Content

Introduction

In order to support the powerful PLC modules there are library extensions for the CoDeSys development environment. Any libraries are internal, that means they are implemented in the PLC runtime system. Others are external IEC-Code libraries.

The following libraries are available:

Library name	available	Description	Type
FBESysEncoder.lib	12.2010	Support of incremental encoders	RT
FBESysFile.lib	12.2010	Support of file system	RT
FBESysInterrupt.lib	12.2010	Implementation of Interrupt service routines	RT
FBESysPWM.lib	12.2010	Support of pulse wide modulation	RT
FBESysSmPos.lib	12.2010	Support of stepper motor	RT
FBESysUtil.lib	12.2010	Sundry helpful functions	RT
FBESysCAN.lib	12.2010	Functions for basic CAN support	RT
FBESysCiA405.lib	12.2010	Functions for CiA405 handling	RT
FBESysCiA405_StdBus0	12.2010	Same as FBESysCiA405 but only for standard bus interface 0	RT
FBESysIO.lib	12.2010	Functions for fast IO handling	RT
FBESysMemory.lib	12.2010	Support of EEPROM handling	RT
FBESysNet.lib	03.2011	Support of hipecs webserver	RT
FBESysSerial.lib	12.2010	Support of serial interfaces	RT
FBESysTask.lib	12.2010	Support of Task handling	RT
FBESysTime.lib	12.2010	RTC handling	RT
FBESysUSB.lib	12.2010	Support of USB interfaces	RT
FBESysJ1939.lib	07.2013	Support of J1939 message format for basic CAN	RT
RT: included in PLC Runtime system IEC: external IEC-Code library			

Version History

Library Version	date	Description
1.00 R3	10.11.2011	first release with hipecs serial production start
1.00 R4	15.01.2014	added J1939 Lib and several other functtions

Hardware to Library Cross-Reference

Not all libraries work with each hipecs Module because they have different hardware units for application. This reference shows the libraries, which can be used with each hipecs.

Library name	**hipecs PLC 1010**	**hipecs PLC 1020**	**hipecs PLC 1030**	**hipecs PLC 1210**	**hipecs PLC 1220**	**hipecs PLC 1230**			
FBESysEncoder.lib	X	X	X	X	X	X			
FBESysFile.lib	X	X	X	X	X	X			
FBESysInterrupt.lib	X	X	X	X	X	X			
FBESysPWM.lib	X	X	X	X	X	X			
FBESysSmPos.lib	X	X	X	X	X	X			
FBESysUtil.lib	X	X	X	X	X	X			
FBESysCAN.lib	X	X	X	X	X	X			
FBESysCiA405.lib	X	X	X	X	X	X			
FBESysCiA405_StdBus0	X	X	X	X	X	X			
FBESysIO.lib	X	X	X	X	X	X			
FBESysMemory.lib	X	X	X	X	X	X			
FBESysNet.lib		X	X		X	X			
FBESysSerial.lib	X	X	X	X	X	X			
FBESysTask.lib	X	X	X	X	X	X			
FBESysTime.lib	X	X	X	X	X	X			
FBESysUSB.lib			X			X			

FBESysEncoder.lib

Functions	***Description***
SysEncoder_Clear	This function clears the encoder by resetting its value. New value will be 0.
SysEncoder_Control	This function sets, reads, clears or presets the encoder value at once. Also enabling or disabling (start/stop) of encoder counting is possible with this function.
SysEncoder_Init	Initialize an encoder interface
SysEncoder_Preset	Set new encoder value
SysEncoder_Read16Bit	Read encoder value
SysEncoder_Read32Bit	Read encoder value
SysEncoder_Start	Start encoder
SysEncoder_Stop	Stops encoder
SysEncoder_RegisterCB	Register Call Back: Define an IRQ that shall be called if an over-/underflow of the encoder occurs.

Data types defined for encoder library

```
TYPE type_ENCODER : (
            ENCODER0:= 0,
            ENCODER1:= 1,
            ENCODER2:= 2,
            ENCODER3:= 3
);
END_TYPE
```

Hardware Reference

hipecs PLC1000				
Number of PWM channels	4			
Channel number	0	1	2	3
Channel name	ENCODER0	ENCODER1	ENCODER2	ENCODER3
Input frequency max. (kHz)	100	100	100	100
Input track A	DIN0.0	DIN0.2	DIN0.4	DIN0.6
Input track B	DIN0.1	DIN0.3	DIN0.5	DIN0.7
hipecs PLC1000 offers 3 channels for track A/B encoder and 1 event counter. These channels are special functions on the digital input group 0.				

SysEncoder_Clear

name	**SysEncoder_Clear**		*type*	Function
return value	*type*	BOOL		
		TRUE	*Function ended successfully.*	
		FALSE	*Function skipped.*	
input value 1	*name*	**Encoder**		
	type	type_ENCODER	Selects the hardware related encoder channel by name.	
	value	[Channel name]	*See hardware reference table for valid channel name*	
description	This function clears the encoder by resetting its value. New value will be 0.			

SysEncoder_Control

name	**SysEncoder_Control**			*type*	Function
return value	*type*	DINT			
		[0 .. FFFFFFFF]	*Actual encoder Value.*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
input value 2	*name*	**Enable**			
	type	BOOL	Enables / Disables encoder for counting.		
	value	TRUE	Encoder will be enabled.		
		FALSE	Encoder will be disabled.		
input value 3	*name*	**Clear**			
	type	BOOL	Encoder Clear Flag.		
	value	TRUE	Encoder will be cleared. New Value 0.		
		FALSE	Encoder value will be unchanged.		
input value 4	*name*	**Preset**			
	type	BOOL	Encoder preset flag.		
	value	TRUE	Encoder value will be set to preset value.		
		FALSE	Encoder value will be unchanged.		
input value 5	*name*	**PresetValue**			
	type	DINT	New encoder value		
	value	[0 .. FFFFFFFF]			
description	This function sets, reads, clears or presets the encoder value at once. Also enabling or disabling (start/stop) of encoder counting is possible with this function.				

SysEncoder_RegisterCB

name	**SysEncoder_RegisterCB**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped. Faulty parameters*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
input value 2	*name*	**nPOU_ID**			
	type	INT	Index of interrupt service routine that shall be called at over- or underflow		
	value	INDEXOF(XXX)	By using the CODESYS operator "INDEXOF(My_POU)" you can assign the POU ID		
description	By using this function, you can call an interrupt service routine if an overflow or underflow of the encoder value occurs.				

SysEncoder_Init

name	**SysEncoder_Init**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
input value 2	*name*	**Mode**			
	type	USINT	Operation mode		
	value	0	Encoder channel close/release.		
		1	Track A/B encoder mode.		
		2	Event counter mode.		
input value 3	*name*	**HighRes**			
	type	BOOL			
	value	TRUE	Activates High Resolution Mode		
		FALSE	Low Res Mode		
input value 4	*name*	**InvertDir**			
	type	BOOL	Invertes counting direction		
	value	TRUE			
		FALSE			
input value 5	*name*	**Start**			
	type	BOOL	Starts encoder function		
	value	TRUE	counting starts immediately		
		FALSE	encoder is initialized but doesn't start counting yet		
description	Function must be called once to register encoder in the OS. The inputs for CODESYS are NOT available while encoders are initialized.				

SysEncoder_Preset

name	**SysEncoder_Preset**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped. Channel number is out of range.*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
input value 2	*name*	**PresetValue**			
	type	DINT	New encoder value		
	value	[0 .. FFFFFFFF]			
description	Sets a new value for the encoder channel				

SysEncoder_Read16Bit

name	**SysEncoder_Read16Bit**			*type*	Function
return value	*type*	INT			
	value	[0 .. FFFF]	*Actual encoder value*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
description	read encoder value in 16 bit format				

SysEncoder_Read32Bit

name	**SysEncoder_Read32Bit**			*type*	Function
return value	*type*	DINT			
	value	[0 .. FFFFFFFF]	*Actual encoder value*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
description	read encoder in 32 bit format				

SysEncoder_Start

name	**SysEncoder_Start**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function executed successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
description	Starts counting				

SysEncoder_Stop

name	**SysEncoder_Stop**			*type*	Function
return value	*type*	BOOL			
	value	TRUE			
		FALSE			
input value 1	*name*	**Encoder**			
	type	type_ENCODER	Selects the hardware related encoder channel by name		
	value	{Channel name}	*See hardware reference table channel name*		
description	stops counting				

FBESysFile.lib

Functions	***Description***
SysDirSet	Set a new active directory for file system. All SysFileOpen commands are referred to this directory, if no complete path information is included. Default active directory name is: **a:\usr\**
SysFileClose	Closes an opened file
SysFileCopy	copies a file to a new filename
SysFileDelete	deletes a file
SysFileEOF	checks if the file pointer is on the end of file position
SysFileGetSize	returns file size on a closed file
SysFileGetSizeId	returns file size of an opened file
SysFileOpen	This function returns a handle to the new file. This handle is the reference for all file access commands. If return value = 0, opening of the file failed
SysFileRead	reads data from a file
SysFileSetPos	sets the file pointer to a new position
SysFileWrite	writes data to a file

Hardware Reference

	PLC1010/PLC1210	**PLC1020 / PLC1220**	**PLC1030 / PLC1230**
Number of drives	2	2	3
default internal drive	a:\	a:\	a:\
external drive	c:\	c:\	c:\ / d:\

SysDirSet

name	**SysDirSet**	*type*	Function
return value	*type*	BOOL	
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped.*
input value 1	*name*	**DirName**	
	type	STRING	New active directory
	value	[path]	*specify complete patch with drive letter*
description	This function sets a new active directory for file system. All SysFileOpen commands are referred to this directory, if no complete path information is included. Default active directory name is: C:\USR\		

SysFileClose

name	**SysFileClose**	*type*	Function
return value	*type*	BOOL	
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped.*
input value 1	*name*	**File**	
	type	DWORD	File handle
	value	[1..FFFFFFFF]	*File handle number*
description	This function closes an opened file. File handle will be invalid then.		

SysFileCopy

name	**SysFileCopy**			*type*	Function
return value	*type*	DWORD			
	value	[1..FFFFFFFF]	*Number of copied bytes.*		
input value 1	*name*	**FileDest**			
	type	STRING	File to create new		
	value	[path / filename]	*File name (path)*		
input value 2	*name*	**FileSource**			
	type	STRING	File to read		
	value	[path / filename]	*File name (path)*		
description	Copies a file.				

SysFileDelete

name	**SysFileDelete**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**FileName**			
	type	STRING	File to delete		
	value	[path / filename]	*File name (path)*		
description	Function deletes a file.				

SysFileEOF

name	**SysFileEOF**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*End of file is reached.*		
		FALSE	*End of file is not reached or function skipped.*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
description	Returns if current position is the end of file.				

SysFileGetPos

name	**SysFileGetPos**			*type*	Function
return value	*type*	DINT			
	value	[0..FFFFFFFF]	*Actual Seek-Pointer of file.*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
description	Returns the position of the current file pointer.				

SysFileGetSize

name	**SysFileGetSize**			*type*	Function
return value	*type*	DINT	File size		
	value	[0..FFFFFFFF]	*Number of bytes.*		
input value 1	*name*	**FileName**			
	type	STRING	File to delete		
	value	[path / filename]	*File name (path)*		
description	returns file size of an closed file				

SysFileGetSizeId

name	**SysFileGetSizeId**			*type*	Function
return value	*type*	DINT	File size		
	value	[0..FFFFFFFF]	*Number of bytes.*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
description	returns file size of an opened file				

SysFileOpen

name	**SysFileOpen**			*type*	Function
return value	*type*	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
		[0]	*opening of the file failed*		
input value 1	*name*	**FileName**			
	type	STRING	File to delete		
	value	[path / filename]	*File name (path)*		
input value 2	*name*	**Mode**			
	type	STRING	Acces mode		
	value	[r]	*read*		
		[w]	*write*		
		[a]	*append*		
description	This function returns a handle of the new file. This handle is the reference for all file access commands. If return value = 0, opening of the file failed.				

