

SMARTWARE

11 avenue des Andes

91940 LES ULIS

FRANCE

☎ +33 (0) 1.64.86.25.25 - Fax +33 (0) 1.64 86 25 26

<http://www.smartware.fr>

SMARTGEAR

Applications Programmer Manual

January 2006

Reference SMARTGEAR_DEV_KIT_V2R00.DOC

DOCUMENT OUTLINE

SMARTGEAR200	January 2006	F.GIRARDOT
SMARTGEAR116	September 2004	F.GIRARDOT
SMARTGEAR102	April 2002	F.GIRARDOT
SMARTGEAR102	April 2002	R.ALVES Preliminary version

TABLE OF CONTENTS

1	<u>PRESENTATION</u>	<u>6</u>
1.1	Terminology	6
1.2	Bibliography	6
1.3	Required hardware and software	6
 2	 <u>SOFTWARE ARCHITECTURE</u>	 <u>7</u>
2.1	Organization of directories	7
2.1.1	SMARTGEAR directory	8
2.1.2	PROJECT directory	9
2.1.3	Additional directories	10
2.1.4	MLOS.INI file (SmartGear)	10
2.2	MLOS.INI file (PaccomSvr)	Erreur ! Signet non défini.
2.3	Window chaining	11
2.4	Other Views	14
2.5	The SMARTGEAR.INI file	16
2.6	The PacOps.ini File	Erreur ! Signet non défini.
2.6.1	Section [SERVER]	Erreur ! Signet non défini.
2.7	The PROJECT.INI file	17
2.7.1	Section [PROJECT]	18
2.7.2	Sections [JOBUSER_X]	19
2.7.3	Sections [JOBUSERMENU_X]	19
2.7.4	Sections [STATION_X]	21
2.8	General context SMARTGEAR	23
2.8.1	spm_AccesControlType	25
2.8.2	spm_MachineStatusType	26
2.8.3	spm_SoftwareInfoType	27
2.8.4	spm_UserInterfaceType	27
2.8.5	spm_StepCfgType	28
2.9	Project management DLL	28
2.9.1	Presentation	28
2.9.2	Specific project context	29
2.9.3	Standard points of entry	30
2.9.3.1	Project_Start ()	30
2.9.3.2	Project_End ()	30
2.10	Job management DLL (JOB)	31
2.10.1	Presentation	31
2.10.2	Specific job context (Job)	32
2.10.3	Standard points of entry (Management/Display Mode)	33
2.10.3.1	Job_ListCreate (...)	33
2.10.3.2	Job_List (...)	33
2.10.3.3	Job_ListSelect (...)	33

2.10.3.4	Job_Start (...)	34
2.10.3.5	Job_End (...)	34
2.10.4	Functions defined by user (Buttons Nos. 1 to 12)	34
2.10.5	Functions defined by user (User menu Nos 1 to 20)	34
2.10.6	Standard points of entry (Production Mode)	35
2.10.6.1	Job_RunningStart	35
2.10.6.2	Job_RunningEnd	35
2.10.6.3	Job_RunningTransferStart	35
2.10.6.4	Job_RunningTransferEnd	35
2.10.6.5	Job_RunningEmptyTransfer	36
2.10.6.6	Job_RunningEmptyMachine	36
2.10.6.7	Job_RunningButtonStart	36
2.10.6.8	Job_RunningButtonPause	36
2.10.6.9	Job_RunningButtonAbort	36
2.10.6.10	Job_RunningButtonEmptyMachine	36
2.10.6.11	Job_RunningPCAMessage	37
2.11	Station management DLL (STEP)	38
2.11.1	Presentation	38
2.11.2	Card status management	38
2.11.3	General station context	39
2.11.4	Station specific context	40
2.11.5	Station operating modes	42
2.11.5.1	MACHINE Mode	42
2.11.5.2	STATIC Mode	42
2.11.5.3	DYNAMIC and COMPUTE Modes	43
2.11.5.4	MULTIPRINT Mode	44
2.11.5.5	INDEXED Mode	45
2.11.5.6	STEPINDEXED Mode	46
2.11.5.7	FREE Mode	46
2.11.6	Standard entry points	47
2.11.6.1	Step_Start (Task initialization)	47
2.11.6.2	Step_CardTransfer	47
2.11.6.3	Step_CardStart	47
2.11.6.4	Step_CardRun	48
2.11.6.5	Step_CardEnd	48
2.11.6.6	Step_CardOutOfRange	48
2.11.6.7	Step_End	48
2.11.7	Specific entry points for INDEXED stations	49
2.11.7.1	Step_CardIn	49
2.11.7.2	Step_CardOut	49
2.12	Card context	50
2.12.1	Presentation	50
2.12.2	Size	50
2.12.3	Back-up of machine context	50
2.12.4	Associated C functions	51
2.12.4.1	CardCtx_Write ()	51
2.12.4.2	CardCtx_Read ()	52
2.12.4.3	CardCtx_Info ()	52
2.12.4.4	CardCtx_Clear ()	53

2.12.4.5	Example of use	53
2.13	Spm30 standard API	56
2.13.1.1	Msg_Stack	56
2.13.1.2	UserButtonXXX	56
2.14	SmartGearUI API	57
2.14.1.1	Intranet view management	57
2.14.1.1.1	sgui_IntranetPageSet	57
2.14.1.1.2	sgui_IntranetHomeSet	57
2.14.1.1.3	sgui_IntranetBack	57
2.14.1.1.4	sgui_IntranetForward	57
2.14.1.2	Batch Bitmap management	57
2.14.1.2.1	sgui_BatchBitmapSet	57
2.14.1.3	SmartGear Control Interface	57
2.14.1.3.1	sgui_ProductionModeSet	57
2.14.1.4	SmartMotion Interface	58
2.14.1.4.1	sgui_StationMaintenanceProcess	58
2.14.1.4.2	sgui_SetPCPStatus	58
2.14.2	Writing conventions	59
2.14.2.1	Modules and Symbols	59
2.14.2.2	Errors	59
2.14.2.3	Function comments	59
3	<u>DEVELOPMENT KIT INSTALLATION</u>	60
3.1	Installation procedure	60
3.2	Visual C++ configuration	60
3.3	Demo project	62
3.3.1	Demo project building	62
3.3.2	Debugging DEMO Application project	64
3.4	SMARTGEAR WORKFLOW	65

1 PRESENTATION

This manual is intended for application developers for SMARTGEAR. It details the general concepts used by the SMARTGEAR software from version v1r02.

1.1 Terminology

PCA	Logic control PC
PCP	Customization PC
FACET	FASt Card Encoder and Tester (Customization tester)

1.2 Bibliography

Microsoft Visual C++ 6.0 or later
SMARTWARE – PowerFACET or UltraSmart development kit

1.3 Required hardware and software

PC PENTIUM 4 1,5 GHz – RAM 512 MO
PowerFACET or UltraSmart tester (optional)
Microsoft Windows NT , 2000 or XP
Microsoft Visual C++ 6.0 or later

2 SOFTWARE ARCHITECTURE

2.1 Organization of directories

The structure of the directories for SMARTGEAR must be complied with for correct operation and consistency between the various projects. However, directories specific to the projects (for data and traces) can be generated. The standard tree structure is defined below.

```

\SMARTGEAR
  \BIN
    SMARTGEAR.EXE
    SMARTGEAR.INI
    SMARTGEARUI.DLL
    SGRES.DLL,
    SPM30.DLL
    MLFC.DLL
    MLOS.DLL , CARD.DLL, MLOSMAN.DLL
    MLOS.INI
    MLOSV3.DLL
    MLLWKS.DLL
    MLLWKS.INI
    MLOSWKS.INI
    STALK.DLL, START.MLL
    SPMWIN.MSG
    SMARTGEAR.BMP
    INTRANET.HTML
    OPENPERSO.SWF
    CLOCK.AVI
  \CFG
  \DLL
  \PROJECT
    \<PROJECT NAME No. 1>
      \DLL
      \SRC1
        \COMMON
        \TASK N°1
        ...
        \TASK N°X
    \<PROJECT NAME No. X>
      ...
  \KIT1
    \INCLUDE
      *.H
      \MLOS
        *.H
    \LIB
      SPM30.LIB
      SMARTGEARUI.LIB
      \MLOS

```

BOLD FOR MANDATORY DIRECTORIES OR FILES¹ (For development station only)

2.1.1 SMARTGEAR directory

The executable program *SMARTGEAR.EXE* and all the DLLs needed for the operation of SMARTGEAR are to be found in the ***BIN*** sub-directory. The SMARTGEAR.INI file (optional) can be used for configuring some of the operating parameters of the software (see paragraph 0

The SMARTGEAR.INI file).

When an SMARTGEAR session is interrupted and there are still cards in the machine, the configuration (card status and context) is memorized in the **MACHINE.CFG** file of the **CFG** sub-directory. Note that the file is deleted when the machine no longer contains any cards.

The **KIT** sub-directory contains all the "header" (.h) and interface (.lib) files needed for the development of applications running SMARTGEAR. Respectively, they are to be found in the **INCLUDE** and **LIB** sub-directories.

Finally, any existing projects are stored in the **PROJECT** sub-directory.

2.1.2 PROJECT directory

This directory contains the list of projects appearing on the machine. Note that these directories appear in the list of projects when SMARTGEAR is launched. Therefore, all your projects must be placed in the **PROJECT** sub-directory.

Each project directory contains, at the least, the **DLL** sub-directory. It is in this directory that SMARTGEAR will seek, as a priority, the DLLs associated with this project.

It is customary to use the **SRC** sub-directory to store the sources of the various DLLs associated with the project. Similarly, the **COMMON** sub-directory is used for placing common files needed for generating the different DLLs.

2.1.3 Additional directories

As you wish, you can new sub-directories to structure your projects.

In the above example, the sub-directories:

- *In*: contains the data files needed for customization;
- *Log*: contains the production log files;
- *Facet*: contains the programs to be downloaded into the encoding stations.

Some of these directory names can be specified in a standard manner in the **PROJECT.INI** file.

Caution, these directories are not generated automatically by SMARTGEAR (C.F. 2.5.1)

2.1.4 MLOS.INI file (SmartGear)

The **MLOS.INI** file used for configuring the various networks and bridges used by MLOS. These networks are a way of establishing dialogue with the different objects set out through the system, for instance, the FACET or PowerFACET testers or the security servers.

Example of MLOS.INI file

```
[ETHERNET]
IpBase=192.168.0.80
Ip=192.168.0.159

; MLP with RS232 or RS485 link
[MLP]
AutoStart = 0
LocalNode = 0x4F
Verbose = 0
Port = 1
TA = 5

; MLP with IP link
[MLPIP]
AutoStart = 1
Verbose = 0
LocalNode = 0x4F
TargetNodes = 1..64

;inter-process communication network
[MLPMEM]
Verbose=1
AutoStart=1
LocalNode=0x4F
TargetNodes=0x4E ; Pacopssvr.exe local node

[ETHERNET]
LocalNode=0x4F
Ip=192.168.0.159
[NETWORK]
NodeName=SMARTGEAR01 ; Network name
```

For more details see PowerFACET development kit.

