_USER,
                       PM_START,
                       PM_END,
                       0,
                       "");

  }
  IcpEndApplication();
  return ret;
}
```

17.2 Software Example: Standalone Mode, Gang Programming

```
/* This example shows how to use ICP2-GANG/ICP2-COMBO:
1) One-time operation: validate that environments inside ICP2-GANG channels
are OK, otherwise transfer desired environment to the non-volatile memory of the programmer
IMPORTANT: transfer environment as less as possible to keep endurance of internal ICP2 flash (10K
cycles minimum, 100K typical)
2) Multiple operation: GANG programming
Be sure that COM port and other settings in *.cfg file meet your hardware setup
*/

#include "fr_exp.h"
#include "c_icpexp.h"

#define CFG_FILE              "icp01.cfg"
#define CHANNEL_QUAN          4              //4 GANG channels

unsigned short MyChecksum     = 0xA882;      //example
char MyEnvFile[]              = "MyEnv.pj2"; //example
int Res[CHANNEL_QUAN];        //programming results for all GANG channels

int FullCycle(void)
{
int ret;
  ret = IcpStartApplication (CFG_FILE);
  if(ret)
    return ret;

  ret = IcpInitCom (0, 0, 0); //init COM from *.cfg file
  if(ret) {
    IcpEndApplication();
    return ret;
  }
  //At this point application and USB/RS-232 COM port are successfully open

  //Validate that all channels contain correct environment
  //------------------------------------------------------
  ret=ValidateEnvironments(); //see function below
  Sleep(1500); //delay should be at least 1 sec due to internal "dispatcher" delay in ICP firmware

  //Execute programming - this operation can be done several times
  //--------------------------------------------------------------
  if (! ret) {
    ret=ProgramGang();//see function below
    //<Analyze Res[]
  }
  IcpEndApplication();
  return ret;
```

```
}

  // Validate that all channels contain correct environment
  // (***only after selection of new project/product***)
  // ======================================================
int ValidateEnvironments(void)
{
int ch_i;
int err;
int update_env;
ICP_INFO ChannelInfo;

  IcpEnableProgressWindow(0); //disable progress window (can be enabled if you wish)
  for (ch_i=0; ch_i<CHANNEL_QUAN; ch_i++) //channel-by-channel
  {
    //Step 1.1: select GANG single channel + PC-driven
    //-------------------------------------------------
    IcpDllSelectProg(PROG_GANG4_SINGLECHAN, SPM_PCDRIVEN, ch_i);

    //Step 1.2: Read environment info from a selected channel
    //-------------------------------------------------------
    err = IcpDllGetInfoSingle(&ChannelInfo);
    //<analyze err>

    update_env=0; //prepare
    if (ChannelInfo.iValid) //valid structure
    {
      if (ChannelInfo.iEnvStat==prjINVALID) //invalid environment
        update_env=1;
      else if (ChannelInfo.iEnvHexFileCs != MyChecksum) //incorrect HEX checksum
        update_env=1;
      //additional analysis can be added here
    }
    if (update_env) //environment to be updated
    {
      //Transfer environment
      //--------------------
      IcpEnableProgressWindow(1); //enable progress window
      err = IcpTransferEnvironmentToIcp(MyEnvFile); //transfer environment
      //<analyze err>
      IcpEnableProgressWindow(0); //disable progress window
    }
  }
  return err;
}

  // GANG Programming
  // ================
int ProgramGang(void)
{
int ch_i;
int err;

  // Step 1: select programmer as GANG + standalone
  // ----------------------------------------------
  IcpDllSelectProg(PROG_GANG4, SPM_STANDALONE, 0);

  // Step 2: execute programming
  // ---------------------------
  IcpEnableProgressWindow (1);//enable progress window (optional)
  err=IcpDoAction(
        ACT_STA_PROG,  //aAction
        0,             //aMemorySpace (does nothing in standalone programming)
        0,             //aPmUserRange (does nothing in standalone programming)
        0,             //aPmAddrBeg   (does nothing in standalone programming)
        0,             //aPmAddrEnd   (does nothing in standalone programming)
        0,             //aSaveResult
        "");           //aReadFile