SysFileRead

name	**SysFileRead**			*type*	Function
return value	*type*	DWORD	Number of read bytes		
	value	[0..FFFFFFFF]	*Bytes read*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
input value 2	*name*	**Buffer**			
	type	DWORD	Address of buffer to save read bytes		
	value	[0..FFFFFFFF]	*Address value*		
input value 3	*name*	**Size**			
	type	DWORD	Size of Bytes that must be read		
	value	[0..FFFFFFFF]	*size value*		
description					

SysFileSetPos

name	**SysFileSetPos**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
input value 2	*name*	**Pos**			
	type	DWORD	Position of the file pointer		
	value	[1..FFFFFFFF]			
description	Set a new position of the file pionter				

SysFileWrite

name	**SysFileWrite**			*type*	Function
return value	*type*	DWORD	Number of written bytes		
	value	[0..FFFFFFFF]	*Bytes written*		
input value 1	*name*	**File**			
	type	DWORD	File handle		
	value	[1..FFFFFFFF]	*File handle number*		
input value 2	*name*	**Buffer**			
	type	DWORD	Address of buffer to write		
	value	[0..FFFFFFFF]	*Address value*		
input value 3	*name*	**Size**			
	type	DWORD	Size of Bytes that must be written		
	value	[0..FFFFFFFF]	*size value*		
description					

FBESysInterrupt.lib

The Library FBESysInterrupt.lib is a Library extension for the CoDeSys PLC runtime system and enables implementation of Interrupt services for IEC61131 applications. It is an internal library; all functions are included in the runtime system.
Each Interrupt channel is assigned to a dedicated interrupt input pin. The interrupts are edge sensitive and may be configured to positive, negative or both transitions at the corresponding interrupt input pin. The interrupt priority may be selected from thirtytwo levels.
An interrupt may not only be activated from hardware signal transitions, but also by IEC61131 application software. This feature enables implementation of program units at different CPU priorities.

The following functions are implemented:

Functions	***Description***
SysInterrupt_ClrRequest	clears IRQ request flag
SysInterrupt_DeleteService	deletes a previously registered IRQ channel. Input can then be used as regular input again
SysInterrupt_Disable	disables an IRQ
SysInterrupt_Enable	enables an IRQ
SysInterrupt_RegService	registers an IRQ in the hipecs operating system
SysInterrupt_SetRequest	sets the IRQ requst flag in the OS to trigger an interrupt without hardware input

Hardware Reference

hipecs PLC10XX						
Available Interrupt Channels	6					
Interrupt Number in CoDeSys	2	3	4	5	6	7
Hardware Input Pin	IN1.2	IN1.3	IN1.4	IN1.5	IN1.6	IN1.7

SysInterrupt_ClrRequest

name	**SysInterrupt_ClrRequest**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	Request successfully cleared	
		FALSE	Function skipped. Channel number is out of range.	
input value 1	*name*	**IrqNr**		
	type	UINT	Interrupt channel	
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins	
description	Clears the interrupt request flag of the chosen interrupt channel.			

SysInterrupt_DeleteService

name	**SysInterrupt_DeleteService**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	Service successfully deleted.	
		FALSE	Function skipped. Channel number is out of range.	
input value 1	*name*	**IrqNr**		
	type	UINT	Interrupt channel	
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins	
description	This function deletes the Interrupt service of a requested interrupt channel. To enable the Interrupt again it must be used the function "SYSINTERRUPT_REGSERVICE" and then "SYSINTERRUPT_ENABLE".			

SysInterrupt_Disable

name	**SysInterrupt_Disable**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	Interrupt channel successfully disabled.		
		FALSE	Function skipped. Channel number is out of range.		
input value 1	*name*	**IrqNr**			
	type	UINT	Interrupt channel		
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins		
description	Disables an interrupt channel for reception of interrupt requests.				

SysInterrupt_Enable

name	**SysInterrupt_Enable**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	Interrupt channel successfully enabled.		
		FALSE	Function skipped. Channel number is out of range.		
input value 1	*name*	**IrqNr**			
	type	UINT	Interrupt channel		
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins		
description	Enables an interrupt channel for reception of interrupt requests. Previously set request bits will be cleared. With registration of the interrupt function, the interrupt keeps still disabled. In order to use this interrupt channel, it must be enabled with this function. The user must take care of the correct handling of registering and enabling of interrupts. This function does not check, whether there is a interrupt task registered to the channel, that should be enabled.				

SysInterrupt_RegService

name	**SysInterrupt_RegService**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	Interrupt channel successfully enabled.		
		FALSE	Function skipped. Channel number is out of range.		
input value 1	*name*	**IrqNr**			
	type	UINT	Interrupt channel		
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins		
input value 2	*name*	**nPOU_ID**			
	type	INT	Number of the program module that must be registered for this interrupt.		
	value		Check the CoDeSys Operator INDEXOF(CallMeFromIrq)		
input value 3	*name*	**IrqPriority**			
	type	UINT	Priority level of the Interrupt channel / 0 : Lowest Priority 32 : Highest Priority **Use with care!!! Priority level 32 is higher than all other interrupt sources. Only for very short interrupt program!**		
	value	[0..X]	Note! Only one irq at the same priority		
	name	**Edge**			
	type	UINT	Enables the active edge for interrupt activation (Both edges may be enabled at the same time)		
	value	[0..3]	Bit0 Enables/Disables interrupt enable on rising edge of input signal at dedicated interrupt pin Bit1 Enables/Disables interrupt enable on falling edge of input signal at dedicated interrupt pin Setting of the bits is interpreted as follows Bitx = 0 Edge disabled, Interrupt is not activated at this transition Bitx = 1 Edge enabled, Interrupt is activated at this transition		
description	Registers a function for the use with the interrupt control system. Registering of program modules as an interrupt task is done with the individual Id of this module. The Id can be checked with the function "INDEXOF" of the runtime system. With registration of the interrupt function, the interrupt keeps still disabled. In order to use this interrupt channel, it must be enabled with function "SYSINTERRUPT_ENABLE".				

SysInterrupt_SetRequest

name	**SysInterrupt_Request**			*type*	Function
return value	*type*	BOOL			
	value	TRUE	Request successfully set		
		FALSE	Function skipped. Channel number is out of range.		
input value 1	*name*	**IrqNr**			
	type	UINT	Interrupt channel		
	value	[2..7]	0..1: reserved for future use. / [2..7] check hardware reference for available Pins		
description	Sets the interrupt request flag of a dedicated interrupt channel. If this channel was previously enabled, the interrupt will be called.				

FBESysPWM.lib

Functions	***Description***
SysPwm_Close	Closes a PWM channel by resetting PWM unit of this channel. Additionally the channel's output will be released. Then it works as in default as standard digital output.
SysPwm_Open	Opens a PWM channel by initialisation a output for using with PWM unit. This function must be called once before using any other PWM functions of the corresponding PWM channel.
SysPwm_SetDuty	Changes the duty cycle of the selected PWM channel.
SysPwm_Start	Starts PWM signal with preset duty cycle.
SysPwm_Stop	Stops PWM signal. PWM channel output goes to passive level.

Hardware Reference

hipecs PLC1000			
Number of PWM channels	11		
Channel number	0 .. 4	5 .. 7	8 .. 10
PWM frequency max. (kHz)	100	100	1
Resolution max. (steps)	10000	10000	10000
Channels uses same base frequency	no	yes	yes
Output	DOUT0.0 to 0.4	DOUT0.5 to 0.7	DOUT1.0 to 0.2
hipecs PLC1000 offers 11 PWM channels. 5 of them have independent clock sources while 3 + 3 use the same base clock. So up to 7 channels can work with different base frequency. Channel 8 to 10 are on the normal speed digital outputs (Out Byte 1). That limits these output frequencies to a maximum of 1 kHz. Do not use higher frequencies for this channels.			

SysPwm_Close

name	**SysPwm_Close**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Channel number is out of range.*	
input value 1	*name*	**Channel**		
	type	UINT	Selects the hardware related PWM channel	
	value	[channel number]	*See hardware reference table for channel number*	
description	This function closes a PWM channel by resetting PWM unit of this channel. The corresponding output may now be used as regular digital output again.			

SysPwm_Open

name	**SysPwm_Open**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Channel number is out of range or a input value was 0.*
input value 1	*name*	**Channel**	
	type	UINT	Selects the hardware related PWM channel
	value	[channel number]	*See hardware reference table for channel number*
input value 2	*name*	**BaseFrequency**	
	type	UDINT	Selects the hardware related PWM frequency.
	value	[1..X]	(Hz) *See hardware reference*
input value 3	*name*	**Steps**	
	type	UINT	Selects the hardware related PWM resolution.
	value	[1..X]	(steps) *See hardware reference*
input value 4	*name*	**InactivePolarityHigh**	
	type	BOOL	Defines output level at PWM stopped or duty cycle 0%.
	value	TRUE	Output level is high
		FALSE	Output level is low
input value 5	*name*	**Option**	
	type	UINT	*Reserved for future use*
	value	-	-
description	This function opens a PWM channel by initialisation a output for using with PWM unit. This function must be called once before using any other PWM functions of the corresponding PWM channel. - The product of BaseFrequency and Steps must be less or equal of 100 MHz. Exceeds the product 100MHz, then the Steps has priority and the resulting frequency is calculated as follow: 100MHz / Steps = PWM-freq. - Not all frequencys are possible. This function calculates and sets PWM frequency as near as possible to the requested. In most cases this result is better with a lower value of Steps.		

SysPwm_SetDuty

name	**SysPwm_SetDuty**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Channel is not open or channel number is out of range.*
input value 1	*name*	**Channel**	
	type	UINT	Selects the hardware related PWM channel
	value	[channel number]	*See hardware reference table for channel number*
input value 2	*name*	**Duty**	
	type	UINT	New duty cycle value.
	value	[0..10000]	(x 0.01%) represents duty cycle as 1/100 %
description	This Function changes the duty cycle of the selected PWM channel. The selected channel must be open. New duty cycles can be set before PWM is started or may be changed while PWM is running.		

SysPwm_Start

name	**SysPwm_Start**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Channel is not open or channel number is out of range.*	
input value 1	*name*	**Channel**		
	type	UINT	Selects the hardware related PWM channel	
	value	[channel number]	*See hardware reference table for channel number*	
description	This function starts PWM signal with preset duty cycle.			

SysPwm_Stop

name	**SysPwm_Start**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Channel is not open or channel number is out of range.*	
input value 1	*name*	**Channel**		
	type	UINT	Selects the hardware related PWM channel	
	value	[channel number]	*See hardware reference table for channel number*	
description	This function stops PWM signal generation. PWM channel output switches to passive level.			