2.2 Window chaining

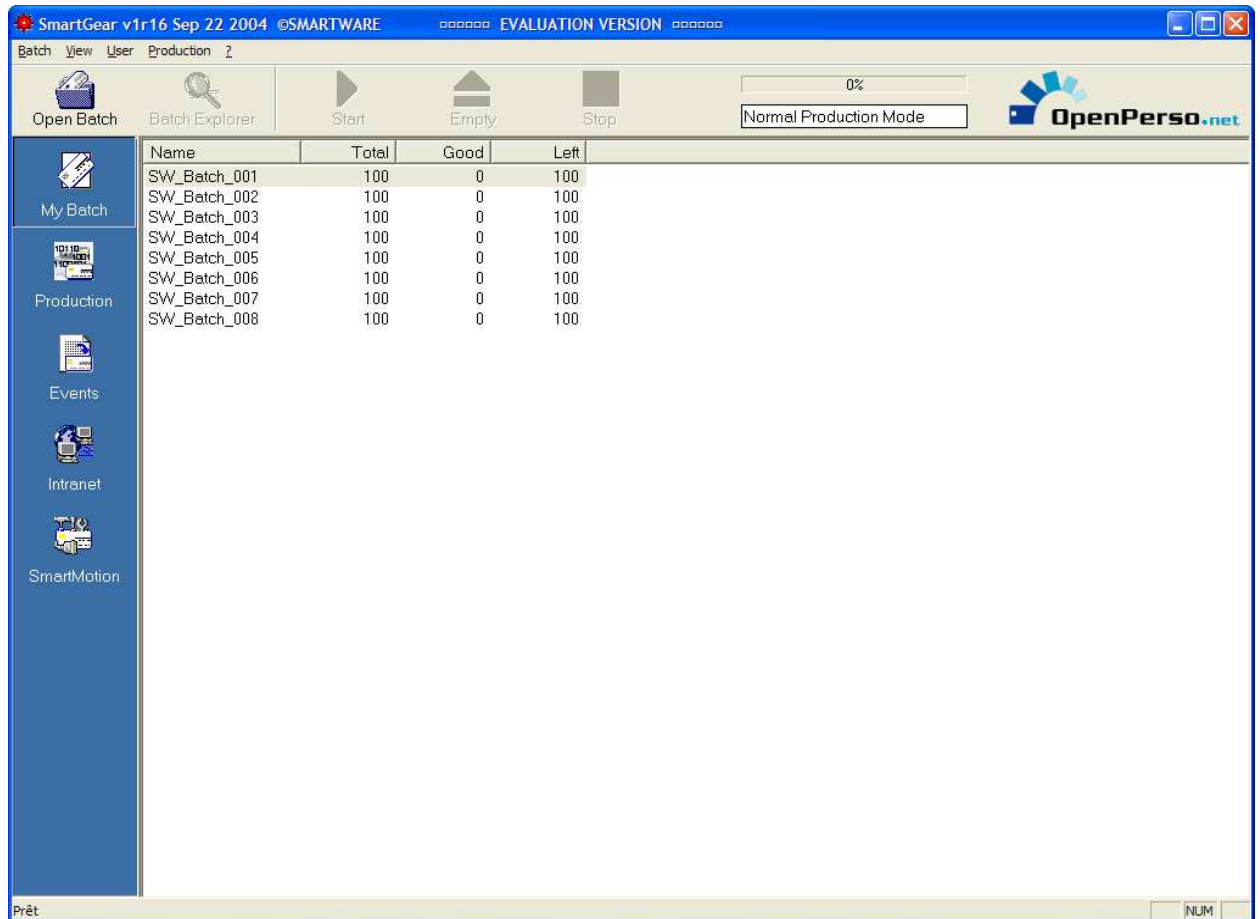
There are three main windows in SMARTGEAR called:

ProjectList The available project list is displayed



When SMARTGEAR starts, the **ProjectList** screen appears. It is then possible to select a project by double-clicking its name in the list (first column of the list).

MyBatch The list of associated jobs for a given project



Once a project has been selected, the **MyBatch** view appears with a list of jobs associated with the project. By double-clicking on a job or selecting it and pressing the OpenBatch button, we move on to the **Production** view.

The screen also has buttons accessible for access to the functions defined for the user.

Production view Production mode in which the operator can monitor and check the operation of the machine.

SmartGear v1r16 Sep 22 2004 ©SMARTWARE EVALUATION VERSION

Batch View User Production 2

Close Batch Batch Explorer Pause Empty Stop 69% Normal Production Mode OpenPerso.net

Batch Name : Gsm1912 Card Total 1500 Start Time 10:43

From : C:\SMARTGEAR\PROJECT\DEMO Estimated End Time 11:01

Card Good 1043 Card Left 457 Average Cards/h 5116

Card Bad 0 Card to Remake 0 Instant Card/h 5128

Machine List Station List

Date	Hour	Message
2004/09/22	10:55:55	0000001056
2004/09/22	10:55:55	SCCAR (6) CardOut Station #01 UID 00001044, Status 06
2004/09/22	10:55:56	pacom_Transfer request
2004/09/22	10:55:56	Transfer End (0)
2004/09/22	10:55:56	0000001057
2004/09/22	10:55:56	SCCAR (6) CardOut Station #02 UID 00001045, Status 06
2004/09/22	10:55:56	pacom_Transfer request
2004/09/22	10:55:56	Transfer End (0)
2004/09/22	10:55:56	0000001058
2004/09/22	10:55:56	SCCAR (6) CardOut Station #03 UID 00001046, Status 06
2004/09/22	10:55:57	pacom_Transfer request
2004/09/22	10:55:57	Transfer End (0)
2004/09/22	10:55:57	0000001059
2004/09/22	10:55:57	SCCAR (6) CardOut Station #04 UID 00001047, Status 06
2004/09/22	10:55:58	pacom_Transfer request
2004/09/22	10:55:58	Transfer End (0)
2004/09/22	10:55:58	0000001060
2004/09/22	10:55:58	SCCAR (6) CardOut Station #05 UID 00001048, Status 06
2004/09/22	10:55:58	pacom_Transfer request
2004/09/22	10:55:58	Transfer End (0)
2004/09/22	10:55:58	0000001061
2004/09/22	10:55:58	SCCAR (6) CardOut Station #06 UID 00001049, Status 06
2004/09/22	10:55:59	pacom_Transfer request
2004/09/22	10:55:59	Transfer End (0)
2004/09/22	10:55:59	0000001062
2004/09/22	10:55:59	SCCAR (6) CardOut Station #07 UID 00001050, Status 06
2004/09/22	10:56:00	pacom_Transfer request
2004/09/22	10:56:00	Transfer End (0)
2004/09/22	10:56:00	0000001063
2004/09/22	10:56:00	SCCAR (6) CardOut Station #08 UID 00001051, Status 06

Prêt NUM

2.3 Other Views

Event view :

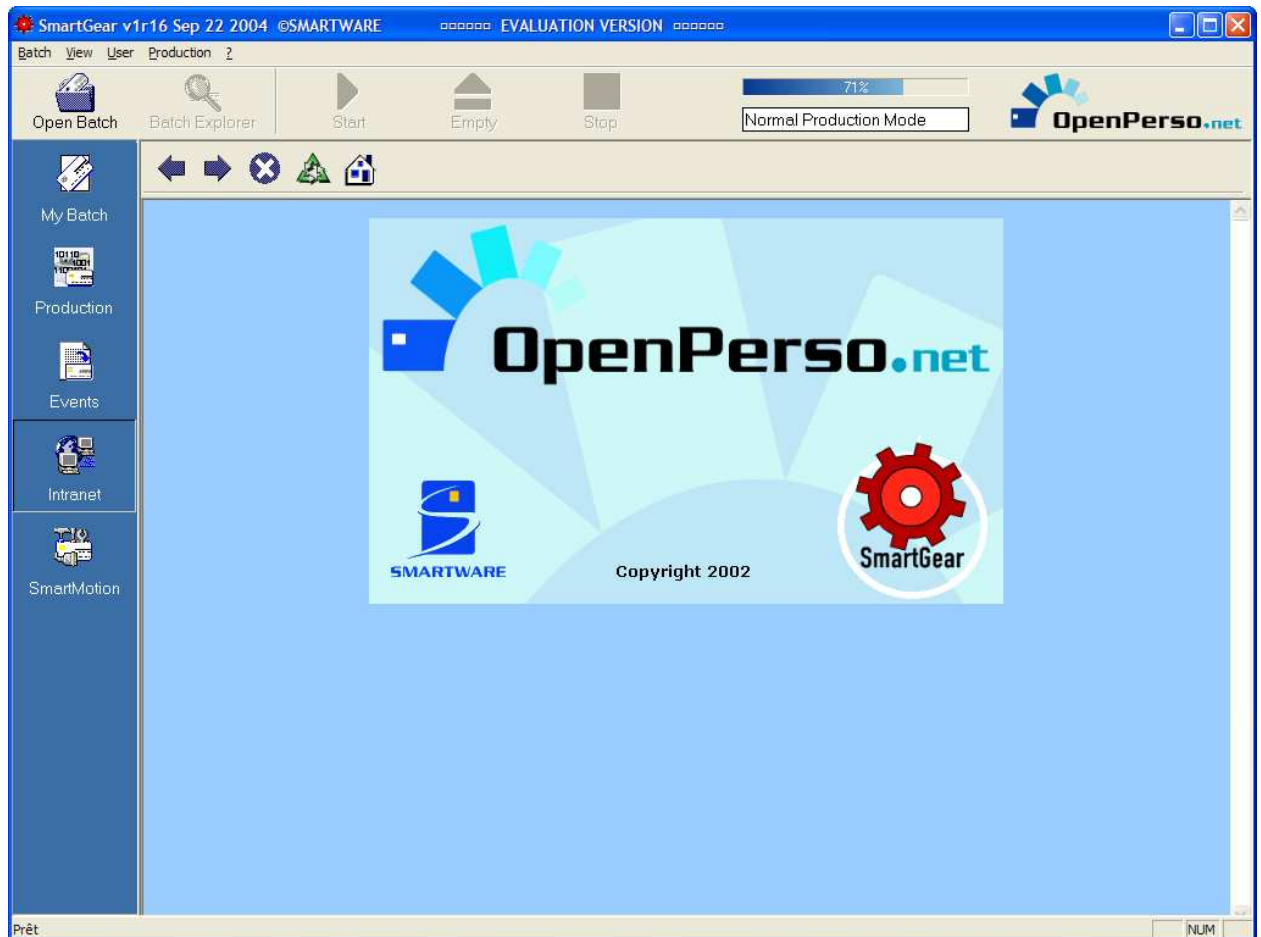
Date	Hour	Source	Error	Message	User Name
2004/09/22	10:43:28	SmartGear		Open batch (Gsm1912)	SmartGear
2004/09/22	10:43:29	SmartGear		Production start	SmartGear
2004/09/22	10:49:35	SmartGear		Production pause	SmartGear
2004/09/22	10:49:38	SmartGear		Production resume	SmartGear
2004/09/22	10:56:04	SmartGear		Production pause	SmartGear
2004/09/22	10:57:44	SmartGear		Empty machine	SmartGear
2004/09/22	10:58:04	SmartGear		End of production	SmartGear
2004/09/22	10:58:05	SmartGear		Close batch (Gsm1912)	SmartGear

Prêt

Intranet view

The intranet view is used to display the html page defined in the SmartGear.ini file.

This page can reference any internet or intranet site and can be used to provide to the operator some html production report forms, batch information, cards previews, etc ...



2.4 The SMARTGEAR.INI file

The **SMARTGEAR.INI** file contains configuration and initialization data for SMARTGEAR. By convention, it is placed in the \SMARTGEAR\BIN directory. **It is not mandatory and default values can be used.**

The file is structured into 3 sections: [PROJECT], [MCM] and [USERINTERFACE].