 IcpEnableProgressWindow (0);//disable progress window

  //Step 2.3: Analyze result
  //------------------------
  if (err==AUTO_OK)
  {
    ; //do nothing, all channels OK
  }
  else
  {
```

```
    for (ch_i=0; ch_i<CHANNEL_QUAN; ch_i++)
      Res[ch_i]=IcpReadStaResOneCh(ch_i); //read and save all results
  }
  return err;
}
```

17.3 More Software Examples

Contact Softlog Systems for more software examples (GANG programming with RAM buffer, other PC operations during programming, final test functions, etc.)

18 Revision History

- Revision 8.8.1a (Jul-18):
  - added reference to new product ICP2-Portable(G3)
  - updated structure ICP_INFO
  - increased number of COMs to 256

- Revision 8.1.1a (Mar-17):
  - added reference to new products (“G3 products”): ICP2(G3), ICP2-GANG(G3) and ICP2-COMBO(G3)
  - added more spaces to enum MEMORY_SPACES
- Revision 4.16.1a (Jan-16):
  - added maximum COM port warning
  - minor description changes
- Revision 4.13.1a (Jan-15):
  - changed ICP software setup destination (new: C\Softlog\..., old: C:\Program Files\Soft-Log)
  - added ICP2-COMBO related info
  - added description of 1500ms delay between PC-driven and standalone commands
- Revision 4.12.1 (Aug-13):
  - added description of IcpSetChipEraseBeforeProg – see 6.14
  - Device ID read is also available for PIC24/dsPIC33 – see 6.12
  - added warnings for data manipulation – see chapters 10 and 12
- Revision 4.10.1 (Aug-12): added Gap Eliminator™ paragraph
- Revision 4.9.2 (Apr-2012): added description of Enhanced ICSP limitations - see Chapter 13
- Revision 4.9.1 (Jan-2012):
  - added IcpBufWr() and IcpBufRd() for easy modification of programming buffers
  - added “Enhanced and Low-Level ICSP™” - see Chapter 13
  - added “Secure Programming” - see Chapter 14
- Revision 4.8.2 (Aug-2011): added “Read Device ID” functionality for IcpSingleWordRead()
- Revision 4.8.1 (Jul-2011):
  - added function IcpCalcPmRange() for the best PM range selection
  - added new return values (AUTO_SQTP_CONFLICT, AUTO_INVALID_DEVICE_CFG)

19 Technical Assistance

You may contact Softlog Systems for technical assistance by calling, sending a fax or e-mail. To help us give you quick and accurate assistance, please provide the following information:

- Software version number, firmware version number and product serial number (if available). This information is displayed at the program start
- Detailed description of the problem you are experiencing
- Error messages (if any)
- Microcontroller part number (if device-related)
- Send us your "icp01.cfg" file

20 Warranty

Softlog Systems (2006) Ltd. warrants this product against defects in materials and workmanship for a period of 1 (one) year. This warranty will not cover programmers that, in the opinion of Softlog Systems, have been damaged due to abuse, improper use, disassembly, replacement of parts or attempted repair by anyone other than an authorized Softlog Systems service technician.

This product must be returned to the supplier for warranty service within the stated period. The buyer shall pay all shipping costs and other charges or assessments for the product by the supplier.

Softlog Systems shall not be liable for any indirect, incidental, or consequential damages, regardless of whether liability is based upon breach of warranty, negligence, strict liability in tort, or any other theory, Softlog Systems will never be liable in an amount greater than the purchase price of the products described by this express warranty. No agent, distributor, salesperson, or wholesale or retail dealer has the authority to bind Softlog Systems to any other affirmation, representation, or warranty concerning these goods.

21 Contact

Softlog Systems (2006) Ltd.

6 Hayotzrim St. Or-Yehuda 6021820 Israel

Phone: 972-3-9515359
Fax: 972-3-9527520
Web: www.softlog.com
E-mail: sales@softlog.com, support@softlog.com

22 Copyright Notice