FBESysSmPos.lib

Functions	***Description***
SmPos_ChangeRampPara	This function calculates the RAMP parameter for stepper motor channel via e-function and is only need, if the ramp must recalculated with other parameters or for other RampModes in special cases. At normal it is not necessary to use this function. The ramp will be calculated with the SmPos_Open function in default for normal operation.
SmPos_CloseAxis	This function closes a axis channel by resetting SMP unit of this channel and re-initialization the corresponding output as default digital output.
SmPos_DefHome	Sets the actual AXIS position to home position. For this the pulse counter and actual position value will reset to zero.
SmPos_EncAssign	(For future use / not available this time)
SmPos_EncRelease	(For future use / not available this time)
SmPos_EncStopUse	(For future use / not available this time)
SmPos_EncUse	(For future use / not available this time)
SmPos_GetActualPos	This function returns actual position of an axis.
SmPos_GetActualSettings	This function returns actual setting of several values. (Fstart, Fmax, Fstop)
SmPos_GetDemandPos	This function returns actual defined demand position of an axis.
SmPos_GetDemandVelocity	This function returns actual demand velocity of an axis in %.
SmPos_GetSlip	(For future use / not available this time)
SmPos_IsAxisActive	This function returns the active state of axis. It returns TRUE if axis is active. Checks wether the axis is Active. Please Note: Axis might be halted by function SmPos_SetVelocity or by a Master Axis. For Checking for Movement please use function SmPos_IsAxisMoving
SmPos_IsAxisMoving	This function returns the actual moving state of axis.
SmPos_IsPositionReached	This function returns whether demand position of axis is reached.
SmPos_OpenAxis	This function opens and initializes a stepper motor channel for using. It also calculates the RAMP parameter for this channel via e-function. The corresponding outputs will be connected to the SmPos unit.
SmPos_SetActualPos	
SmPos_SetActualToHome	
SmPos_SetDemandPos	
SmPos_SetOffset	
SmPos_SetVelocity	
SmPos_StartMotion	
SmPos_StopMotion	
SmPos_SyncAxis	
SmPos_UnSyncAxis	

Data types defined for encoder library

```
TYPE SysSmPos_Axis : (
            SMPOS_AXIS0:= 0,
            SMPOS_AXIS1:= 1,
            SMPOS_AXIS2:= 2,
            SMPOS_AXIS3:= 3,
            SMPOS_AXIS4:= 4,
            SMPOS_AXIS5:= 5,
            SMPOS_AXIS6:= 6,
            SMPOS_AXIS7:= 7,
            SMPOS_NOAXIS:= -1
);
END_TYPE
```

```
TYPE SysSmPos_Position : DINT;
```

```
END_TYPE
TYPE SysSmPos_RampMode : (
                                        SMPOS_RAMP_EXP3:= 0
);
END_TYPE
```

Hardware Reference

hipecs PLC1000				
Number of stepper motor channels	4			
Channel number	0	1	2	3
Channel name	SMPOS_AXIS0	SMPOS_AXIS1	SMPOS_AXIS2	SMPOS_AXIS3
Output Clock	DOUT0.0	DIN0.1	DIN0.2	DIN0.3
Output Direction	DOUT0.4	DIN0.5	DIN0.6	DIN0.7
Ramp Length	[10 .. 500]			
Ramp Modes	SMPOS_RAMP_EXP3 → *F = Fstart + (Fmax - Fstart) * (1 - exp(-((3 * i)/RampLength)))*			
All position values [MinPos .. MaxPos]	[-500000000 .. +500000000]			
hipecs PLC1000 offers 4 stepper motor channels. Each channel uses 2 outputs.				
Frequency limitation depending on start frequency. The maximum output clock frequency and the frequency range are depending on the requested/defined start frequency of the stepper motor. Therefore the following values are relevant.				

if Fstart is in range of	resulting ranges of		
	Fstart	Fstop	Fmax
0 .. 99	50 .. 99	50 .. 1450	100 .. 1500
100 .. 399	100 .. 199	100 .. 2950	150 .. 3000
200 .. 399	200 .. 399	200 .. 5950	250 .. 6000
400 .. 799	400 .. 799	400 .. 11950	450 .. 12000
> 800	800 .. 23950	800 .. 23950	850 .. 24000

SmPos_ChangeRampPara

name	**SmPos_ChangeRampPara**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Channel number is out of range or an error occurred.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
input value 2	*name*	**RampMode**		
	type	SysSmPos_RampMode	*Selects the ramp curve mathematical function*	
	value	[Ramp Modes]	*See hardware reference for ramp mode*	
input value 3	*name*	**RampLength**		
	type	UINT	Number of ramp steps	
	value	[Ramp Length]	*See hardware reference for ramp length*	
input value 4	*name*	**FrequencyStart**		
	type	UINT	Start frequency of stepper motor	
	value	[Fstart]	*See hardware reference for start frequency*	
input value 5	*name*	**FrequencyMax**		
	type	UINT	Maximum frequency of stepper motor	
	value	[Fmax]	*See hardware reference for maximum frequency*	

input value 6	*name*	**FrequencyStop**	
	type	UINT	Stop frequency of stepper motor
	value	[Fstop]	*See hardware reference for stop frequency*
description	This function calculates the RAMP parameter for stepper motor channel via e-function and is only need, if the ramp must recalculated with other parameters or for other RampModes in special cases. At normal it is not necessary to use this function. The ramp will be calculated with the SmPos_Open function in default for normal operation.		

SmPos_CloseAxis

name	**SmPos _CloseAxis**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped. Axis channel is not open or channel is out of range.*		
input value 1	*name*	**Axis**			
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name		
	value	[Channel name]	*See hardware reference table for axe channel name*		
description	This function closes a axis channel by resetting SMP unit of this channel and re-initialization the corresponding output as default digital output.				

SmPos_DefHome

name	**SmPos _DefHome**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped. Axis channel is not open or channel is out of range.*		
input value 1	*name*	**Axis**			
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name		
	value	[Channel name]	*See hardware reference table for axe channel name*		
description	Sets the actual AXIS position to home position. For this the puls counter and actual position value will reset to zero.				

SmPos_EncAssign

(For future use / not available this time)

SmPos_EncRelease

(For future use / not available this time)

SmPos_EncStopUse

(For future use / not available this time)

SmPos_EncUse

(For future use / not available this time)

SmPos_GetActualPos

name	**SmPos _GetActualPos**			*type*	Function
return value	*type*	SysSmPos_Position	Returns the actual position of axis. (signed 32 bit value)		
	value	[MinPos .. MaxPos]	*See hardware reference table for position value range*		
input value 1	*name*	**Axis**			
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name		
	value	[Channel name]	*See hardware reference table for axe channel name*		
description	This function returns actual position of an axis.				

SmPos_GetActualSettings

name	**SmPos _GetActualSettings**		*type*	Function
return value	*type*	UDINT	Returns the actual value selected by parameter Selector	
	value	[0 .. FFFFFFFF]	*Value of the selected parameter.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
input value 2	*name*	**Selector**		
	type	UINT	Selects the returned value	
	value	0	*Return value is Fstart (start frequency of stepper motor)*	
		1	*Return value is Fstop (stop frequency of stepper motor)*	
		2	*Return value is Fmax (maximum frequency of stepper motor)*	
description	This function returns actual setting of several values. (Fstart, Fmax, Fstop)			

SmPos_GetDemandPos

name	**SmPos _GetDemandPos**		*type*	Function
return value	*type*	SysSmPos_Position	Returns the actual defined demand position of axis. (signed 32 bit value)	
	value	[FFFFFFFF .. 0 .. 7FFFFFFF]	*Demand position.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function returns actual defined demand position of an axis.			

SmPos_GetDemandVelocity

name	**SmPos _GetDemandVelocity**		*type*	Function
return value	*type*	UINT	Returns the actual defined demand velocity of axis.	
	value	[0 .. 100]	*Demand velocity in %.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function returns actual demand velocity of an axis in %.			

SmPos_GetSlip

(For future use / not available this time)

SmPos_IsAxisActive

name	**SmPos _IsAxisActive**		*type*	Function
return value	*type*	BOOL	Returns the active state.	
	value	TRUE	*Axis is active.*	
		FALSE	*Axis is not active.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function returns the active state of axis. It returns TRUE if axis is active. Checks wether the axis is Active. Please Note: Axis might be halted by function SmPos_SetVelocity or by a Master Axis. For Checking for Movement please use function SmPos_IsAxisMoving			

SmPos_IsAxisMoving

name	**SmPos _IsAxisMoving**	*type*	Function
return value	*type* BOOL	Returns the actual moving state of axis.	
	value TRUE	*Axis is moving now.*	
	FALSE	*Axis is not moving now.*	
input value 1	*name* **Axis**		
	type SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value [Channel name]	*See hardware reference table for axe channel name*	
description	This function returns the actual moving state of axis.		

SmPos_IsPositionReached

name	**SmPos _IsPositionReached**	*type*	Function
return value	*type* BOOL	Returns whether demand position of axis is reached.	
	value TRUE	*Demand position of Axis is reached.*	
	FALSE	*Demand position of Axis is not reached.*	
input value 1	*name* **Axis**		
	type SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value [Channel name]	*See hardware reference table for axe channel name*	
description	This function returns whether demand position of axis is reached.		

SmPos_OpenAxis

name	**SmPos _OpenAxis**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Axis channel is out of range or an error occurred.*
input value 1	*name*	**Axis**	
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name
	value	[Channel name]	*See hardware reference table for axe channel name*
input value 2	*name*	**RampMode**	
	type	SysSmPos_RampMode	*Selects the ramp curve mathematical function*
	value	[Ramp Modes]	*See hardware reference for ramp mode*
input value 3	*name*	**RampLength**	
	type	UINT	Number of ramp steps
	value	[Ramp Length]	*See hardware reference for ramp length*
input value 4	*name*	**FrequencyStart**	
	type	UINT	Start frequency of stepper motor
	value	[Fstart]	*See hardware reference for start frequency*
input value 5	*name*	**FrequencyMax**	
	type	UINT	Maximum frequency of stepper motor
	value	[Fmax]	*See hardware reference for maximum frequency*
input value 6	*name*	**FrequencyStop**	
	type	UINT	Stop frequency of stepper motor
	value	[Fstop]	*See hardware reference for stop frequency*
input value 7	*name*	**SoftStopPulses**	
	type	UINT	Number of steps that to drive with Fstop before motor stops
	value	[0 .. FFFF]	*Number of soft stop steps*
input value 8	*name*	**Tolerance**	
	type	UINT	motor stops when position is in range of demand position +- tolerance
	value	[0 .. FFFF]	*Number of tolerance steps*
input value 9	*name*	**InvertOutputLevel**	
	type	UINT	Defines the output default level
	value	[0 .. 1]	*if value is greater zero, output level is inverted*

description	This function opens and initializes a stepper motor channel for using. It also calculates the RAMP parameter for this channel via e-function. The corresponding outputs will be connected to the SmPos unit.

SmPos_SetActualPos

name	**SmPos _SetActualPos**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Axis channel is not open or out of range.*
input value 1	*name*	**Axis**	
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name
	value	[Channel name]	*See hardware reference table for axe channel name*
input value 2	*name*	**NewActualPos**	
	type	SysSmPos_Position	*New position value*
	value	[MinPos .. MaxPos]	*See hardware reference table for position value range*
description	This function changes the actual position to new value.		

SmPos_SetActualToHome

name	**SmPos _SetActualToHome**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Axis channel is not open or out of range.*
input value 1	*name*	**Axis**	
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name
	value	[Channel name]	*See hardware reference table for axe channel name*
description	This function sets the actual AXIS position to home position. For this the internal pulse counter and actual position value will reset to zero.		

SmPos_SetDemandPos

name	**SmPos _SetDemandPos**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*Function ended successfully.*
		FALSE	*Function skipped. Axis channel is not open or out of range.*
input value 1	*name*	**Axis**	
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name
	value	[Channel name]	*See hardware reference table for axe channel name*
input value 2	*name*	**NewActualPos**	
	type	SysSmPos_Position	*New demand position value*
	value	[MinPos .. MaxPos]	*See hardware reference table for position value range*
description	This function changes the demand position to new value.		

SmPos_SetOffset

name	**SmPos _SetOffset**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
input value 2	*name*	**Offset**		
	type	SysSmPos_Position	*Offset value that will be added to demand position*	
	value	[MinPos .. MaxPos]	*See hardware reference table for position value range*	
description	This function changes the demand position. The offset value is added to the demand position. If demand position exceeds the range [MinPos .. MaxPos] it will be truncated.			

SmPos_SetVelocity

name	**SmPos _SetVelocity**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
input value 2	*name*	**Velocity**		
	type	UINT	*New velocity value*	
	value	[0 .. 100]	*Velocity as %*	
description	This function changes the maximum velocity.			

SmPos_StartMotion

name	**SmPos _StartMotion**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function starts motion of an axis.			

SmPos_StopMotion

name	**SmPos _StopMotion**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function stops motion of an axis.			

SmPos_SyncAxis

name	**SmPos _SyncAxis**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
input value 2	*name*	**MasterAxis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function connect the axis for synchronization to an other axis as so, that the axis goes on moving same speed and direction to the master axis. (Note: mater and slave axis must have same initialization for this function)			

SmPos_UnSyncAxis

name	**SmPos _SyncAxis**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Function ended successfully.*	
		FALSE	*Function skipped. Axis channel is not open or out of range.*	
input value 1	*name*	**Axis**		
	type	SysSmPos_Axis	Selects the hardware related AXIS channel by name	
	value	[Channel name]	*See hardware reference table for axe channel name*	
description	This function release the axis from synchronization of an other axis. The axis then runs with her own parameter.			