[PROJECT]

WorkDir= Project search directory (\SMARTGEAR\PROJECT\) by default

[MCM]

CardCtxSize= Size of card context in bytes (8192 bytes by default)

Simul = X, Operating mode (X=0: Simulation with pause before each new transfer, X>0 Simulation with pause for X milliseconds before each new transfer, X=<empty> Machine production mode). The simulation mode can also be activated by starting SMARTGEAR using "-sX" as command line argument.

TaskEndTimeOut = <Value in second default 15s> Wait TaskEndTimeOut second before displaying "Task still running" after an ABORT.

PaccommDllPathName = Pacops.dll full path. (Default : .\pacops.dll)

PACCOMVerbose = Pacops.dll verbose level.(Default : 0x00)

WaitForSGTermination = Specify the time in milliseconds to wait for the unloading of SmartGear before re-starting a new instance (Default 2500).

WaitForInitialization = Specify the time in milliseconds to wait for the SmartGear environment initialization before starting the process(Default 2500).

[MACHINE]

MachineName = Machine name (used only with Workshop Monitor module)

MachineModel = Machine model (used only with Workshop Monitor module)

[USERINTERFACE]

Language=Language used for messages extracted from SPMWIN.MSG file

0 : English by default

1 : French

2 : Spanish

3 : German

DisableEmptyButton= 0: Enable Empty button (default), 1: Disable Empty button

DisablePauseButton= 0 Enable Pause button (default), 1 Disable Pause button

[INTRANET_PAGE]

Page = SmartGear intranet page full path.

2.5 The PROJECT.INI file

The **PROJECT.INI** file contains configuration and initialization information for SMARTGEAR.

It is specific to each project for a given machine configuration. By convention, it is placed in the directory \SMARTGEAR\PROJECT\<Project name> of each project.

The file is structured into 3 sections: [PROJECT], [JOBUSER_X] and [STATION_X].

It is also possible to add one or several sections incorporating keys specific to a project so as to configure a project in the simplest manner. It is then up to the application to recover the values associated with these new keys.

2.5.1 Section [PROJECT]

The section includes a given number of keys:

Key	Description
Name	Project name
Comment1.. Comment5	Comments (5 lines) displayed on the various SMARTGEAR screens
ProjectDLLName	Name of project management DLL
JobDLLName	Name of job management DLL
InputDataPath	Directory and search extension for data files
InputDataExt	
JobExt	Extension for job management files
ImportPath	Directory and extension for data files to be imported
ImportExt (3 characters)	
ExportPath	Directory and extension for data files to be exported
ExportExt (3 characters)	
LogPath	Directory and extension for production follow-up files
LogExt (3 characters)	
ShowDebugLevel	Filter for displaying operating traces in hexa on 4 digits (e.g. FF01). Mcm_ShowDebugTransfer 0x0001 Mcm_ShowDebugStep 0x0002 Mcm_ShowDebugLoad 0x0004 Mcm_ShowDebugTask 0x8000 Mcm_ShowDebugStepCtx 0x0008
Language	Language code used by the application to display messages 0: English (by default) 1 : French 2 : Spanish 3 : German In addition to the messages displayed by SMARTGEAR, it is up to the application to display its specific messages according to the language code.
BatchBitmap	Card preview (*.bmp) full path.
CardCtxSize	Used for indicating, in bytes, the size of each card context (by default the size is 8192 bytes).

JobListDisplay

0: No selectable Batch. Go directly to the production view

1: Default Mode. Display Batch list

EnableBatchButton

0: Hide Open/Close batch button if JobListDisplay=0 (Default)

1: Show Open/Close batch button if JobListDisplay=0

2.5.2 Section [PERSO]**BadLevelAutoEmpty**

0: No automatic empty machine(Default)

1: Automatic empty machine if bad level is reached on a station

2.5.3 Sections [JOBUSER_X]

These sections define the user buttons used on the **MyBatch view** screen.

These buttons are not directly visible on the screen but can be used as context menu by right clicking anywhere on the batch list.

In standard mode, the system manages 12 buttons (display and call of the desired function when the button is pressed). Each has a section and predefined location.

Accordingly, the button defined in [JOBUSER_1] will be the first. [JOBUSER_2] will be placed just underneath, and so on...

Each [JOBUSER_X] section contains the following keys:

Name Name displayed in the button (the & activates the key stroke)

FuncName Function name called when the button is clicked

AccessLevel 0 : Default. Available only when no batch is open

1: Available if the production is not running (even if a batch is already opened)

2: Always available (even if a production is running)

SecurityLevel Security level (32 bit field)

A function bearing the **FuncName** must be present in the job management DLL (JOB.DLL) for the button to appear. It is therefore important to comply with case sensitivity for the **FuncName** function name.

2.5.4 Sections [JOBUSERMENU_X]

These sections define the additional menus a user can insert in the main menu bar of SmartGear.

Each menu is associated with a particular function of the job.dll.
A total of 20 menu items can be managed.

Each section represents a main menu and all its sub-menus and has a predefined location in the SmartGear menu bar.

Accordingly, the user menu defined in the [JOBUSERMENU_1] section will be displayed as the first user menu just after the "Production" menu of SmartGear.
The menu defined in the [JOBUSERMENU_2] will be placed just after and so on ...

Each [JOBUSERMENU_X] section contains the following keys:

Name: Label of the main Menu

MenuItemY_Label: Label of the "Y" Submenu

MenuItemY_Func: Function name called when the submenu "Y" is clicked

MenuItemY_SecurityLevel: Security level (32bits field)

...

MenuItemZ_Label: Label of the "Z" Submenu

MenuItemZ_Func: Function name called when the submenu "Z" is clicked

MenuItemZ_SecurityLevel: Security level (32bits field)

A function bearing the **MenuItemX_Func** name must be present in the job management DLL (JOB.DLL) for the sub menu to appear. It is therefore important to comply with case sensitivity for the **MenuItemX_Func** function name.

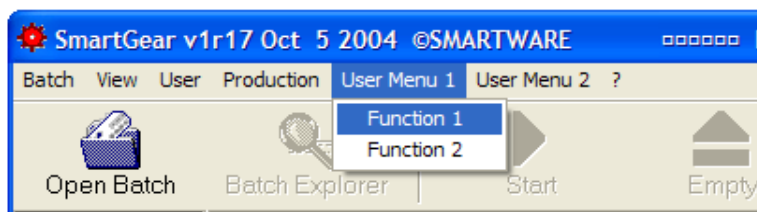
Example:

```
[JOBUSERMENU_1]
Name= User Menu 1
MenuItem1_Label=Function 1
MenuItem1_Func=Job_Func1

MenuItem2_Label=Function 2
MenuItem2_Func=Job_Func2

[JOBUSERMENU_2]
Name= User Menu 2
MenuItem1_Label=Function 3
MenuItem1_Func=Job_Func3

MenuItem2_Label=Function 4
MenuItem2_Func=Job_Func4
```



2.5.5 Sections [STATION_X]

The ([STATION_0] to [STATION_X]) sections are used for configuring all the machine stations.

Key	Description
StepType	Type of station among those recognized
	TRINP Input conveyor LOADR Loader MAGRW Magnetic track reading-encoding SCCAR Chip encoding drive wheel SCENC Encoding head/chip test PRJET Ink jet marking PRLAS Laser marking FLIPO Flip FLIEJ Flip/Eject EJECT Simple eject UNLDR Unloader TROUT Exit conveyor TREND Transfer end FREE Step free (value by default)
	XXXXX Other name for COMPUTE type steps
Mode	Step operating mode among MACHINE FREE STATIC DYNAMIC COMPUTE INDEXED MULTIPRINT
	The step operating modes are described in the following
Activated	ENABLE/DISABLE to (de)activate the step (DISABLE by default)
StepId	Identifier to differentiate between several steps of the same type on the machine
PCAStatusInputMin	Minimum card status to enter into step (for PCA)
PCAStatusInputMax	Maximum card status to enter into step (for PCA)
PCPStatusInputMin	Minimum card status to enter into step (for PCP)
PCPStatusInputMax	Maximum card status to enter into step (for PCP)

StatusCardGood	Output status if card good (generally 01 _h to 2F _h)
StatusCardBad	Output status if card bad (generally 30 _h to 3E _h)
BadLevelMin	Number of bad consecutive cards after automatic pausing of machine
BadLevelMax	Maximum number of bad cards before automatic pausing of machine
BadLevelMinIndexed	Number of bad consecutive cards after automatic pausing of machine for IndexedStation
BadLevelMaxIndexed	Maximum number of bad cards before automatic pausing of machine for IndexedStation
LoadSequence	Used when machine is not equipped with wheel but with series of in-line customizing heads to manage the head descent sequencing (grouped or not grouped)
PCACmdLine	Command line for logic control PC – refer to machine documentation
PCPCmdLine	Command line for customization PC
DLLName	Name of DLL in charge of this step (if necessary)
Indexed	Number of sub-steps if Mode=INDEXED
DLLNameIndexed	Name of DLL if Mode=INDEXED
StationActivated	ENABLE/DISABLE to (de)activate the sub-steps <u>E.g.</u> Indexed= 12 StationActivated= EDEDEDEDED Station 1, 3, 5, 7, 9, 11 are activated

2.6 General context SMARTGEAR

The general context of SMARTGEAR contains a set of environment variables (Directory path, Window pointers, Context, DLL Handle...). A pointer to the context is systematically as used as argument at the points of entry to the specific DLLs of a project. It is the ideal means of getting to know or conveying global data within the project itself.

```
typedef struct {
    ushort          Version;
    TCHAR           SpmWorkDir [_MAX_DIR]; // Launch directory
    TCHAR           SpmCurrDir [_MAX_DIR];  // Current directory
    spm_ProjectType * pProject; // Pointer to project context
    spm_JobType     * pJob; // Pointer to Job context

    spm_AccessControlType * pAccessControl; // Pointer to access control context

    uchar           * pSysInfo; // Internal use only
    uchar           * pInfo; // Pointer for specific allocation by application

    spm_StepCfgType * pSpmStepCfg; // Pointer to configuration

    BOOL McmContextShift; // Card transfer stop indicator
                        // between two stations on restart
    BOOL McmRecalibration; // Transfer recalibration error indicator
    BOOL McmSimul;        // Operating mode indicator
                        // TRUE if SMARTGEAR is started with option -s
    BOOL McmEmptyMachine; // Indicator that machine dumping is underway
    BOOL McmEmptyTransfer; // Indicator that transfer dumping is underway
    BOOL McmManualMode;   // Box status indicator
                        // check "Manual mode" on Running screen
    BOOL McmRunning;      // Status indicator (TRUE if card production
                        // is underway)

    // Windows Handle
    HANDLE          hInstance; // Program instance handle
    HANDLE          hWndMain;  // Main window handle
    HANDLE          hWndCurr;  // Current window handle
    HWND            hWndProjectCommentList;
    HWND            hWndJobList;
    HWND            hWndJobCommentList;
    // Running
    HWND            hWndStepList; // Handle ListBox Message Station
    HWND            hWndMachineList; // Handle ListBox Message Machine
    // DLL Handles
    HINSTANCE        hLibProject;
    HINSTANCE        hLibJob;
    // New Version 3.25 10/04/1998
    spm_MachineStatusType * pMachineStatus; // Pointer to context
                        // of machine status
    spm_SoftwareInfoType * pSoftwareInfo;    // Pointer to context
                        // of information about software
    spm_UserInterfaceType * pUserInterface; // Pointer to context
                        // about user interface

    BOOL bDCOMEnable;
    DWORD dwDCOMVerbose;
    BOOL McmMaintenance;
    HINSTANCE        hLibPaccomm; /* Paccomm library */
}
```

```
|BOOL bBadLevelAutoEmpty; // v1r18 Automatic Empty machine if badlevel is
|reached
|
|// SmartGear v2.00 - System dll
|spm_SysType * pSys;
|HINSTANCE      hLibSys; /*Sys library*/
|} spmwin_CtxType;
```