FBESys_Util.lib

Functions	***Description***
SysUtil_GetFV	Returns the version number of the internal run time system (example 1234 ==> Version 1.234)
SysUtil_GetSysTime	Returns system time.
SysUtil_GetTargetId	Returns the target identification for CodeSys development environment
SysUtil_LedSet	Sets the lighting state of an programmable LED
SysUtil_GetTaskInfo	Returns information about tasks
SysUtil_GetSerNo	Returns serial number of the hipecs PLC

SysUtil_GetFV

name	**SysUtil_GetFV**			*type*	Function
return value	*type*	UDINT			
		[0...FFFFFFFF]	*Version of internal Firmware. (Example 1234 ==> Version 1.234)*		
input value 1	*name*	**Dummy**			
	type	UINT	reserved.		
	value	[0]	*Set this always zero*		
description	This function returns the Version Number of the internal run time system.				

SysUtil_GetSysTime

name	**SysUtil_GetSysTime**			*type*	Function
return value	*type*	UDINT			
		[0...FFFFFFFF]	*System time*		
input value 1	*name*	**Scale**			
	type	UINT	Time scale ident.		
	value	[0]	*Systemtime in milli seconds*		
		[x]	*Others reserved*		
description	This function returns the system time.				

SysUtil_GetTargetId

name	**SysUtil_GetTargetId**			*type*	Function
return value	*type*	UINT			
		[0...FFFF]	*Target Identification*		
input value 1	*name*	**Dummy**			
	type	UINT	reserved.		
	value	[0]	*Set this always zero*		
description	Returns the Target Identification for CodeSys development environment.				

SysUtil_LedSet

name	**SysUtil_LedSet**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**LED_NR**			
	type	UINT	Selects the hardware LED by number		
	value	[Led_Nr]	*See hardware reference table for valid LED number*		
input value 2	*name*	**LIGHT**			
	type	SysSmPos_Axis	Defines the lighting state of selected LED		
	value	[-9…100]	*????*		
description	Sets the lighting state of an programmable LED: negative value: number of flashes positive values: duty cycle				

SysUtil_GetTaskInfo

name	**SysUtil_GetTaskInfo**			*type*	Function
return value	*type*	UINT	Depending on the InfoID (see below)		
	value	[0..X]	IF InfoID=0, function returns task state: 0: undefined 1: running 2: halted		
input value 1	*name*	**TaskID**			
	type	UINT	Id of the requested Task:		
	value	[0 … 8]	0: Task, running the PLC_PRG POU 1: Visu Task 2..8: Additional tasks		
input value 2	*name*	**InfoID**			
	type	UDINT	*Defines what kind of Information is returned*		
	value	[0…2]	0: function returns task status (see description above) 1: function returns period in ms 2: function returns execution time		
description	Returns task information or period/execution time				

SysUtil_GetSerNo

name	**SysUtil_GetSerNo**			*type*	Function
return value	*type*	UDINT			
		[0...FFFFFFFF]	*serial number of the system*		
input value 1	*name*	**Dummy**			
	type	UINT	reserved.		
	value	[0]	*Set this always zero*		
description	Returns the serial number of the PLC controller				

FBESysSerial.lib

Functions	***Description***
SysCom_Clear	
SysCom_Close	
SysCom_GetRxBufNum	
SysCom_GetStatus	
SysCom_Init	
SysCom_IsRxReady	
SysCom_Read	
SysCom_ReadBlock	Reads "Len" characters from the serial port to the buffer at Address until end of string or MaxLen is reached. Returns the number of characters
SysCom_ReadString	Reads characters from the serial port to the buffer StringData until end of string or MaxLen is reached
SysCom_ReOpen	
SysCom_Write	
SysCom_WriteBlock	
SysCom_WriteString	

Hardware Reference

hipecs PLC1000			
Available COM-Ports			
COM Nr.	1	2	3
COM Type	RS232	RS232	RS232 / RS422

Data types defined for serial library

```
TYPE
type_COM_BAUD : UDINT;
END_TYPE
```

```
TYPE
type_COM_DATABITS : INT;
END_TYPE
```

```
TYPE

 type_COM_PARITY : (

   COM_PARITY_EVEN:=69,
   COM_PARITY_ODD:=79,
   COM_PARITY_NONE:=78,
);
END_TYPE
```

```
TYPE

 type_COM_PORT:(
COM1:=1,
COM2:=2,
COM3:=3,
COM4:=4,
COM5:=5,
COM6:=6
);
END_TYPE
```

```
TYPE
type_COM_STOPBITS :  INT;
END_TYPE
```

SysCom_Clear

name	**SysCom_clear**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
description	Clears the receiver register and receiver Buffer				

SysCom_Close

name	**SysCom_close**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Function ended successfully.*		
		FALSE	*Function skipped.*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
description	Closes the corresponding COM interface. Receiver and transmitter will be disabled.				

SysCom_GetRxBufNum

name	**SysCom_GetRxBufNum**			*type*	Function
return value	*type*	UINT	Returns the result state.		
	value	[0..?]	*Number of received characters*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
description	Function returns the number of received characters				

SysCom_GetStatus

name	**SysCom_GetStatus**							*type*	Function
return value	*type*	BYTE	Returns the result state.						
	value	[0..FF]	*State of the Com Port*						
		Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
		TxOVL	RxOVL	-	-	-	TxRDY	TxEM	RxRDY
	RxRDY	Receiver Ready 1: The COM interface has received one or more characters. 0: There are no received characters stored to the receiver FIFO buffer							
	TxEM	Transmitter empty 1: There are no more characters in the transmission FIFO buffer. 0: There are characters in the transmitter FIFO							
	TxRDY	Transmitter Ready 1: The transmitter FIFO buffer is ready for storing additional characters. 0: The transmitter FIFO buffer is full. Do not start any further transmissions.							

	RxOVL	Receiver Overflow 1: There was an overflow of the receiver FIFO buffer. There are some lost characters. 0: No overflow occurred. The Overflow flag is reset after reading the status byte using function SYSCOM_GETSTATUS. So this overflow can only be read for one time.	
	TxOVL	Transmitter Overflow 1: There was an overflow of the transmitter FIFO buffer. There are some lost characters. 0: No overflow occurred. The Overflow flag is reset after reading the status byte using function SYSCOM_GETSTATUS. So this overflow can only be read for one time.	
input value 1	*name*	**ComPort**	
	type	type_COM_PORT	Selects the COM Port
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*
description	The function returns the status of a serial COM interface in a Byte.		

SysCom_Init

name	**SysCom_Init**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Com port initialized sucessfully*		
		FALSE	*Function skipped. An error occurred.*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the hardware related COM channel by name / number		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
input value 2	*name*	**Baud**			
	type	type_COM_Baud (UDINT)	*Selects the baudrate for the selected COM port*		
	value		*See data type reference for available baudrates*		
input value 3	*name*	**DataBits**			
	type	type_COM_DATABITS (INT)	Number of data bits for the selected COM port		
	value	[0.. X]			
input value 4	*name*	**Parity**			
	type	type_COM_PARITY	Selects the parity of the serial transmission		
	value		*See data type reference for valid parities*		
input value 5	*name*	**StopBits**			
	type	type_COM_STOPBITS (INT)	Selects the number of stopbits used for serial transmission		
	value	[0.. X]			
description	Initializes a COM interface and opens it for data transfer operations. If the user configures the serial channel within the CoDeSys system configuration dialog, there is no need to call the SYSCOM_INIT function. Function only needs to be called once at the beginning and not every PLC cycle.				

SysCom_ IsRxReady

name	**SysCom_ IsRxReady**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*minimum one character is in the receiver buffer*		
		FALSE	*receiver buffer empty*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
description	Check for characters in the receiver buffer. Returns TRUE if minimum one character is in the receiver buffer.				

SysCom_Read

name	**SysCom_Read**			*type*	Function
return value	*type*	Byte	Character read from buffer		
	value	[0..?]	*If return value is 0, then buffer is empty*		
input value 1	*name*	**ComPort**			
	type	type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
description	Reads one character from the receiver FIFO buffer. If there is no received character in the buffer, the function returns "0".				

SysCom_ReadBlock

name	**SysCom_ReadBlock**			*type*	Function
return value	*type*	UINT	The function returns the number of characters of the received string in bytes		
	value	[0..X]	*Number of characters received*		
input value 1	*name*	**ComPort**			
	type	Type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
input value 2	*name*	**Address**			
	type	UDINT	*Destination where the function has to copy the len characters to.*		
	value	[Adress]	*Address can be defined by the CoDeSys "ADR(variable name)" operation*		
input value 3	*name*	**len**			
	type	UINT	Maximum valid length of this string.		
	value	[Length of string]	*Length of string can be determined with the GetRxBufNum function*		
description	Reads Len characters from the serial port to the buffer at Address until end of string or Len is reached and returns the number of characters				

SysCom_ReadString

name	**SysCom_ReadString**			*type*	Function
return value	*type*	UINT	The function returns the number of characters of the received string in bytes		
	value	[0..X]	*Number of characters received*		
input value 1	*name*	**ComPort**			
	type	Type_COM_PORT	Selects the COM Port		
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*		
input value 2	*name*	**Stringdata**			
	type	STRING	*Destination where the function has to copy the string to.*		
	value	[String Data]	*Destination must be a variable of the string type*		
input value 3	*name*	**Maxlen**			
	type	UINT	Maximum valid length of this string.		
	value	[Maximum string length]			
description	Read a complete String from the receiver FIFO buffer. The string is either terminated with character ZERO or if the maximum string length is exceeded.				

SysCom_ ReOpen

name	**SysCom_ ReOpen**	*type*	Function
return value	*type*	BOOL	Returns the result state.
	value	TRUE	*COM Port reopened successfully*
		FALSE	*Function skipped. An error occurred.*
input value 1	*name*	**ComPort**	
	type	type_COM_PORT	Selects the COM Port
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*
description	Opens a COM interface again with last parameters. Receiver and transmitter register and buffer will cleared. The functions SYSCOM_CLOSE and SYSCOM_REOPEN may be used to block serial reception for some time.		

SysCom_Write

name	**SysCom_Write**	*type*	Function
return value	*type*	BOOL	Writes a single character to the transmitter FIFO buffer.
	value	TRUE	*Transmission of the data byte was successful*
		FALSE	*Transmission of the data byte failed*
input value 1	*name*	**ComPort**	
	type	Type_COM_PORT	Selects the COM Port
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*
input value 2	*name*	**Data**	
	type	BYTE	*Data Byte to transmit*
	value	[00..FF]	
description	Writes a single character to the transmitter FIFO buffer.		

SysCom_WriteString

name	**SysCom_WriteString**	*type*	Function
return value	*type*	UINT	The function returns the length of the transmitted string in bytes
	value	[0.. X]	*Number of character transmitted in bytes*
input value 1	*name*	**ComPort**	
	type	Type_COM_PORT	Selects the COM Port
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*
input value 2	*name*	**StringData**	
	type	String	*String data to transmit*
	value	[String data]	*Variable must be of the string type*
description	Writes a complete string to the transmitter FIFO buffer. The string must be terminated by a character ZERO		

SysCom_WriteBlock

name	**SysCom_WriteBlock**		*type*	Function
return value	*type*	UINT	The function returns the number of characters of the transmitted string	
	value	[0..X]	*Number of characters transmitted*	
input value 1	*name*	**ComPort**		
	type	Type_COM_PORT	Selects the COM Port	
	value	COM1, COM2, COM3	*See hardware reference table for valid COM Port*	
input value 2	*name*	**Address**		
	type	UDINT	*Destination from where the function has to copy the characters from*	
	value	[Adress]	*Address can be defined by the CoDeSys "ADR(variable name)" operation*	
input value 3	*name*	**len**		
	type	UINT	Maximum valid length of this string.	
	value	[Length of string]	*Length of string can be determined with the GetRxBufNum function*	
description	Writes Len characters from the variable to serial transmit buffer until end of string or Len is reached and returns the number of characters			

FBESysUSB.lib

Hardware Reference

hipecs PLC	PLC1010 / PLC1020	PLC1030
Available USB Ports	0	2
Hardware name / Connector name	USB-1	USB-2
Note: The hipecs has 2 available ports for each connector. USB-1 is reserved for CoDeSys programming interface and hyperterminal communication. USB-2 is for use with this library and supports 2 separated USB ports in device mode!		

SysUSB_Clear

name	**SysUSB_Clear**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Port successfully cleared.*	
		FALSE	*Function skipped. An error occurred.*	
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	*0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.*	
description	*Function clears receive and transmit buffer*			

SysUSB_Close

name	**SysUSB_Clear**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Port successfully closed.*	
		FALSE	*Function skipped. An error occurred.*	
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	*0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.*	
description	*Function closes a previously installed USB connection*			

SysUSB_IsConnected

name	**SysUSB_IsConnected**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	Port connected		
		FALSE	Port not connected		
input value 1	*name*	**Port**			
	type	UINT	Selects the USB channel by number		
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.		
description	Function checks if the USB port is connected to another system.				

SysUSB_IsRxReady

name	**SysUSB_IsRxReady**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	Data pending in receive buffer		
		FALSE	Receive buffer empty		
input value 1	*name*	**Port**			
	type	UINT	Selects the USB channel by number		
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.		
description	Function check receive buffer for data.				

SysUSB_Open

name	**SysUSB_Open**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	Port successfully opened.		
		FALSE	Error occurred. No port initialized.		
input value 1	*name*	**Port**			
	type	UINT	Selects the USB channel by number		
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.		
description	Opens a USB port				

SysUSB_RxBlock

name	**SysUSB_RxBlock**			*type*	Function Block
return value 1	*name*	**Confirm**			
	type	BOOL			
	value	FALSE			
		TRUE			
return value 2	*name*	**Busy**			
	type	BOOL	Checks if function is still busy.		
	value	FALSE	Function block available / not busy.		
		TRUE	Function block still busy.		
return value 3	*name*	**connected**			
	type	BOOL			
	value	FALSE			
		TRUE			
return value 4	*name*	**RxCount**			
	type	UDINT	Returns the number of characters read from the buffer		
	value	[0.. X]			
input value 1	*name*	**Port**			
	type	UINT	Selects the USB channel by number		
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.		
input value 2	*name*	**Enable**			
	type	BOOL			
	value	FALSE			
		TRUE			
input value 3	*name*	**pData**			
	type	POINTER TO BYTE	Address of the destination		
	value	[..]			
input value 4	*name*	**BlockSize**			
	type	UDINT	Number of character that are read from the buffer		
	value	[..]			
description	Reads "BlockSize" characters from the USB port to the destination at pData until end of string or Len is reached.				

SysUSB_RxByte

name	**SysUSB_RxByte**			*type*	Function
return value	*type*	BYTE	Returns the first byte of the receiver FIFO buffer		
	value	[..]	Port successfully opened.		
input value 1	*name*	**Port**			
	type	UINT	Selects the USB channel by number		
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.		
description	Reads one character from the receiver buffer				

SysUSB_RxString

name	**SysUSB_RxString**		*type*	Function Block
return value 1	*name*	**Confirm**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 2	*name*	**Busy**		
	type	BOOL	Checks if function is still busy.	
	value	FALSE	Function block available.	
		TRUE	Function block still busy.	
return value 3	*name*	**connected**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 4	*name*	**Strg**		
	type	STRING	Returns the string read from buffer	
	value	[..]		
return value 5	*name*	**RxCount**		
	type	UDINT	Returns the number of characters read from the buffer	
	value	[0.. X]		
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.	
input value 2	*name*	**Enable**		
	type	BOOL		
	value	FALSE		
		TRUE		
input value 4	*name*	**MaxSize**		
	type	UDINT	Maximum valid length of the string	
	value	[..]		
description	Reads a complete String from the receiver FIFO buffer. The string is either terminated with character ZERO or if the maximum string length is exceeded.			

SysUSB_TxBlock

name	**SysUSB_TxBlock**		*type*	Function Block
return value 1	*name*	**Confirm**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 2	*name*	**Busy**		
	type	BOOL	Checks if function is still busy.	
	value	FALSE	Function block available / not busy.	
		TRUE	Function block still busy.	
return value 3	*name*	**connected**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 4	*name*	**TxCount**		
	type	UDINT	Returns the number of characters written to the buffer	
	value	[0.. X]		
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.	
input value 2	*name*	**Enable**		
	type	BOOL		
	value	FALSE		
		TRUE		
input value 3	*name*	**pData**		
	type	POINTER TO BYTE	Address from where the function block fetches the data.	
	value	[..]	Check CoDeSys "ADR()" operator.	
input value 4	*name*	**BlockSize**		
	type	UDINT	Number of character that are written to the buffer.	
	value	[..]		
description	Writes "BlockSize" characters from a Address until end of string or Len is reached to the buffer of the USB port.			

SysUSB_TxByte

name	**SysUSB_TxByte**		*type*	Function
return value	*type*	BOOL		
	value	FALSE	A failure occurred. Not byte written.	
		TRUE	Data byte successfully written.	
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.	
input value 2	*name*	**Data**		
	type	BYTE	Byte which is written to the transmit buffer	
	value	[..]		
description	Write one character to the transmit buffer of the USB port			

SysUSB_TxString

name	**SysUSB_TxString**		*type*	Function Block
return value 1	*name*	**Confirm**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 2	*name*	**Busy**		
	type	BOOL	Checks if function is still busy.	
	value	FALSE	Function block available.	
		TRUE	Function block still busy.	
return value 3	*name*	**connected**		
	type	BOOL		
	value	FALSE		
		TRUE		
return value 4	*name*	**TxCount**		
	type	UINT	Returns the number of Bytes written to the buffer	
	value	[..]		
input value 1	*name*	**Port**		
	type	UINT	Selects the USB channel by number	
	value	[0..X]	0 is the lower virtual COM Port of a connected PC. 1 is the higher COM.	
input value 2	*name*	**Enable**		
	type	BOOL		
	value	FALSE		
		TRUE		
input value 3	*name*	**Strg**		
	type	STRING	String which is written to the transmit buffer	
	value	[..]		
description	Write a complete string to the transmit buffer. String must be terminated by a character ZERO.			

FBESysCAN.lib

Functions	***Description***
SysCAN_InitBasicCan	Initializes the Basic-CAN-Interface
SysCAN_IsRxMsg	Checks whether there are received CAN frames
SysCan_Receive	
SysCAN_RxMsg	Receives a CAN message
SysCan_Send	
SysCAN_TxMsg	Transmits a CAN message

Hardware Reference

hipecs PLC1010/1020		
	Available CAN Interfaces	
CAN Interface Nr.	0	1
CoDeSys Enumeration	CAN 0	CAN 1
CAN Mode	CANopen / Basic CAN	CANopen / Basic CAN

hipecs CORE10 Modules				
	Available CAN Interfaces			
CAN Interface Nr.	0	1	2	3
CoDeSys Enumeration	CAN 0	CAN 1	CAN 2	CAN 3
CAN Mode	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939

Data types defined for Basic CAN library

```
TYPE SYSCAN_CANMSG :
STRUCT
        Id      : UINT;
        Data    : ARRAY[0..7] OF BYTE;
        Len     : UINT;
END_STRUCT
END_TYPE
```

```
TYPE SYSCAN_CANNODE
CAN0:=0,
CAN1:= 1

);
END_TYPE
```

```
TYPE SYSCAN_DIR : (

CAN_RXD:=1,(*Indication for receive messages*)
CAN_TXD:=2 (*Indication for transmit messages*)
);
END_TYPE
```

SysCAN_InitBasicCan

name	**SysCAN_InitBasicCan**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*Init successful*	
		FALSE	*Function skipped. Init failed.*	

input value 1	*name*	**Node**	
	type	SYSCAN_CANNODE	CAN Interface Number
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*
description	Initializes the Basic-CAN-Interface. In order to use Basic-CAN add a CANopen master to the system configuration. Set baud rate within CANopen master.		

SysCAN_IsRxMsg

name	**SysCAN_IsRxMsg**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*One or more messages available*		
		FALSE	*No message available*		
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
description	Functions check if there is one or more messages in the receiver buffer.				

SysCAN_Receive

name	**SysCAN_Receive**			*type*	Function
return value	*type*	BOOL			
	value	TRUE			
		FALSE			
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
input value 2	*name*	**pMsg**			
	type	POINTER TO SYSCAN_MSG	*Pointer to the CAN message*		
	value		*check CoDeSys "ADR()" operator*		
description	Same functionality as SysCAN_RxMsg but realized as function and not as function block				

SysCAN_RxMsg

name	**SysCAN_RxMsg**			*type*	Function Block
return value 1	*name*	**Success**			
	type	BOOL	Returns the result state.		
	value	FALSE	*If message contend is invalid*		
		TRUE	*If message contend is valid*		
return value 2	*name*	**ID**			
	type	UINT	returns the CAN identifier of the received message		
	value	[0.. X]			
return value 3	*name*	**Data**			
	type	ARRAY [0..7] OF BYTE	*Data contend of the received message*		
	value	[Data bytes of the message]			
return value 4	*name*	**Len**			
	type	UINT	returns the number of data bytes that are valid		
	value	[0.. X]			
return value 5	*name*	**Flags**			
	type	UINT	Reserved for future use		
	value	[0..X]			
input value 1	*name*	**Enable**			
	type	BOOL	Must be True in order to enable this function block		
	value	FALSE	Function block won't be executed		
		TRUE	Function will be executed		

input value 2	*name*	**Node**	
	type	SYSCAN_CANNODE	CAN Interface Number
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*
description	Checks whether there is a received message available and returns the CAN message Note: Each message can only be read for one time from the receiver queue Reading of the message deletes the message from the FIFO automatically ! Note: This function only works with 11 Bit identifiers		

SysCAN_Send

name	**SysCAN_Send**			*type*	Function
return value	*type*	BOOL			
	value	TRUE			
		FALSE			
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
input value 2	*name*	**pMsg**			
	type	POINTER TO SYSCAN_MSG	*Pointer to the CAN message*		
	value		*check CoDeSys "ADR()" operator*		
description	Same functionality as SysCAN_RxMsg but realized as function and not as function block				

SysCAN_TxMsg

name	**SysCAN_TxMsg**			*type*	Function Block
return value 1	*name*	**Success**			
	type	BOOL	Returns the result state.		
	value	FALSE	If message contend is invalid		
		TRUE	If message contend is valid		
input value 1	*name*	**Enable**			
	type	BOOL	Must be True in order to enable this function block		
	value	FALSE	Function block won't be executed		
		TRUE	Function will be executed		
input value 2	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	See hardware reference table for valid CAN interface		
input value 3	*name*	**ID**			
	type	UINT	CAN identifier for the message		
	value	[0.. X]			
input value 4	*name*	Data			
	type	ARRAY [0..7] OF BYTE	Data contend of the message		
	value	[Data bytes of the message]			
input value 5	*name*	**Len**			
	type	UINT	defines the number of data bytes that are valid		
	value	[0.. X]			
input value 6	*name*	**Flags**			
	type	UINT	Reserved for future use		
	value	[0..X]			
description	Writes a CAN message to the transmit buffer. Note: This function only works with 11 Bit identifiers				

SysCAN_CanNodeGetStatus

name	**SysCAN_CanNodeGetStatus**		*type*	Function
return value	*type*	UINT		
	value	[0...FFFF]	*Bit 0 :1 if node is running* *Bit6 : 1 if node is in error warning state* *Bit7 : 1 if the node is in bus off error state*	
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
description	Returns the state of the CAN hardware module			

SysCAN_CanNodeRecover

name	**SysCAN_Recover**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	*recovering procedure startet*	
		FALSE	*starting of recovering procedure not successful*	
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
description	Function tries to recover a SYSCAN_CANNODE from bus off error			

SysCAN_CanNodeRecover

name	**SysCAN_Recover**		*type*	Function
return value	*type*	BOOL		
	value	TRUE	*recovering procedure startet*	
		FALSE	*starting of recovering procedure not successful*	
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
input value 2	*name*	***NodeId***		
	type	UINT	*New node ID for CANopen interface*	
	value	0...127	*0 : Node-ID will not be changed*	
input value 3	*name*	***CANBaud***		
	type	UDINT	*New baud rate for CAN interface*	
	value	[0...1.000.000]	*0 : Baud rate will not be changed*	
description	This function reconfigs the CAN bus interface. The bus must have been initialized using the system configuration dialog. If you do not want to start the bus before calling reconfig from PLC application use CANopen node ID = 127. It is recommended to use this function only in the main task calling PLC_PRG			

SysCAN29Bit_IsRxMsg

name	**SysCAN29Bit_IsRxMsg**		*type*	Function
return value	*type*	BOOL	Returns the result state.	
	value	TRUE	*One or more messages available*	
		FALSE	*No message available*	
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
description	Function checks if there is one or more messages with 29 Bit Identifier in the receiver buffer.			