2.6.1 spm_AccesControlType

This structure enables the application to store information about the current operator:

```
typedef struct {  
    char OperatorName[64]; // Operator name  
    DWORD SecurityLevel; // Access level (32 bit field)  
    BOOL Administrator; // Administrator Indicator(TRUE/FALSE)  
    uchar * pSysInfo;  
    uchar * pInfo; // Pointer for specific allocation by application for  
    char Password [64]; // Password  
    // fields not provided for as standard by SMARTGEAR  
} spm_AccessControlType;
```

2.6.2 spm_MachineStatusType

This structure informs the application of information about the machine status:

```
typedef struct {
    uchar * pSysInfo;
    uchar * pInfo; // Pointer for specific allocation by application for
                  // fields not provided for as standard by SMARTGEAR
    BOOL    fTransferEmpty; // Indicator of Empty transfer on restart
    uchar    MachineCfgPath [_MAX_PATH]; // MACHINE.CFG file path
    uchar    MachineId [SpmMachineIdMax]; // Name of connected machine
    ushort   MachineAverageSpeed; // Average production speed
                                   // (returned by machine)
    ushort   MachinePeakSpeed; // Maximum speed reached during production
                                   // (returned by machine)

    // New v3r25c
    uchar    * pPCAStatus; // PCA status (see define in spm.h)
    uchar    * pPCPStatus; // PCP status (see define in spm.h)
    uchar    CmdLine [256];
    // New v3r25d
    uchar    TransferDirection; // Card transfer direction
    uchar    PCAMessageType; // Type of last message (Error or Warning)
    uchar    PCAMessage [SpmPCAMessageMax]; // Text of last message
                                           // from machine

    // New v3r45a
    BOOL    fManualStationPositioning;
    // New v3r50f
    DWORD    TaskEndTimeOut; //default=10 (in seconds !) SMARTGEAR.INI [MCM]-
    >TaskEndTimeOut
    // new v3r60b
    DWORD    TransferRequest;
    DWORD    TransferOk;

    /* v4r00 */
    DWORD    PCPStatus;
    DWORD    PCAStatus;

    DWORD    Mode; /* Machine Mode */

    // COM/DCOM Communication (SmartMotion)
    void *pPacObj; // Paccom object */
    paccom_FuncType PaccomFunc; /* Paccom function pointer struct */

    // Statistics
    DWORD    ProductionStartTime;
    DWORD    ProductionEndTime;
    DWORD    CycleStart;
    DWORD    StationCycleStart;
    DWORD    CycleDuration;
    DWORD    CycleDurationTotal;

    char    ProjectName [_MAX_PATH];
    char    JobName      [_MAX_PATH];

    DWORD    PCAAbort;
} spm_MachineStatusType;
```

2.6.3 spm_SoftwareInfoType

This structure indicates to the application the general information about SMARTGEAR. More particularly it involves the card context size.

```
typedef struct {
    uchar * pSysInfo;
    uchar * pInfo; // Pointer for specific allocation by application for
                  // fields not provided for as standard by SMARTGEAR
    ushort Version; // Version of SMARTGEAR e.g. 0325h
    ushort Release; // Release example 'c'

    uchar    Copyright [128];
    uchar    Author [64];
    DWORD    CardCtxSize; // Card context size
    // New v3r35a
    DWORD    SimulWait;
    // New v3r35c
    BOOL    fTraceJob; // Activate Trace for project
    FILE    * hTraceJob;
    ushort TraceCtxTag;
    BOOL    fTraceProject; // Activate Trace for project
    FILE    * hTraceProject;
} spm_SoftwareInfoType;
```

2.6.4 spm_UserInterfaceType

This structure indicates to the application information about the user interface.

```
#define SpmWin_LanguageEnglish 0
#define SpmWin_LanguageFrench 1
#define SpmWin_LanguageSpanish 2

// JOB User Defined Button v3r25
#define SpmWin_UserButtonLabelMax 32+1
#define SpmWin_JobUserMax 12

typedef struct {
    int DlgItem;
    char Label [SpmWin_UserButtonLabelMax];
    char FuncName [64];
    char DLLName [_MAX_PATH];
    FARPROC fProc;
    DWORD SecurityLevel;
} spm_UserButtonType;

// User Interface v3r25
typedef struct {
    uchar * pSysInfo;
    uchar * pInfo;
    ushort Language;
    spm_UserButtonType * pJobUserButton;
    BOOL fJobListDisplay; // v3r35B default = TRUE
    char RunningDlgName[64]; // v3r50 default="IDD_RUNNING"
} spm_UserInterfaceType;
```

2.6.5 spm_StepCfgType

There are two methods in a project for configuring the machine steps. The first consists in using the sections **[STATION_X]** of the *PROJECT.INI* file. The second consists in initializing by program the table of structures pointed to by **pStepCfg** of the type *spm_StepCfgType* in the general SMARTGEAR context. For more details, refer to section 2.9 Station management DLL (STEP).

2.7 Project management DLL

2.7.1 Presentation

This DLL is a way of customizing access to a given project. The presence of the DLL or a special entry point is not mandatory.

The general context of SMARTGEAR contains a pointer to a structure called **pProject**. This structure of the **spmProject_Type** type contains various fields of which a major part is initialized from the *[PROJECT]* section of the *PROJECT.INI* file.

```
typedef struct {
    ushort Version;

    /* [PROJECT] section from PROJECT.INI File */
    TCHAR Name [_MAX_PATH];
    TCHAR Path [_MAX_DIR];

    TCHAR Comment [SpmProjectCommentLineMax] [SpmProjectCommentMax];
    char InputDataPath    [_MAX_DIR];
    char InputDataExt     [_MAX_EXT+1];
    char ImportPath       [_MAX_DIR];
    char ImportExt        [_MAX_EXT+1];
    char ExportPath       [_MAX_DIR];
    char ExportExt        [_MAX_EXT+1];
    char JobExt           [_MAX_EXT+1];
    char LogPath          [_MAX_DIR];
    char LogExt           [_MAX_EXT+1];

    TCHAR ProjectDLLName [_MAX_PATH];
    TCHAR JobDLLName      [_MAX_PATH];
    ushort ShowDebugLevel;

    // Internal variable
    TCHAR ProjectINIFile [_MAX_PATH];
    spm_StepCfgType * pStepCfg; // Step configuration

    uchar * pSysInfo;

    uchar * pInfo; // Specific Project context
} spm_ProjectType
```

2.7.2 Specific project context

The fields not provided for can be allocated through a structure specific to the project in the **pInfo** variable of the **pProject** structure.

Example

```
// PROJECTX.H
typedef struct {
    char DataServerName [256] ;
    char UserId [32];
    ...
} app_ProjectCtxType;

...
// PROJECT.DLL Allocation of structure
DllExport Err_Type Project_Start (
    SpmWin_CtxType * pC
)
{
    Err_Type Err;
    // Allocation of specific project context
    Err=syst_MemAlloc (&pC->pProject->pInfo, size of(app_ProjectCtxType));
    ...
}

DllExport err_Type Project_End (
    SpmWin_CtxType * pC
)
{
    Err_Type Err;
    // Releasing of specific project context
    Err= syst_MemFree (pC->pProject->pInfo, size of(app_ProjectCtxType));
    ...
}

// JOB.DLL
DllExport Err_Type Job_Start (
    SpmWin_CtxType * pC
)
{
    app_ProjectCtxType *pPC = (app_ProjectCtxType *)pC->pProject->pInfo;

    // Enter PassWord
    // Use of structure fields
    // Connect (pPC->UserId, PassWord, pPC->DataServerName) ;
}
```

2.7.3 Standard points of entry

2.7.3.1 *Project_Start ()*

If there is one, it is called when a project is selected from the project list (ProjectList).

As a general rule, we use this point of entry to:

- Allocate project or job context
- Carry out an access check specific to the project
- Connect to a database
- ...

```
|DllExport Err_Type Project_Start (
|    SpmWin_CtxType * pC, // SMARTGEAR Context
|);
```

NOTE: If *Project_Start* returns a 0 error code then the *JobList* window will be displayed; otherwise it will remain in the *ProjectList* window.

2.7.3.2 *Project_End ()*

This function is called when the project closes (closing of the *JobList* screen).

As a general rule, this point of entry is used to :

- Liberate the project or job contexts
- Close the sessions with the database
- ...

```
|DllExport Err_Type Project_End (
|    SpmWin_CtxType * pC, // SMARTGEAR Context
|);
```

2.8 Job management DLL (JOB)

2.8.1 Presentation

This DLL is used for customizing the management of the jobs for a given project.

The general context of SMARTGEAR contains a standard structure called **pJob**. This structure of the type **spmJob_Type** contains the various needed for most of the customization applications.

```
typedef struct
{
    ulong    Size;           // Sizeof this structure
    ushort Version;         // Version
    ushort Status;          // See below

    TCHAR Name  [_MAX_PATH];
    TCHAR Path  [_MAX_PATH];
    TCHAR Comment [5]      [SpmJobCommentMax];

    FILETIME InputFileLastWrite;
    time_t TimeCreate;      // Job Date+Time Creation
    time_t TimeStart;       // Job Date+Time First Time production
    time_t TimeEnd;         // Job Date+Time Terminated

    ushort MachineSpeedMax; // Maximum machine speed reached during
production
    ushort MachineSpeedAverage; // Average machine speed

    // Batch Counter
    DWORD CardTotal;        // Total Card in batch
    DWORD CardGood;
    DWORD CardBad;
    DWORD CardMissing;
    DWORD CardRemake;

    DWORD RecPointer;
    char  Reserved [4096];
    char  Rfu;

    uchar * pSysInfo;       // Internal use only

    uchar * pInfo;          // Specific Job context

    HANDLE hInputFile;
    HANDLE hJobFile;
    HANDLE hLogFile;

    DWORD Modify;           // for Job List refresh
    char BatchBitmap[255]; //BatchBitmap displayed in SmartGear production View.
} spmjob_Type;
```

2.8.2 Specific job context (Job)

The fields that were not provided for can be allocated through a structure specific to the project in the **pInfo** variable of the **pJob** structure.

Example

```
// PROJECTX.H
...
typedef struct {
    char PrintingSide; // 'F'ront or 'B'ack
    char CustomerName [32];
    ...
} app_JobCtxType;

// PROJECT.DLL
DllExport Err_Type Project_Start (
    SpmWin_CtxType * pC
)
{
    Err_Type Err;

    Err=syst_MemAlloc (&pC->pJob->pInfo, size of(app_JobCtxType));
    ...
}
DllExport Err_Type Project_End (
    SpmWin_CtxType * pC
)
{
    Err_Type Err;

    Err=syst_MemFree (pC->pJob->pInfo, size of(app_JobCtxType));
    ...
}

// JOB.DLL
DllExport Err_Type Job_Start (
    SpmWin_CtxType * pC
)
{
    app_JobCtxType *pJC = (app_JobCtxType *)pC->pJob->pInfo;

    pJC->PrintingSide=xxx; // xxx: get from Job Definition File
    strcpy (pJC->CustomerName, xxx); // xxx: Get from Job Definition File
    ...
}
```

2.8.3 Standard points of entry (Management/Display Mode)

The presence of the following functions is optional in the JOB.DLL.

2.8.3.1 *Job_ListCreate (...)*

This function is called once when a *JobList* window is created.