SysCAN29Bit_Send

name	**SysCAN29Bit_Send**		*type*	Function
return value	*type*	BOOL		
	value	TRUE		
		FALSE		
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
input value 2	*name*	pMsg		
	type	POINTER TO SYSCAN29BIT_CANMSG	*Pointer to the* 29 bit identifier *CAN message*	
	value		*check CoDeSys "ADR()" operator*	
description	Same functionality as SysCAN_RxMsg but for 29 bit identifiers			

SysCAN29Bit_Receive

name	**SysCAN29Bit_Receive**		*type*	Function
return value	*type*	BOOL		
	value	TRUE		
		FALSE		
input value 1	*name*	**Node**		
	type	SYSCAN_CANNODE	CAN Interface Number	
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*	
input value 2	*name*	**pMsg**		
	type	POINTER TO SYSCAN29BIT_CANMSG	*Pointer to the* 29 bit identifier *CAN message*	
	value		*check CoDeSys "ADR()" operator*	
description	Same functionality as SysCAN_RxMsg but for 29 bit identifiers			

SYSCiA405.lib

The run time system includes a very powerful CANopen master. The CANopen interface is based on a two level software structure. Adding a CANopen master to the PLC configuration activates the lower level according to DS302 level. This layer handles the complete network boot up and PDO transfer automatically.
The second level according to DS405 establishes an interface for the IEC61131 application to the CANopen layer. This layer is implemented in the " FBE_CIA405.lib" library file.

Configuration:
The configuration of this master is done with the CoDeSys PLC configuration dialog. The master is enabled if there is a CANopen master added to the system configuration. The functionality and the maximum number of slaves depends on the target system.

Note: The hipecs supports two separated CAN busses, so the "CanNode" parameter is necessary to choose the right CAN interface. If only one CAN bus is used, it is possible to use the SysCia405_StdBus0.lib library, which has no "CanNode" parameter and always works with the CAN interface 0. This is the IEC conform library, since the IEC only defines a device with a single CAN interface.

Hardware Reference

hipecs PLC1010/1020		
	Available CAN Interfaces	
CAN Interface Nr.	0	1
CoDeSys Enumeration	CAN 0	CAN 1
CAN Mode	CANopen / Basic CAN	CANopen / Basic CAN

hipecs CORE10 Modules				
		Available CAN Interfaces		
CAN Interface Nr.	0	1	2	3
CoDeSys Enumeration	CAN 0	CAN 1	CAN 2	CAN 3
CAN Mode	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939

Data types defined for CANopen CiA DS405 library

```
TYPE
        SysCiA405_CANopen_Kernel_Error : WORD;
END_TYPE
```

```
TYPE
        SysCiA405_Device: USINT (0..127);
END_TYPE
```

```
TYPE SysCiA405_EMCY_Error :
STRUCT
        EMY_ERROR_CODE          : WORD;
        ERROR_REGISTER          : BYTE;
        ERROR_FIELD             : ARRAY[1..5] OF BYTE;
END_STRUCT
END_TYPE
```

```
TYPE
        SysCiA405_SDO_Error : UDINT;
END_TYPE
```

```
TYPE SysCiA405_State : (
        INIT:= 0,
        RESET_COMM:= 7,
        RESET_APP:= 6,
        PRE_OPERATIONAL:= 127,
        STOPPED:= 4,
        OPERATIONAL:= 5,
        UNKNOWN:= 8,
        NOT_AVAIL:= 1
);
END_TYPE
```

```
TYPE SysCiA405_Transition_State : (
        START_REMOTE_NODE:= 16#01,
        STOP_REMOTE_NODE:= 16#02,
        ENTER_PRE_OPERATIONAL:= 16#80,
        RESET_NODE:= 16#81,
        RESET_COMMUNICATION:= 16#82
);
END_TYPE
```

SysCiA405_Get_Kernel_State

name	**SysCiA405_Get_Kernel_State**		*type*	Function Block
return value 1	*name*	**CONFIRM**		
	type	BOOL	Returns the result state.	
	value	FALSE	An error occured. Return values are invalid!	
		TRUE	Function successfully completed. Return values are valid.	
return value 2	*name*	**STATE**		
	type	SysCiA405_CANopen_kernel_Error	Returns the state of the CANopen kernel software	
	value	[..]	Check data type reference for available values.	
input value 1	*name*	**CanNode**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	*name*	**Enable**		
	type	BOOL	Must be True in order to enable this function block	
	value	FALSE	Function block won't be executed	
		TRUE	Function will be executed	
description	Reads the error state of the CANopen kernel driver. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.			

SysCiA405_Get_Local_Node_Id

name	**SysCiA405_Get_Local_Node_Id**		*type*	Function Block
return value 1	*name*	**CONFIRM**		
	type	BOOL	Returns the result state.	
	value	FALSE	An error occured. Return values are invalid!	
		TRUE	Function successfully completed. Return values are valid.	
return value 2	*name*	**Device**		
	type	SysCiA405_Device (USINT)	Returns the node ID of the device	
	value	[0..127]		
input value 1	*name*	**CanNode**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	*name*	**Enable**		
	type	BOOL	Must be True in order to enable this function block	
	value	FALSE	Function block won't be executed	
		TRUE	Function will be executed	
description	Function returns the Node ID of the device.			

SysCiA405_Get_State

name	**SysCiA405_Get_State**		*type*	Function Block
return value 1	*name*	**CONFIRM**		
	type	BOOL	Returns the result state.	
	value	FALSE	An error occured. Return values are invalid!	
		TRUE	Function successfully completed. Return values are valid.	
return value 2	*name*	**State**		
	type	SysCiA405_State	Returns NMT state of the selected device.	
	value	[..]	Check data type reference for available values.	
input value 1	*name*	**CanNode**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	*name*	**Device**		
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.	
	value	[0..127]		
input value 3	*name*	**Enable**		
	type	BOOL	Must be True in order to enable this function block	
	value	FALSE	Function block won't be executed	
		TRUE	Function will be executed	
description	Reads the NMT state of a selected CANopen node. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.			

SysCiA405_Is_Any_EMCY

name	**SysCiA405_Is_Any_EMCY**		*type*	Function
return value	*type*	BOOL	The function checks if there are any emergency messages stored in the emergency FIFO memory	
	value	TRUE	One or more emergency messages pending	
		FALSE	No emergency	
input value 1	*name*	**Node**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	*name*	**Enable**		
	type	BOOL	Must be True in order to enable this function	
	value	FALSE	Function won't be called	
		TRUE	Function will be called	
description	Checks whether there are any emergencies stored in the emergency FIFO memory.			

SysCiA405_NMT

name	**SysCiA405_NMT**		type	Function Block
return value 1	name	**CONFIRM**		
	type	BOOL	Returns the result state.	
	value	FALSE	An error occured. Return values are invalid!	
		TRUE	Function successfully completed. Return values are valid.	
return value 2	name	**ERROR**		
	type	SysCiA405_CANopen_Kernel _Error (WORD)	Returns the CANopen Error Code if NMT transmission failed.	
	value	[..]	Check data type reference for available values.	
input value 1	name	**CanNode**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	name	**Device**		
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.	
	value	[0..127]		
input value 3	name	**State**		
	type	SysCiA405_Transition_State	Defines the new state for selected Node	
	value	[..]	Check data type reference for available values.	
input value 4	name	**Enable**		
	type	BOOL	Must be True in order to enable this function	
	value	FALSE	Function won't be called	
		TRUE	Function will be called	
description	Sends a NMT command to the CANopen network. This command may be used, if slaves transition states must be changed while the network is still running.			

SysCiA405_Recv_EMCY

name	**SysCiA405_Recv_EMCY**		type	Function Block
return value 1	name	**CONFIRM**		
	type	BOOL	Returns the result state.	
	value	FALSE	An error occured. Return values are invalid!	
		TRUE	Function successfully completed. Return values are valid.	
return value 2	name	**Device**		
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.	
	value	[0..127]		
return value 3	name	**Error**		
	type	SysCiA405_CANopen_Kernel _Error (WORD)	Returns the state of the CANopen kernel software	
	value	[..]	Check data type reference for available values.	
return value 4	name	**EMY_ERROR**		
	type	SysCiA405_EMCY_Error	Emergency message of the CANopen node	
	value	[..]	Check data type reference for available values.	
input value 1	name	**CanNode**		
	type	UINT	CAN Interface Number	
	value	[0, 1]	See hardware reference table for valid CAN interface	
input value 2	name	**Enable**		
	type	BOOL	Must be True in order to enable this function	
	value	FALSE	Function won't be called	
		TRUE	Function will be called	
description	Reads the oldest emergency message stored in the emergency FIFO memory. Reading of an emergency will also delete this message from the FIFO memory, so each message can only be read once. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters. The Function may either return an emergency sent by a slave (external emergency) or an emergency created from the CANopen Master itself (internal emergency).			

External Emergency:

Sent by an external CANopen slave. In this case the Variable EMY_ERROR exactly represents the values transmitted by the connected Slave.

Name	Data-Type	Description
DEVICE	SysCiA405_Device (USINT)	Node ID of the CANopen node that produced this emergencies
ERROR	SysCiA405_CANopen_Kernel_Error (WORD)	State of the CANopen kernel software.
EMY_ERROR.	SysCiA405_EMCY_Error	Emergency message of the CANopen node.
Emy_Error_Code		Error Code sent from the slave within the emergency message
Error_Register		Error Register sent from the slave within the emergency message
Error_Field[1]..[5]		Error Field sent from the slave within the emergency message

Internal Emergency:

Created from the CANopen master. In this case the Variable EMY_ERROR shows the slave number that caused the Emergency of the master.

Name	Data-Type	Description
DEVICE	CIA405_DEVICE	0: Always zero to indicate, that the emergency was created from the master firmware and not transmitted over the CAN network
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
EMY_ERROR.	CIA405_EMY _ERROR	Emergency message of the CANopen master.
Emy_Error_Code		Error Code created from the master firmware
Error_Register		1: Error is set 0: Information for no Error (or automatically fixed error)
Error_Field[1]		Slave-ID that caused the emergency For example if the node guarding of a connected slave fails, the node ID of this slave is reported in Error_Field[1].
Error_Field[2]..[5]		0x00000000 The CANopen master was not able to check the exact reason for the emergency. For example bus errors or distortions may cause such entries in the emergency FIFO. 0x00000001 There was a guarding error detected at this slave (node guarding or heartbeat will not be distinguished) 0x00000002 The slave answered a SDO transfer with an Abort SDO message other codes reserved for future use

SysCiA405_Recv_EMCY_Dev

name	**SysCiA405_Recv_EMCY_Dev**	*type*	Function Block
return value 1	*name*	**CONFIRM**	
	type	BOOL	Returns the result state.
	value	FALSE	An error occured. Return values are invalid!
		TRUE	Function successfully completed. Return values are valid.
return value 2	*name*	**Error**	
	type	SysCiA405_CANopen_Kernel _Error (WORD)	Returns the state of the CANopen kernel software
	value	[..]	Check data type reference for available values.
return value 3	*name*	**EMY_ERROR**	
	type	SysCiA405_EMCY_Error	Emergency message of the CANopen node
	value	[..]	Check data type reference for available values.
input value 1	*name*	**CanNode**	
	type	UINT	CAN Interface Number
	value	[0, 1]	See hardware reference table for valid CAN interface
return value 2	*name*	**Device**	
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.
	value	[0..127]	
input value 3	*name*	**Enable**	
	type	BOOL	Must be True in order to enable this function
	value	FALSE	Function won't be called
		TRUE	Function will be called
description			