As a general rule, it is used for customizing the titles of the column in the job list.

```
|DllExport Err_Type Job_ListCreate (
|    SpmWin_CtxType * pC, // SMARTGEAR Context
|    HWND hList          // Job List Windows Handle
|);
```

2.8.3.2 *Job_List (...)*

This function is called the first time after a call on *Job_ListCreate* and each time a user button or the <START> button is activated.

```
|DllExport Err_Type Job_List (
|    SpmWin_CtxType * pC, // SMARTGEAR Context
|    HWND hList          // Job List Windows Handle
|);
```

2.8.3.3 *Job_ListSelect (...)*

This function is called when the <START> button or a button defined by the user in the JOBLIST window is clicked.

```
|DllExport Err_Type Job_ListSelect (
|    SpmWin_CtxType * pC, // SMARTGEAR Context
|    HWND hList          // Job List Windows Handle
|);
```

2.8.3.4 *Job_Start (...)*

This function is called when the <START> button or a button defined by the user is clicked, or on a double-click on a job in the JOBLIST window.

```
DllExport Err_Type Job_Start (
    SpmWin_CtxType * pC, // SMARTGEAR Context
);
```

NOTE: if *Job_Start* returns a 0 error code then the job production window will be displayed. Otherwise it remains on the JOBLIST window.

2.8.3.5 *Job_End (...)*

Click on the <ABORT> button or close the production window.

```
DllExport Err_Type Job_End (
    SpmWin_CtxType * pC, // SMARTGEAR Context
);
```

2.8.4 Functions defined by user (Buttons Nos. 1 to 12)

The functions defined in PROJECT.INI have to be defined in JOB.DLL for the associated buttons to appear.

2.8.5 Functions defined by user (User menu Nos 1 to 20)

The functions defined in PROJECT.INI have to be defined in JOB.DLL for the associated user menu items to appear.

2.8.6 Standard points of entry (Production Mode)

The presence of the following functions is optional in the JOB.DLL.

2.8.6.1 *Job_RunningStart*

Click on the <START> button in the PRODUCTION screen. This will enter the production mode.

```
|DllExport Err_Type Job_RunningStart (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.2 *Job_RunningEnd*

This function is called on exit from the production mode (all the station tasks are stopped). When the <ABORT> button is clicked in the PRODUCTION screen there are no more cards in the machine.

```
|DllExport Err_Type Job_RunningEnd (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.3 *Job_RunningTransferStart*

This function is called each time a card transfer starts.

```
|DllExport Err_Type Job_RunningTransferStart (  
|    C  
|);
```

2.8.6.4 *Job_RunningTransferEnd*

This function is called after each card transfer end.

```
|DllExport Err_Type Job_RunningTransferEnd (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.5 *Job_RunningEmptyTransfer*

This function is called when the <EMPTY TRANSFER> button is clicked.

```
|DllExport Err_Type Job_RunningEmptyTransfer (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.6 *Job_RunningEmptyMachine*

This function is called when the <EMPTY MACHINE> button is clicked.

```
|DllExport Err_Type Job_RunningEmptyMachine (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.7 *Job_RunningButtonStart*

This function is called when button <START> is clicked.

```
|DllExport Err_Type Job_RunningButtonStart (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.8 *Job_RunningButtonPause*

This function is called when button <PAUSE> is clicked.

```
|DllExport Err_Type Job_RunningButtonPause (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.9 *Job_RunningButtonAbort*

This function is called when button <ABORT> is clicked.

```
|DllExport Err_Type Job_RunningButtonAbort (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.10 *Job_RunningButtonEmptyMachine*

This function is called when button <EMPTY MACHINE> is clicked.

```
|DllExport Err_Type Job_RunningButtonEmptyMachine (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

2.8.6.11 Job_RunningPCAMessage

This function is called whenever a message (Warning or Error) is sent by PCP (Machine).

```
|DllExport Err_Type Job_RunningPCAMessage (  
|    SpmWin_CtxType * pC, // SMARTGEAR Context  
|);
```

pC->pMachineStatus-> PCAMessageType and pC->pMachineStatus->PCAMessageType contains detailed message error.

2.9 Station management DLL (STEP)

2.9.1 Presentation

A DLL can be associated with each active station of the machine if a specific transfer has to be carried out on this station. Although it is not mandatory to declare a DLL on the **MACHINE** type stations, it is on the other types of stations other than the **FREE** stations.

2.9.2 Card status management

Each card in the machine has a customization status which changes gradually as the card moves from one machine station to another.

For a station to accept a card as good (and process it), the status of the card entering this station must fall between the limits **PCAStatusInputMin** and **PCAStatusInputMax** defined in the **PROJECT.INI** file.

After processing, the station assigns a new output status to the card. If the card is good after processing, the card status takes the value **StatusCardGood** and inversely, **StatusCardBad** if it is bad. The two output values are thus defined in the **PROJECT.INI** file.

Accordingly, each station must be configured to accept or refuse the card depending on the output status of the previous station.

2.9.3 General station context

Each station has a context allocated to it dynamically by SMARTGEAR. This context breaks down into two data groups. In the first group is the static information obtained from the *PROJECT.INI* file. The second group contains dynamic information resulting from machine operation, such as counters, card context pointers, etc.

```

struct {
    ushort StepNum;          /* 0 -> 254, SpmNumLast */
    char StepType [SpmStepTypeMax];
    ushort Mode;             /* SpmStatic/SpmDynamic */
    ushort Indexed;          /* 0 -> n */
    ushort StepId;           /* 0 -> n */
    ushort LoadSequence;    /* SpmYes/SpmNo */
    char PCACmdLine [SpmStepCmdLineMax];
    ushort Activated;        /* SpmEnable/SpmDisable */
    uchar PCAStatusInputMin;
    uchar PCAStatusInputMax;
    uchar StatusCardGood;
    uchar StatusCardBad;
    ushort BadLevelMin;     /* Bad consecutive card */
    ushort BadLevelMax;     /* Max bad card */
    uchar Show;
    ushort ShowDebugLevel;

    char DLLName [_MAX_PATH];
    char DLLNameIndexed [_MAX_PATH];
    DWORD StationActivated;  /* 1 bit per station 31 -> bit 0

    uchar PCPStatusInputMin;
    uchar PCPStatusInputMax;
    char PCPCmdLine [SpmStepCmdLineMax];
    uchar Rfu;

    /* Internal */
    DWORD StepActiveNum;
    HANDLE StaticReadyEvent;
    HANDLE StaticActiveEvent;
    HANDLE TaskRunning;
    uchar * pSysInfo;
    uchar * pInfo;
    uchar StepStatus;

    SpmCardType * pCard;    /* Card on this step
    SpmCardType * pCardPrev; /* for Card Sequence/Remake

    SystTaskType Task;

    DWORD TotalCardIn;
    DWORD TotalCardOut;
    DWORD TotalCardGood;
    DWORD TotalCardBad;
    DWORD TotalCardBadConsecutive;

    // Indexed Step Managing
    struct spm_StepCfgType * pStepIndexed;

```

```

    ushort StepIndexedCur;
    DWORD   IndexedUniqueId;
    ushort IndexedActivated;
    BOOL EndOfData;
    HINSTANCE hDLL;
} spm_StepCfgType, * spm_StepCfgRef;

```

2.9.4 Station specific context

The fields not provided for can be allocated through a station-specific structure in the *pInfo* variable of the structure.

Example

```

File READER.INI
[READER_01]
  IPAddress=123.456.789.123
  Encoding=Y

[READER_02]
  IPAddress=123.456.789.124
  Encoding=Y
...

// STATIONX.C

typedef struct {
    char IPAddress [20];
    uchar Encoding [2];
    ...
} Step_InfoType;

DllExport Err_Type Step_Start (
    SpmWin_CtxType * pC,
    spm_StepCfgType * pSC
)
{
    Err_Type Err;
    Step_InfoType * pSI

    Err=syst_MemAlloc (&pSC->pInfo, sizeof(Step_CtxType));
    if (Err)
        return Err;
    ...
    pSI=(Step_InfoType *)pSC->pInfo
    // Get IPAddress from configuration File
    {
        uchar Section [64];
        sprintf (Section,"READER_%u", (pSC->StepId+1));
        GetPrivateProfileString(
            Section, // points to section name
            "IPAddress", // points to key name
            "", // points to default string
            pSI->IPAddress, // points to destination buffer
            sizeof(pSI->IPAddress), // size of destination buffer
            "READER.INI" // points to initialization filename
        );
    }
}

```

```

        GetPrivateProfileString(
            Section, // points to section name
            "Encoding", // points to key name
            "Y", // points to default string
            pSI->Encoding, // points to destination buffer
            sizeof(pSI->Encoding), // size of destination buffer
            "READER.INI" // points to initialization filename
        );
    }
}

DllExport Err_Type Step_End (
    SpmWin_CtxType * pC,
    spm_StepCfgType * pSC
)
{
    Err_Type Err;