SysCiA405_SDO_READ4

name	**SysCiA405_SDO_READ4**	*type*	Function Block
return value 1	*name*	**CONFIRM**	
	type	BOOL	Returns the result state.
	value	FALSE	An error occured. Return values are invalid!
		TRUE	Function successfully completed. Return values are valid. Check ERRORINFO to verify SDO was read correct.
return value 2	*name*	**Error**	
	type	SysCiA405_CANopen_Kernel _Error (WORD)	Returns the state of the CANopen kernel software.
	value	[..]	Check data type reference for available values.
return value 3	*name*	**ERRORINFO**	
	type	SysCiA405_SDO_Error (UDINT)	Abort Code in case of the SDO transfer fails. Zero if the SDO transfer was successfully completed and the data is valid
	value	[..]	Check data type reference for available values.
return value 4	*name*	**DATA**	
	type	ARRAY [1..4] OF BYTE	Data field representing the result of the SDO transmission.
	value	[4 data bytes]	The data must be converted to the requested data type by calling the corresponding typecast function.
return value 5	*name*	**DATALENGTH**	
	type	USDINT	Length of the valid data field.
	value	[0..4]	
input value 1	*name*	**CanNode**	
	type	UINT	CAN Interface Number
	value	[0, 1]	See hardware reference table for valid CAN interface
input value 2	*name*	**Device**	
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.
	value	[0..127]	

input value 3	*name*	**INDEX**	
	type	WORD	Index of the object to be read
	value	[0..X]	
input value 4	*name*	**SUBINDEX**	
	type	BYTE	Subindex of the object to be read
	value	[0..X]	
input value 5	*name*	**Enable**	
	type	BOOL	Must be True in order to enable this function
	value	FALSE	Function won't be called
		TRUE	Function will be called
description	Read data from a slave node using SDO transfer. Maximum size of data is 4 bytes. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.		

SysCiA405_SDO_WRITE4

name	**SysCiA405_SDO_WRITE4**			*type*	Function Block
return value 1	*name*	**CONFIRM**			
	type	BOOL	Returns the result state.		
	value	FALSE	An error occured. Return values are invalid!		
		TRUE	Function successfully completed. Return values are valid. Check ERRORINFO to verify SDO was read correct.		
return value 2	*name*	**Error**			
	type	SysCiA405_CANopen_Kernel_Error (WORD)	Returns the state of the CANopen kernel software.		
	value	[..]	Check data type reference for available values.		
return value 3	*name*	**ERRORINFO**			
	type	SysCiA405_SDO_Error (UDINT)	Abort Code in case of the SDO transfer fails. Zero if the SDO transfer was successfully completed and the data is valid		
	value	[..]	Check data type reference for available values.		
input value 1	*name*	**CanNode**			
	type	UINT	CAN Interface Number		
	value	[0, 1]	See hardware reference table for valid CAN interface		
input value 2	*name*	**Device**			
	type	SysCiA405_Device (USINT)	Number of the CAN Node. Node ID.		
	value	[0..127]			
input value 3	*name*	**INDEX**			
	type	WORD	Index of the object to be written		
	value	[0..X]			
input value 4	*name*	**SUBINDEX**			
	type	BYTE	Subindex of the object to be written		
	value	[0..X]			
input value 5	*name*	**DATA**			
	type	ARRAY [1..4] OF BYTE	Data field representing the result of the SDO transmission.		
	value	[4 data bytes]	The data must be converted to the BYTE data type by calling the corresponding typecast function.		
input value 6	*name*	**DATALENGTH**			
	type	USDINT	Length of the valid data field.		
	value	[0..4]			
input value 7	*name*	**Enable**			
	type	BOOL	Must be True in order to enable this function		
	value	FALSE	Function won't be called		
		TRUE	Function will be called		
description	Write data to a slave node using SDO transfer. Maximum size of data is 4 bytes. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.				

FBESysTask.lib

The Library FBESysTask.lib is a Library extension for the CoDeSys PLC runtime system and returns the processing / interval time of the last cycle for the chosen task.

SysTask_GetTimeCycle

name	**SysTask_GetTimeCycle**			*type*	Function
return value	*type*	UINT	Returns the interval time of the last cycle for this task in milli seconds		
	value	[0..X]			
input value 1	*name*	**TaskId**			
	type	UINT	0: PLC_PRG Task; 1: Visu_Task;		
	value	[0..X]			
description	Returns the interval time of the last cycle for this task in milli seconds				

SysTask_GetProcessTime

name	**SysTask_GetProcessTime**			*type*	Function
return value	*type*	UINT	Returns the processing time of the last cycle for this task in milli seconds		
	value	[0..X]			
input value 1	*name*	**TaskId**			
	type	UINT	0: PLC_PRG Task; 1: Visu_Task;		
	value	[0..X]			
description	Returns the processing time of the last cycle for this task in milli seconds				

FBESysNet.lib

Hardware Reference

hipecs PLC			
hipecs hardware version	PLC 1010	PLC1020	PLC1030
Available Ethernet Interfaces	none	1	1

Data types defined for FBESysNet.lib library

```
TYPE  tSYSNET_EMAIL :
STRUCT
            Confirm   : BOOL;                    (*Set TRUE if transmission completed                    *)
            Success  : BOOL;                     (*True if email was sent, FALSE if transmission failed  *)
            ErrCode  : UINT;                     (*Errorcode for Transmission                            *)
            MailData : ARRAY[0..16#1000] OF UINT; (* Data Reservation for library use, do not modify
                                                    Maximum mail size is 8 kByte (Text apporx 7 kByte) *)
END_STRUCT
END_TYPE
```

```
TYPE  tSYSNET_HttpScriptCtrl :
STRUCT
            SourceLine  : STRING(255);           (* String that is read from the CGI file and must be processed      *)
            DestinLine  : STRING(255);           (* Output String of the CGI-Processor                               *)
            MaxLen      : UINT;                  (* Maximum Len of DestinLine                                        *)
            pCgiData    : POINTER TO UDINT;      (* This UDINT variable is for CGI processors internal use           *)
END_STRUCT
END_TYPE
```

```
TYPE tSYSNET_NetConfig :
STRUCT
            HostName : STRING(15);               (* Name of the network host we want to setup                        *)
            MAC       : ARRAY[1..6] OF USINT;    (* MAC address for network Interface                                *)
            IP        : ARRAY[1..4] OF USINT;    (* IP address for network host                                      *)
            MASK      : ARRAY[1..4] OF USINT;    (* Sub net mask for local network area                              *)
            GW        : ARRAY[1..4] OF USINT;    (* IP address of standard gateway for internet DNS access etc       *)
            PDNS      : ARRAY[1..4] OF USINT;    (* IP address of primary DNS server                                 *)
            SDNS      : ARRAY[1..4] OF USINT;    (* IP address of secondary DNS server                               *)
            UseDHCP  : BOOL;                     (* if TRUE the network driver uses DHCP service to get an IP address *)
END_STRUCT
END_TYPE
```

SysNet_EmailCreate

name	**SysNet_EmailCreate**			*type*	Function
return value 1	*type*	BOOL	Initializes the mail data for the new mail.		
	value	FALSE	Function skipped. An error occurred.		
		TRUE	Mail data successfully initialized		
input value 1	*name*	**eMail**			
	type	tSYSNET_EMAIL			
	value	[..]	Check data type reference for available values.		
input value 2	*name*	**eMailAddress**			
	type	STRING (64)	eMail-Address of recipient		
	value	[..]			
input value 3	*name*	**Subject**			
	type	STRING (64)	Subject Text for this eMail		
	value	[..]			
input value 4	*name*	**Header**			
	type	STRING (255)	Header text placed in front of mail text		
	value	[..]			
input value 5	*name*	**Footer**			
	type	STRING (255)	Footer text placed behind the mail text		
	value	[..]			
description	Initializes the mail data for the new mail. The mail text is cleared and must be written using function SysNet_EmailWrite				

SysNet_EmailSend

name	**SysNet_EmailSend**			*type*	Function
return value 1	*type*	BOOL	Forwards the email to the mail send service of operating system.		
	value	FALSE	Function skipped. An error occurred.		
		TRUE	Mail data successfully initialized		
input value 1	*name*	**Mail**			
	type	tSYSNET_EMAIL	Name of the Email, that must be sent		
	value	[..]	Check data type reference for available values.		
description	Forwards the email to the mail send service of operating system. The function return immediately. After transmission of the mail, the Confirm-Flag of the email is set TRUE , if transmission was successful, the Success-Flag of the mail is set TRUE				

SysNet_EmailSetFooter

name	**SysNet_EmailSetFooter**			*type*	Function
return value 1	*type*	BOOL	Changes the text footer for an existing mail		
	value	FALSE	Function skipped. An error occurred.		
		TRUE	Footer changed successfully		
input value 1	*name*	**eMail**			
	type	tSYSNET_EMAIL	Name of the Email, that must be changed		
	value	[..]	Check data type reference for available values.		
input value 2	*name*	**Footer**			
	type	STRING (255)	Text of the new footer		
	value	[..]			
description	Changes the text footer for an existing mail				

SysNet_EmailSetHeader

name	**SysNet_EmailSetHeader**		*type*	Function
return value 1	*type*	BOOL	Changes the text header for an existing mail	
	value	FALSE	Function skipped. An error occurred.	
		TRUE	Header changed successfully	
input value 1	*name*	**eMail**		
	type	tSYSNET_EMAIL	Name of the Email, that must be changed	
	value	[..]	Check data type reference for available values.	
input value 2	*name*	**Header**		
	type	STRING (255)	Text of the new header	
	value	[..]		
description	Changes the text header for an existing mail			

SysNet_EmailSetRecipent

name	**SysNet_EmailSetRecipent**		*type*	Function
return value 1	*type*	BOOL	Changes the recipient for an existing mail	
	value	FALSE	Function skipped. An error occurred.	
		TRUE	recipent changed successfully	
input value 1	*name*	**eMail**		
	type	tSYSNET_EMAIL	Name of the Email, that must be changed	
	value	[..]	Check data type reference for available values.	
input value 2	*name*	**eMailAddress**		
	type	STRING (64)	email address of new recipent	
	value	[..]		
description	Changes the recipient for an existing mail			

SysNet_EmailSetSubject

name	**SysNet_EmailSetSubject**		*type*	Function
return value 1	*type*	BOOL	Changes the subject for an existing mail	
	value	FALSE	Function skipped. An error occurred.	
		TRUE	subject changed successfully	
input value 1	*name*	**eMail**		
	type	tSYSNET_EMAIL	Name of the Email, that must be changed	
	value	[..]	Check data type reference for available values.	
input value 2	*name*	**Subject**		
	type	STRING (64)	text of new subject	
	value	[..]		
description	Changes the subject text for an existing mail			

SysNet_EmailTextClear

name	**SysNet_EmailTextClear**		*type*	Function
return value 1	*type*	BOOL	Clears the complete mail text of an existing email	
	value	FALSE	Function skipped. An error occurred.	
		TRUE	text changed successfully	
input value 1	*name*	**eMail**		
	type	tSYSNET_EMAIL	Name of the Email, that must be changed	
	value	[..]	Check data type reference for available values.	
description	Clears the complete mail text of an existing eMail, and may be used to create new text for a mail without creating the mail. All other settings for the eMail (recipient, subject, header and footer) will be unchanged			

SysNet_EmailWrite

name	**SysNet_EmailWrite**		*type*	Function
return value 1	*type*	BOOL	Adds text to the actual eMail text	
	value	FALSE	Function skipped. An error occurred	
		TRUE	Text added successfully	
input value 1	*name*	**eMail**		
	type	tSYSNET_EMAIL	Name of the Email, that must be changed	
	value	[..]	Check data type reference for available values	
input value 2	*name*	**MailText**		
	type	STRING (255)	Add this text to the already existing mail text	
	value	[..]		
input value 3	*name*	**CrLf**		
	type	BOOL	Inserts Carriage Return and Line Feed	
	value	FALSE	Nothing is added	
		TRUE	<Carriage Return> <Line Feed> is placed at the end of MailText	
description	Adds the text to the actual eMail text. This function is used to fill the complete eMail text. If the function has added the complete text to the eMail, TRUE is returned.			