    Err=syst_MemFree (pSC->pInfo, size of(Step_CtxType));
}

```

2.9.5 Station operating modes

The station **Mode** determines the synchronization mode between PCP and PCA. Depending on the chosen **Mode**, the Step_xxx functions associated with the station DLL are called at different times. Therefore, it is essential to choose the correct mode according to the station type (*StepType*).

2.9.5.1 MACHINE Mode

This **Mode** is reserved for the stations managed by the PCA in the same way as the loaders (*LOADR*), unloaders (*UNLDR*), the flip (*FLIPO*), the flip-eject (*FLIEJ*), the eject (*EJECT*) functions. As a general rule, these stations do not have an associated DLL. That is why a task is created by SMARTGEAR only when a DLL name is specified in *DLLName*.

2.9.5.2 STATIC Mode

This **Mode** is reserved for the chip or test encoding head type stations (*SCENC*) or for laser applications (*PRLAS*). It is designed to synchronize the call for the Step_xxx functions as a function of the following logic.

Creation of task by SMARTGEAR

Localization and loading of DLL

Recovery of points of entry points

Call for Err=Step_Start(...)

if Err!= 0 then stop all current tasks

❶ (Body of task)

Wait for start of card transfer

Call for Step_CardTransfer (...)

Wait for end of card transfer

If StepStatus in range (PCASStatusInputMin ↔ PCASStatusInputMax)

Call for Err=Step_CardStart (...).

Wait for station ready (Head down, Laser door closed...)

if Err==0

Call for Err=Step_CardRun (...).

if Err==0

Call for Err=Step_CardEnd (...).

Else

Call for Err=Step_CardOutOfRange()

return to ❶

Call for Err=Step_End (...)

unload DLL

Stop task

2.9.5.3 DYNAMIC and COMPUTE Modes

This *Mode* is reserved for stations of the type: contact-free card reader, magnetic card reader-encoder (MAGRW), bar code reader, or those carrying out totally software processing (such as updating or search in database). Although, physically, these are stations installed in the machine, they are only known to the PCP. Therefore, they do not wait for events from the PCA. The distinction between the DYNAMIC and COMPUTE mode is purely "intellectual". The former is distinguished from the latter by the fact that the DYNAMIC type tasks are associated with physical stations (like a reader-encoder for magnetic cards or a contact-free card reader when COMPUTE is associated with processing tasks).

Creation of task by SMARTGEAR

Localization and loading of DLL

Recovery of points of entry points

Call for Err=Step_Start(...)

if Err!= 0 then stop all current tasks

❶ (Body of task)

Wait for start of card transfer

Call for Step_CardTransfer (...)

Wait for end of card transfer

If StepStatus in range (PCASStatusInputMin ↔ PCASStatusInputMax)

Call for Err=Step_CardStart (..).

if Err==0

Call for Err=Step_CardRun (..).

if Err==0

Call for Err=Step_CardEnd (..).

Else

Call for Err=Step_CardOutOfRange()

return to ❶

Call for Err=Step_End (...)

Unload DLL

Stop task

Note: The fact of putting your code into one of the Step_CardStart (), Step_CardRun (), Step_CardEnd () functions is not really important but does clarify the reading of the program.

2.9.5.4 MULTIPRINT Mode

This *Mode* is reserved for multi-line ink jet head type stations (PRJET). The number of lines and their respective positions are specified in the PCA command line (PCACmdLine: **Lx** refer to the MPR manual).

Creation of task by SMARTGEAR

Localization and loading of DLL
Recovery of points of entry points

Call for Err=Step_Start(...)
if Err!= 0 then stop all current tasks

❶ (Body of task)

Wait for start of card transfer
Call for Step_CardTransfer (...)
Wait for end of card transfer

If StepStatus in range (PCASStatusInputMin ↔ PCASStatusInputMax)

Call for Err=Step_CardStart (..).

❷ (Loop on printout of **Lx** lines)

if Err==0

Call for Err=Step_CardRun (..)

return to ❷

if Err==0

Call for Err=Step_CardEnd (..).

Else

Call for Err=Step_CardOutOfRange()

return to ❶

Call for Err=Step_End (...)

Unload DLL

Task Stop

NOTE : pSC->StepIndexedCur can be used as an index to know which line is currently being printed (0 : First line, 1: second line ...)

2.9.5.5 INDEXED Mode

This *Mode* is reserved for chip encoding wheel type stations (SCCAR). It synchronizes the call for the Step_xxx functions according to the following logic.

Creation of task by SMARTGEAR

Localization of DLL

Recovery of points of entry

Call for Err=Step_Start(...)

if Err!= 0 then stop all current tasks

Wait until all the sub-tasks STEPINDEXED are ready
(Step_Start returns 0 see next paragraph)

❶ (Body of task)

Wait for start of card transfer

Call for Step_CardTransfer (...)

Wait for end of card transfer

If StepStatus in range (PCASStatusInputMin ↔ PCASStatusInputMax)

Call for Err=Step_CardIn (..).

if Err==0

Call for Err=Step_CardOut (..).

Else

Call for Err=Step_CardOutOfRange()

return to ❶

Call for Err=Step_End (...)

Unload DLL

Stop task

2.9.5.6 STEPINDEXED Mode

This *Mode* is reserved for internal use by SMARTGEAR. It controls the stations associated with a station of the INDEXED type as a function of the following logic.

Creation of task by SMARTGEAR

Localization and loading of DLL

Recovery of points of entry points

Call for Err=Step_Start(...)

if Err!= 0 then stop all current tasks

❶ (Body of task)

 If StepStatus in range (PCASStatusInputMin ↔ PCASStatusInputMax)

 Wait for activation to start

 Call for Err=Step_CardStart (..)

 Wait for station ready (head down)

 if Err==0

 Call for Err=Step_CardRun (..).

 if Err==0

 Call for Err=Step_CardEnd (..).

 Else

 Call for Err=Step_CardOutOfRange()

return to ❶

Call for Err=Step_End (...)

Unload DLL

Stop task

2.9.5.7 FREE Mode

This *Mode* is reserved for the stations not used by the machine. No tasks are created for this type of station by SMARTGEAR.

2.9.6 Standard entry points

Each station DLL has 6 standard points of entry. These functions are called automatically by SMARTGEAR when the machine cycle runs and according to the operating mode of the station.

These are optional points of entry and therefore, will only be called if they are present in your DLL.

They all have the two same parameters:

The general pointer to the SMARTGEAR context	SpmWin_CtxType *pC
The pointer to the station configuration	spm_StepCfgType*pSC

A distinction is made between two points of entry:

- 2 for starting and stopping the task supporting this DLL.
- 4 for processing to be carried out on the card.

2.9.6.1 Step_Start (Task initialization)

This point of entry is called once only on station initialization. It can be used to allocate specific station contexts, initialization of variables, opening a file or establishing a connection towards a peripheral.

```

DllExport Err_Type Step_Start (
    SpmWin_CtxType *pC,
    spm_StepCfgType*pSC)
{
}

```

2.9.6.2 Step_CardTransfer

This function is called when the card transfer starts.

CAUTION: The rotation of the card and status contexts is only carried out after a complete and successful card transfer. At this precise moment, the pSC->pCard pointer of a station points to the card being moved from one station to another. If you want to anticipate these actions, you must use pCard for the previous station.

```

DllExport Err_Type Step_CardTransfer (
    SpmWin_CtxType *pC,
    spm_StepCfgType*pSC)
{
}

```

2.9.6.3 Step_CardStart

Step_CardStart is a way of preparing for the processing to be carried out in Step_CardRun.

```

DllExport Err_Type Step_CardStart (
    SpmWin_CtxType *pC,
    spm_StepCfgType*pSC)
{
}

```

2.9.6.4 Step_CardRun

This function is called during the cycle. This is where the main processing is performed.

```

|DllExport Err_Type Step_CardRun (
|    SpmWin_CtxType *pC,
|    spm_StepCfgType*pSC)
|{
|}

```

2.9.6.5 Step_CardEnd

This function is called during the cycle. (It is used more for aesthetic aspects of task subdivision).

```

|DllExport Err_Type Step_CardEnd (
|    SpmWin_CtxType *pC,
|    spm_StepCfgType*pSC)
|{
|}

```

2.9.6.6 Step_CardOutOfRange

This function is called during the cycle is StepStatus is not in the range of PCAStatusInputMin and PCAStatusInputMax.

```

|DllExport Err_Type Step_CardOutOfRange (
|    SpmWin_CtxType *pC,
|    spm_StepCfgType*pSC)
|{
|}

```

2.9.6.7 Step_End

This function is called before a task end. This is where the allocated contexts are released, the files or the connections closed.

```

|DllExport Err_Type Step_End (
|    SpmWin_CtxType *pC,
|    spm_StepCfgType*pSC)
|{
|}

```

2.9.7 Specific entry points for INDEXED stations

The *INDEXED* stations (see the **PROJECT.INI** file) have two different points of entry than the other stations. This type of station is found on the chip encoding carousel (SCCAR).

2.9.7.1 Step_CardIn

Step_CardIn is automatically called when a new card meeting the input criteria appears at the station.

```

DllExport Err_Type Step_CardIn(
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSCIn // Previous card of this station
)
{
}

```

2.9.7.2 Step_CardOut

Step_CardOut must return into pSCT->StepIndexedCur the number of the station to be positioned before the transfer of the next card.

```

DllExport Err_Type Step_CardOut (
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSCT
)
{
}

```

2.10 Card context

2.10.1 Presentation

The purpose of the card context is to convey for a given card a number of different types of information. This information is stored in the form of a chained list of blocks of the type TLV (Type, Length, Value). This list is managed by SMARTGEAR through an API.

The card context is an essential operational element of SMARTGEAR. Indeed, SMARTGEAR is based on a structure of independent tasks with respect to one another and therefore are unaware of their mutual existence. **This mechanism is a way of rendering the station management DLLs totally modular and independent.**

A context is allocated by a card which may be present in the machine. For a standard MPR12 type machine comprising 17 stations for transfer and 12 encoding stations, 29 card contexts are dynamically allocated when the production mode starts up.

The application itself defines the nature and structure of the stored information. As a general rule, a structure using C language is used. The section entitled « *Example of use* » details an example of how to use the context.

2.10.2 Size

The context size is « virtually » unlimited. The default size of a card context is 8 K-bytes. However, the size of this context can be defined for each project by specifying it in the *PROJECT.INI* file section *[PROJECT].CardCtxSize*.

2.10.3 Back-up of machine context

When machine production with cards inside is suspended, all the card contexts and the information about the current job is saved in a file `\\SMARTGEAR\\CFG\\MACHINE.CFG` file.

2.10.4 Associated C functions

The functions associated with context management return error codes. For more details, refer to the file include ***cardctx.h***.

2.10.4.1 ***CardCtx_Write ()***

Used for adding a new block to a card context.

```
#include <Spmwin.h>

err_Type CardCtx_Write (
    spm_CardType *pSC,    /* Pointer to card context */
    ushort Type,          /* Type of block */
    ulong Length,         /* Data length */
    void * pIn            /* Data */
);
```

NOTE: It is possible to update a context block as long as the length of the data to be updated is less than or equal to the length of the data currently to be found in this context block.

2.10.4.2 CardCtx_Read ()

Used for reading a block already existing in a card context.

```
#include <SMARTGEAR.h>
err_Type CardCtx_Read (
    spm_CardType *pCC,      /* Pointer to card context */
    ushort Type,            /* Type of block */
    void *pOut,             /* Data reception buffer*/
    ulong *pLength          /* Pointer to desired length
                           or NULL for stored data length */
);
```

NOTE : Initialize *pLength before calling CardCtx_Read to read *pLength first bytes of card context block. (if *pLength is greater than stored length then *pLength is updated in return) or use NULL to read all block (in that case pOut buffer should be big enough).

```
ulong Length;
err_Type Err;
uchar Buffer [1024];

Length=MAXDWORD;

Err=CardCtx_Read (
    pSC->pCard,      /* Pointer to card context */
    2,              /* Type of block */
    Buffer,          /* Data reception buffer*/
    &Length          /* Pointer to desired length
                   */
);
or
Err=CardCtx_Read (
    pSC->pCard,      /* Pointer to card context */
    2,              /* Type of block */
    Buffer,          /* Data reception buffer*/
    NULL            /* Read all block */
);
CardCtx_Copy ()
```

Used for duplicating an existing card context block towards another (from version 3.35a only).

```
#include <spmwin.h>
err_Type CardCtx_Copy (
    spm_CardType *pCCDest, /* Pointer to target card context */
    spm_CardType *pCCSrc, /* Pointer to source card context */
    ushort Type           /* Type of block */
);
```

2.10.4.3 CardCtx_Info ()

Used for recovering information about the card context (allocated size, busy, number of blocks...) see structure *CardCtx_HeaderType* in the file include **cardctx.h**.

```
#include <spmwin.h>
err_Type CardCtx_Info (
    spm_CardType *pCC,      /* Pointer to card context */
    CardCtx_HeaderType * pCH /* Pointer to information structure*/
);
```

2.10.4.4 CardCtx_Clear ()

Used for clearing all the context blocks.

NOTE: This function is called automatically by SMARTGEAR when a new card enters the machine transfer.

```
#include <Spmwin.h>
err_Type CardCtx_Clear (
    spm_CardType *pCC      /* Pointer to card context */
);
```

2.10.4.5 Example of use

```
/* PROJECT: DEMO
   FILE      : APP.H
*/
...
#define app_CtxChipData 0x0001

typedef struct {
    uchar  Name      [32];
    uchar  CardNumber [20];
    uchar  ADMKey     [8];
    ...
} app_CtxChipData;

#define app_CtxChipPersoStatus 0x0010
typedef struct {
    uchar  Err;
    uchar  ChipSerialNumber [8];
    uchar  PinCode          [4];
    uchar  SW1;
    uchar  SW2;
    ...
} app_CtxChipPersoStatus;

#define app_CtxLaserPersoStatus 0x0011
typedef struct {
    err_Type Status; /* 0 -> OK, if not error */
    ...
} app_CtxChipLaserStatus;
```

NOTE: It is not mandatory (as demonstrated in the example) to number the context types sequentially, starting from 0001.

```

/* PROJECT: DEMO
   FILE      : STATION(x).C (Smart Card Encoding)
*/
#include <Spmwin.h>
#include <app.h>

...

DllExport err_Type Step_CardStart (
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSC
)
{
    err_Type Err=0;
    app_ChipDataType ChipData;

    /* Lecture Contexte app_CtxChipData */
    Err = CardCtx_Read (pSC->pCard, app_CtxChipData, &ChipData, NULL);
    ...

    /* Transmission of data to reader...*/

    return Err;
}

DllExport err_Type Step_CardRun (
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSC
)
{
    err_Type Err=0;
    app_CtxChipPersoStatus ChipPersoStatus;

    /* Start of personalization
       and filling with PersoStatus structure in return */

    /* "{|Writing of result into context app_CtxChipPersoStatus */
    Err = CardCtx_Write (
        pSC->pCard,
        app_CtxPersoStatus,
size of (app_CtxChipPersoStatus)
        &PersoStatus);
    return Err;
}

```

```

/*    PROJECT: DEMO
      FILE      : STATION(x+n).C (Laser engraving)
*/
#include <SMARTGEAR.h>
#include <app.h>

...

DllExport err_Type Step_CardStart (
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSC
)
{
    err_Type Err=0;
    app_ChipPersoStatus ChipPersoStatus;

    /* Reading of Context app_CtxChipPersoStatus */
    Err = CardCtx_Read (pSC->pCard, app_CtxChipPersoStatus, &PersoStatus,
NULL);

    ...

    /* Transmission of data to serial No. laser marking station
    ChipPersoStatus.ChipSerialNumber ... */

    return Err;
}

DllExport err_Type Step_CardRun (
    SpmWin_CtxType * pC,
    spm_StepCfgType* pSC
)
{
    err_Type Err=0;
    app_LaserPersoStatus LaserPersoStatus;

    /* Starting of laser firing
    and filling of LaserPersoStatus structure in return
    LaserPersoStatus.Err)*/

    /* Writing of result into context app_CtxLaserPersoStatus */
    Err = CardCtx_Write (
        pSC->pCard,
        app_CtxLaserPersoStatus,
size of (app_CtxLaserPersoStatus)
        &LaserPersoStatus);

    return Err;
}

```

2.11 Spm30 standard API

2.11.1.1 *Msg_Stack*

Header file: *Spmmsg.h*

```
| ErrType WINAPI Msg_Stack (HWND hWnd, /* Windows Handle (ListBox) */
|      char * fmt, /* Format String same as printf (...) */
|      ...);      /* Variable arguments */
```

`hWnd=pC->hWndMachineList`

Send a message in the SmartGear Machine List

2.11.1.2 *UserButtonXXX*

Header file: *SpmUser.h*

The UserButtonXXX APIs can be used to send a “click” message to SmartGear GUI Production buttons.

```
| err_Type User_ButtonStart (SpmWin_CtxType * pC);
```

Click on SmartGear Start/Pause/Resume button.

```
| err_Type User_ButtonEmptyMachine (SpmWin_CtxType * pC);
```

Click on SmartGear “Empty” button

```
| err_Type User_ButtonAbort (SpmWin_CtxType * pC);
```

Click on SmartGear “Stop” button.

```
| err_Type User_ButtonOpenBatch (SpmWin_CtxType * pC);
```

Click on SmartGear “OpenBatch” button.

```
| err_Type User_ButtonCloseBatch (SpmWin_CtxType * pC);
```

Click on SmartGear “CloseBatch” button.

```
| err_Type User_ButtonLogin (SpmWin_CtxType * pC);
```

*

Display the SmartGear login window for selecting a new project.

2.12 SmartGearUI API

The *SmartGearUI.dll* offers several interfaces to manage some displayed element of the SmartGear GUI such as the batch bitmap or the intranet view.

Theses interfaces can be called from any station DLL that links *SmartGearUI.lib* and includes the right header files.

2.12.1.1 Intranet view management

Header file : *sgui_Intranet.h*

2.12.1.1.1 sgui_IntranetPageSet

```
|err_Type sgui_IntranetPageSet(char * URL);
```

Set the HTML page to be displayed in the intranet view.

2.12.1.1.2 sgui_IntranetHomeSet

```
|err_Type sgui_IntranetHomeSet(char * URL);
```

Define the « Home » HTML page of the intranet view

2.12.1.1.3 sgui_IntranetBack

```
|err_Type sgui_IntranetBack();
```

Go back to the previous HTML page

2.12.1.1.4 sgui_IntranetForward

```
|err_Type sgui_IntranetForward();
```

Go forward to the next HTML page

2.12.1.2 Batch Bitmap management

Header file: *sgui_BatchBitmap.h*

2.12.1.2.1 sgui_BatchBitmapSet

```
|err_Type sgui_BatchBitmapSet(char * BatchBitmap);
```

Set the complete path of the batch bitmap to be displayed in the production view

2.12.1.3 SmartGear Control Interface

Header file: *sgui_SmartGearCtrl.h*

2.12.1.3.1 sgui_ProductionModeSet

```
|err_Type sgui_ProductionModeSet(char * ProductionMode);
```

Set the text to be displayed in the “Production Mode” control (SmartGear toolbar)

2.12.1.3.2 Sgui_ButtonStatusSet

```
|err_Type sgui_ButtonStatusSet(ulong ButtonStatus)
```

2.12.1.4 SmartMotion Interface

Header file: *SmartmotionUI.h*

2.12.1.4.1 sgui_StationMaintenanceProcess

```
|err_Type sgui_StationMaintenanceProcess(long PCPStatus,  
    long StepNum,  
    long ProcessNum,  
    long ButtonState,  
    char * Arg)
```

2.12.1.4.2 sgui_SetPCPStatus

```
|err_Type sgui_SetPCPStatus(UINT PCPStatus)
```

Set SmartMotion Production Mode.

PCPStatus=TRUE -> Maintenance Mode

PCPStatus=FALSE-> Normal production mode

SMARTGEAR interface library

2.12.2 Writing conventions

To improve the legibility and modularity of software in C, it is necessary to comply with a few programming conventions.

2.12.2.1 Modules and Symbols

The functionalities of the same family are grouped together into « Modules ». Each module is defined by a protocol file with respect to the other modules: MODULE.H.

Each of these modules is located using a « body » file having the same MODULE.C name.

All the external symbols (type, functions, etc.) of the modules, and those symbols alone, are grouped together in the MODULE.H prototype file.

All the external symbols are named by the module name and underscored by:

Module_Symbol

2.12.2.2 Errors

To obtain clear error indications, *the error codes must be unique both in the same application and with respect to the operating system.*

- The type `err_Type` is defined in ERR.H.

Code 0 indicates "no error".

The source of the error must be identified clearly. Each module is given a section of N error codes (by default 100 codes) the first of which is defined by the symbol `Module__Err`.

The errors in a given module are therefore numbered:

- `Module__Err +1`
- `Module__Err +2`
- etc...

Each error symbol includes a double underscore.

All the external functions of a module return the same type of error. If other values have to be returned, *they will be returned via an argument pointer.*

2.12.2.3 Function comments

The arguments of the functions are described with the (IN) or (OUT) commands depending on the data transit direction. The absence of a comment is equal to (IN).

The (OUT) arguments are always pointers designating a variable whose value has to be returned. A pointer of the type `(char **)` indicates that a pointer `(char *)` must be returned. A pointer of the type `(char *)` indicates that the content of the table pointed to must be returned.

3 DEVELOPMENT KIT INSTALLATION

This chapter describes how to install KINGSOFT SMARTGEAR Development Kit for Windows.

3.1 Installation procedure

The Setup program provided by SMARTGEAR installs all necessary components.

To run Setup for installation of SMARTGEAR Kit

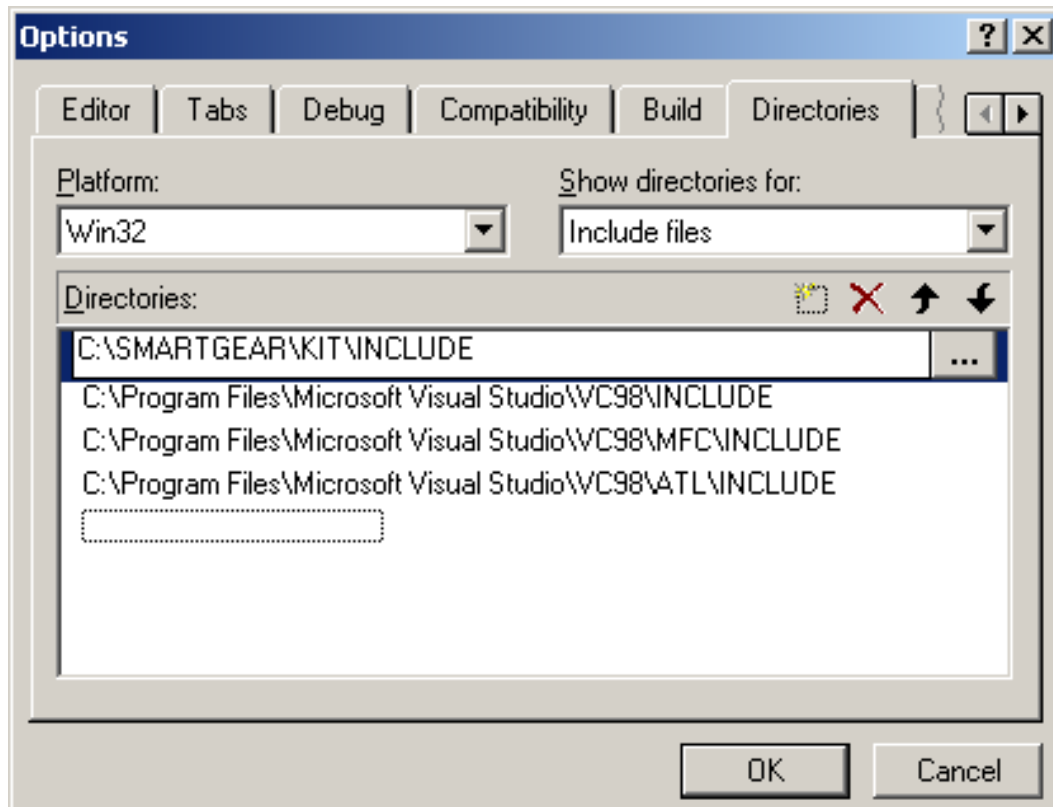
1. Place SMARTGEAR Dev KIT disc in CD reader running under Windows 2000, NT or XP.
2. Use the EXPLORER to run SETUP.EXE
3. Setup prompts you with a Welcome dialog Box that describes the program and lets you continue or exit.

3.2 Visual C++ configuration

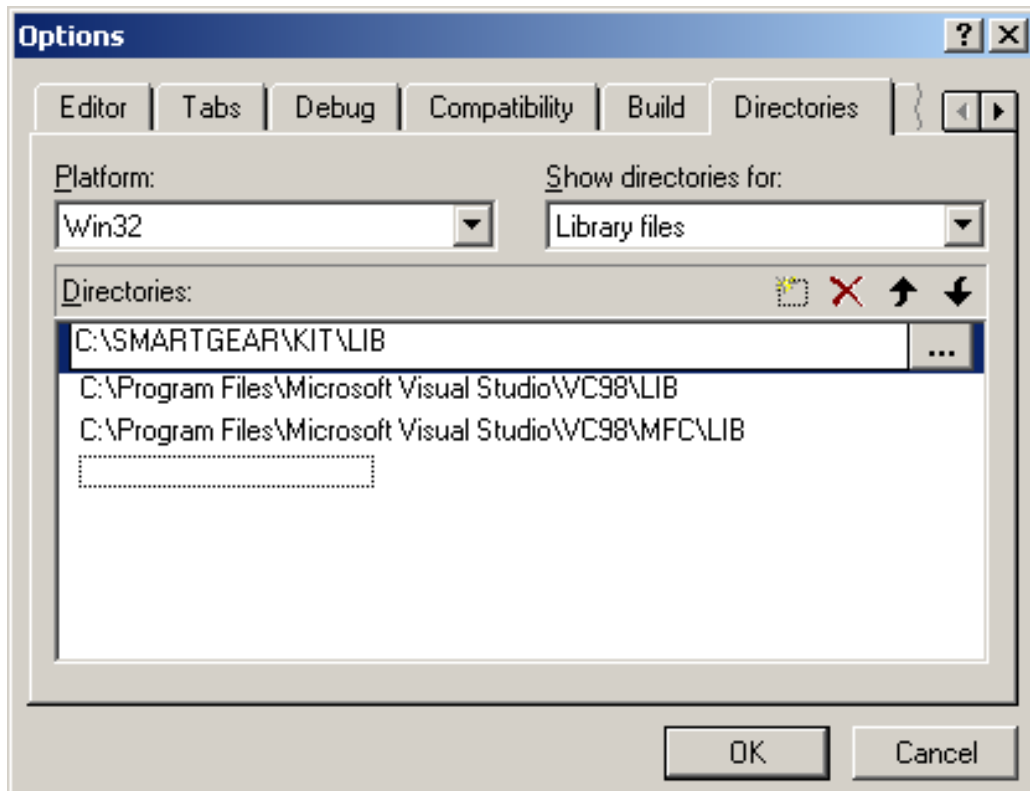
In order to use SMARTGEAR Dev Kit, Microsoft Visual C++ version 6.0 or later must be installed.

After installation you must configure your Microsoft Visual C++ environment, by adding new path from menu TOOLS | OPTIONS

Path for Include files : <YOUR DRIVE>:\SMARTGEAR\KIT\INCLUDE



Path for library files : <YOUR DRIVE>:\SMARTGEAR\KIT\LIB



3.3 Demo project

Demo project provides a example in C language of SMARTGEAR application project.

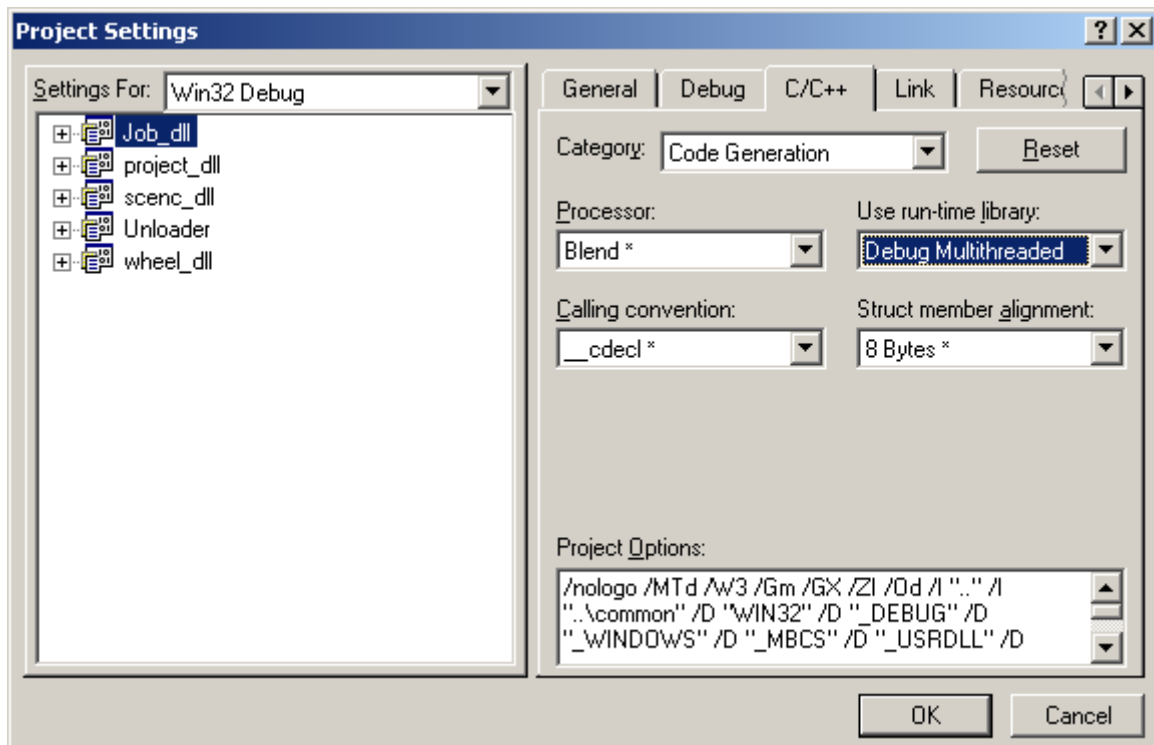
A Microsoft Visual C++ Workspace is located in :

\SMARTGEAR\PROJECT\DEMO\SRC\DEMO\DEMO.DSW

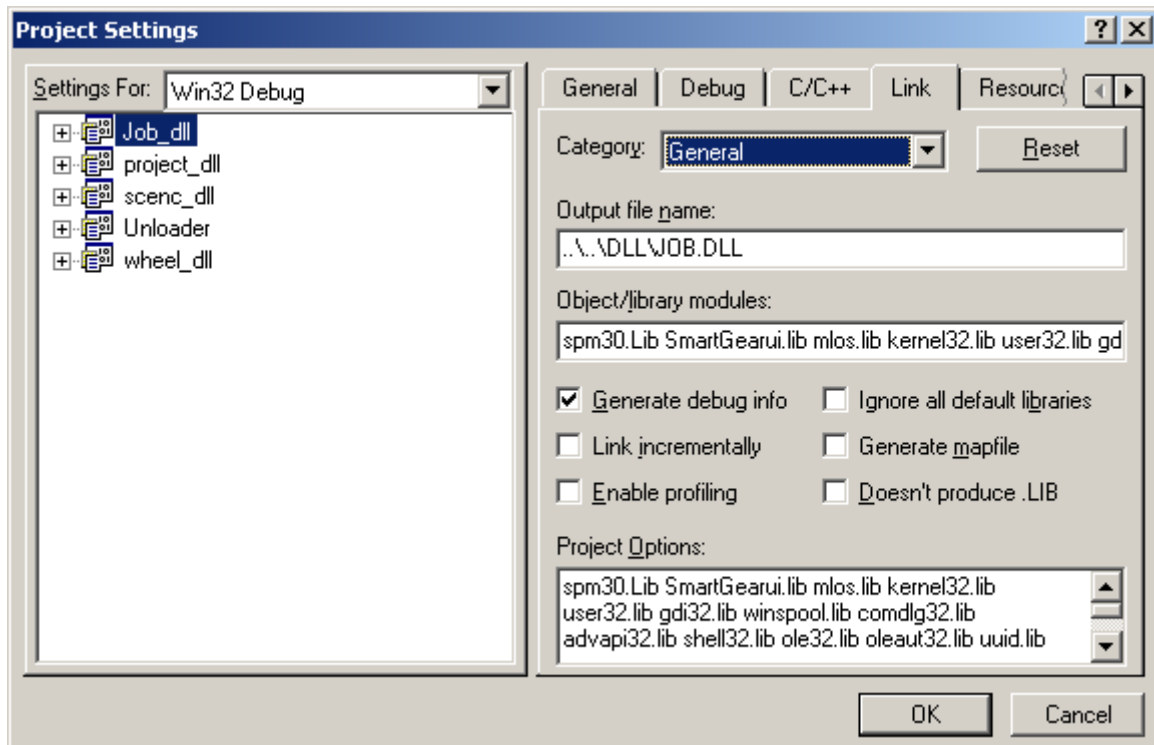
Open **DEMO** Workspace by using File | Open Workspace from Visual C++ Menu.

3.3.1 Demo project building

DLL must always be compiled in Multithreaded (or Debug Multithread for debug session) with struct Member alignment 8 bytes.



Add in Object/library modules **SPM30.LIB**, **SMARTGEARUI.LIB** and **MLOS.LIB**, and renamed the output file name with the full path and name of you DLL (by convention \DLL directory of your PROJECT).

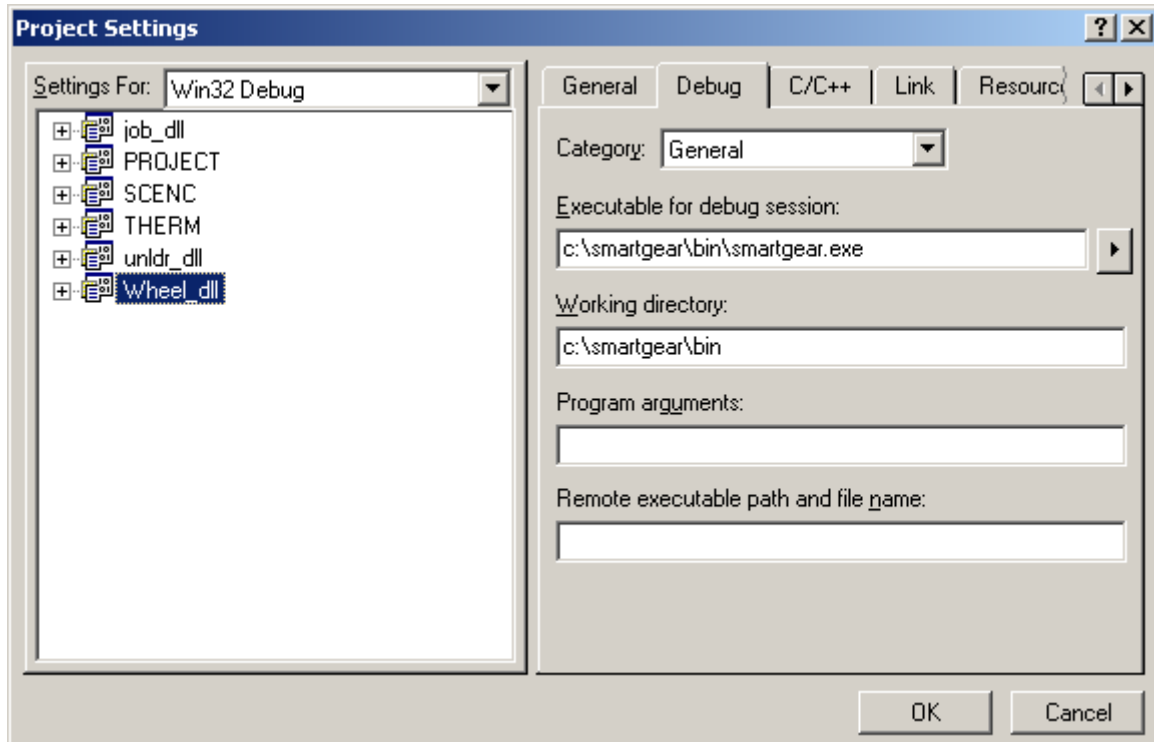


To build a DLL, you can use the build MENU, or in the Project Workspace select a project, click right mouse button and use build option

3.3.2 Debugging DEMO Application project

In the Project Workspace, select the DLL you want to DEBUG by using **Set as Active Project** (right mouse button).

1. Specify the executable for debug session in project settings **SMARTGEAR\BIN\SMARTGEAR.EXE**.
2. Specify Working directory to **SMARTGEAR\BIN** otherwise the MLOS.INI couldn't be located and MLOS network are not launched.



Finally set any necessary breakpoints in source code (**F9**) and then Start Debug with (**F5**)

3.4 SMARTGEAR WORKFLOW