SysNet_HttpSetDir

name	**SysNet_HttpSetDir**		*type*	Function
return value 1	*type*	BOOL	Sets the active directory for the embedded web server	
	value	FALSE	Function skipped. An error occurred.	
		TRUE	directory changed successfully	
input value 1	*name*	**WebDirPath**		
	type	String	Source path name for the web server	
	value	[..]		
description	This function sets the active directory for the embedded web server.			

SysNet_ HttpRegCgiFunction

name	**SysNet_ HttpRegCgiFunction**		*type*	Function
return value 1	*type*	BOOL		
	value	FALSE	Function skipped. An error occurred.	
		TRUE	Function registered successfully.	
input value 1	*name*	**CgiFunction**		
	type	UINT	Index of the CGI function	
	value	[1.. 13]	Check table below for available values	
input value 2	*name*	**FunctionPouUndex**		
	type	UINT	index of CGI Scrip Controller Function, using CoDeSys INDEXOF() operator	
	value	[..]		
description	This function registers the functions for XML script processing in order to support dynamic web sites			

The following in indexes are used for registering CGI functions.

CGI Function	*Index*	*Name of Function*
	1	CGI Processor
	2	CgiGetSTRING
	3	CgiGetBOOL
	4	CgiGetUDINT
	5	CgiGetDINT
	10	CgiSetSTRING
	11	CgiSetBOOL
	12	CgiSetUDINT
	13	CgiSetDINT

Check FBE tutorial 'How To Use WebVisu' in the download area for further instructions.

FBESYSMemory.lib

SysMem_EepromRd

name	**SysMem_EepromRd**		*type*	Function
return value 1	*type*	BOOL	Reads data from the EEPROM	
	value	FALSE	Function skipped. An error occurred	
		TRUE	Data read successfully	
input value 1	*name*	**EEPROMAddress**		
	type	UINT	Address within EEPROM	
	value	[..]		
input value 2	*name*	**MemoryAddress**		
	type	UDINT	Destination Address for data read from EEPROM	
	value	[..]	Check CoDeSys ADR() operator	
input value 3	*name*	**ByteCount**		
	type	UINT	Number of bytes to read from EEPROM	
	value	[..]		
description	The function reads data from the EEPROM to the destination address.			

SysMem_EepromWr

name	**SysMem_EepromWr**		*type*	Function
return value 1	*type*	BOOL	Writes data to the EEPROM	
	value	FALSE	Function skipped. An error occurred	
		TRUE	Data written successfully	
input value 1	*name*	**EEPROMAddress**		
	type	UINT	Address within EEPROM	
	value	[..]		
input value 2	*name*	**MemoryAddress**		
	type	UDINT	Address of first data to write to EEPROM	
	value	[..]	Check CoDeSys ADR() operator	
input value 3	*name*	**ByteCount**		
	type	UINT	Number of bytes to write to EEPROM	
	value	[..]		
description	The function writes data to the EEPROM address from the source address.			

SysMem_NVBlockEnable

name	**SysMem_NVBlockEnable**		*type*	Function
return value 1	*type*	BOOL		
	value	FALSE	Function skipped. An error occurred	
		TRUE	NV Block successfully enabled	
input value 1	*name*	**Enable**		
	type	BOOL	Enables the NON VOLATILE (NV) memory block in "Merker/Memory" address area.	
	value	TRUE		
		FALSE		
description	Enables the NON-VOLATILE (NV) memory block in "Merker/Memory" address area. If enabled, the upper 1 kByte of "Merker/Memory" area is used as non-volatile memory that is never reinitialized. Also loading a new project does not reinitialize the NV memory. Valid address range : %MB15360 ... %MB16375 Please note: CODESYS does not check for data overlap. Take care to declare 16 or 32 bit data to even start address.			

SysMem_NVBlockDisable

name	**SysMem_NVBlockDisable**		*type*	Function
return value 1	*type*	BOOL		
	value	FALSE	Function skipped. An error occurred	
		TRUE	NV Block successfully disabled	
input value 1	*name*	**Enable**		
	type	BOOL	Enables function to disable NV memory	
	value	TRUE		
		FALSE		
description	Disables the NON-VOLATILE (NV) memory block in "Merker" address memory area.			

SysMem_SaveRetainCmd

name	**SysMem_SaveRetainCmd**		*type*	Function
return value 1	*type*	BOOL		
	value	FALSE	Function skipped. An error occurred	
		TRUE	Function successfully executed	
input value 1	*name*	**Size**		
	type	INT	Enables function to disable NV memory	
	value	[-1 ... 32767]	Size of data that must be saved manually. Use -1 to save complete retain data segment	
description	Forces the system to save the retain data segment to non-volatile memory. Function is only needed on Core modules, if target hardware does not support power fail interrupt. Please note: function can take several 100 ms			

FBESysTime.lib

SysTime_GetTime

name	**SysTime_GetTime**		*type*	Function
return value 1	*type*	TOD	read time from RTC	
	value	[tod#hh:mm:ss]		
input value 1	*name*	**Mode**		
	type	UINT	Reserved for future use: Set this parameter to 0	
	value	0		
description	Function reads the time from RTC			

SysTime_GetDate

name	**SysTime_GetDate**		*type*	Function
return value 1	*type*	DATE	read date from RTC	
	value	[d#yyyy-mm-dd]		
input value 1	*name*	**Mode**		
	type	UINT	Reserved for future use: Set this parameter to 0	
	value	0		
description	Function reads the date from RTC.			

SysTime_GetDateandTime

name	**SysTime_GetDateandTime**		*type*	Function
return value 1	*type*	DATE_AND_TIME	read time and date from RTC	
	value	[dt#yyyy-mm-dd-hh:mm:ss]		
input value 1	*name*	**Mode**		
	type	UINT	Reserved for future use: Set this parameter to 0	
	value	0		
description	Function reads time and date from RTC			

SysTime_SetDate

name	**SysTime_SetDate**		*type*	Function
return value 1	*type*	BOOL	Set new date in RTC	
	value	FALSE	An error occurred. Function skipped	
		TRUE	date successfully set	
input value 1	*name*	**NewDate**		
	type	DATE	New date for RTC	
	value	[d#yyyy-mm-dd]		
description	This function write a new date to the RTC			

SysTime_SetDateandTime

name	**SysTime_Get Date**			*type*	Function
return value 1	*type*	BOOL	Set new dat and new time in RTC		
	value	FALSE	An error occurred. Function skipped		
		TRUE	date and time successfully set		
input value 1	*name*	**NewDateAndTime**			
	type	DATE_AND_TIME	New date and time for RTC		
	value	[dt#yyyy-mm-dd-hh:mm:ss]			
description	This function writes a new date and a new time to the RTC				

SysTime_SetTIME

name	**SysTime_Get Date**			*type*	Function
return value 1	*type*	BOOL	Set new time in RTC		
	value	FALSE	An error occurred. Function skipped		
		TRUE	time successfully set		
input value 1	*name*	**NewTime**			
	type	TOD	New time for RTC		
	value	[tod#hh:mm:ss]			
description	This function writes a new time to the RTC				

FBESysIo.lib

The Library FBESysIo.lib is a Library extension for the CoDeSys PLC runtime system and supporting several utilities for IEC61131 applications running on systems from frenzel + berg elektronic. It is an internal library; all functions are included in the runtime system. With these functions a manipulation of the IO system is possible during the PLC loop.

SysIo_RdAllDigitalIn

name	**SysIo_RdAllDigitalIn**		*type*	Function
return value 1	*type*	BOOL	Read all digital input data	
	value	FALSE	An error occurred. Function skipped	
		TRUE	All digital inputs read successfully	
input value 1	*name*	**Mode**		
	type	UINT	Reserved for future use, set to 0	
	value	[0]		
description	This functions reads all digital input data from hardware to PLC data. This function can be used to start an additional hardware input update within the PLC task or from interrupt level			

SysIo_WrAllDigitalOut

name	**SysIo_WrAllDigitalOut**		*type*	Function
return value 1	*type*	BOOL	Write all digital output data	
	value	FALSE	An error occurred. Function skipped	
		TRUE	All digital outputs written successfully	
input value 1	*name*	**Mode**		
	type	UINT	Reserved for future use, set to 0	
	value	[0]		
description	This functions writes the complete digital output data of the PLC kernel to the hardware. This function can be used to force an additional hardware output update within the PLC task or from interrupt level			

SysIo_ResetDigitalOut

name	**SysIo_ResetDigitalOut**		*type*	Function
return value 1	*type*	BOOL	Resets a single digital output channel.	
	value	FALSE	Function skipped. An error occurred	
		TRUE	Output channel successfully reset	
input value 1	*name*	**OutputByteNr**		
	type	UINT	Selects the output byte	
	value	[0 .. 1]		
input value 2	*name*	**OutputBitNr**		
	type	UINT	Selects the bit of the corresponding output byte	
	value	[0 .. 7]		
description	This function resets a single digital output channel. 1) The output is reset directly on the hardware without waiting for the IO update cycle at the end of the PLC loop 2) The corresponding memory location for the output data is also cleared			

SysIo_SetDigitalOut

name	**SysIo_SetDigitalOut**	*type*	Function
return value 1	*type*	BOOL	Sets a single digital output channel.
	value	FALSE	Function skipped. An error occurred
		TRUE	Output channel successfully set
input value 1	*name*	**OutputByteNr**	
	type	UINT	Selects the output byte
	value	[0 .. 1]	
input value 2	*name*	**OutputBitNr**	
	type	UINT	Selects the bit of the corresponding output byte
	value	[0 .. 7]	
description	This function sets a single digital output channel. 1) The output is set directly on the hardware without waiting for the IO update cycle at the end of the PLC loop 2) The corresponding memory location for the output data is also set		

SysIo_GetDigitalIn

name	**SysIo_GetDigitalIn**	*type*	Function
return value 1	*type*	BOOL	reads the state of a digital input bit directly from hardware
	value	FALSE	Bit is low level
		TRUE	Bit is high level
input value 1	*name*	**OutputByteNr**	
	type	UINT	Selects the input byte
	value	[0 .. 1]	
input value 2	*name*	**OutputBitNr**	
	type	UINT	Selects the bit of the corresponding input byte
	value	[0 .. 7]	
description	This function reads the state of a digital input bit directly from hardware. The input data is not updated		

SysIo_GetErrStatus

name	**SysIo_GetErrStatus**	*type*	Function
return value 1	*type*	UINT	This function returns the status of output drivers.
	value	[0 … 65535]	0: no error detected >0: output driver disabled because of overload condition detected
input value 1	*name*	**Mode**	
	type	UINT	Reserved for future use, set to 0
	value	[0]	
description	This function returns the status of output drivers. 0 no error >0 output driver disabled because of overload condition detected		

SysIo_SetWdtMode

name	**SysIo_SetWdtMode**	*type*	Function
return value 1	*type*	BOOL	Modifies the mode for triggering the watchdog for digital output drivers
	value	FALSE	Function skipped
		TRUE	Function successfully executed
input value 1	*name*	**SetMask**	
	type	UINT	Sets the corresponding bits of WDT mode
	value	[0 .. 65535]	
input value 2	*name*	**ClearMask**	
	type	UINT	Clears the corresponding bits of WDT mode
	value	[0 .. 65535]	
description	Modifies the mode for triggering the watchdog for digital output drivers: Bit 0 : Wdt is triggered on I/O update from PLC cycle Bit 1 : Wdt is triggered if CoDeSys application is topped on breakpoint Bit 8 : Wdt is triggered from operation system in any case		

SysIo_WrAllAnalogOut

name	**SysIo_WrAllAnalogOut**	*type*	Function
return value 1	*type*	BOOL	Write all digital output data
	value	FALSE	An error occurred. Function skipped
		TRUE	All digital outputs written successfully
input value 1	*name*	**Mode**	
	type	UINT	Reserved for future use, set to 0
	value	[0]	
description	This functions writes the complete analog output data of the PLC kernel to the hardware. This function can be used to force an additional hardware output update within the PLC task or from interrupt level		

SysIo_WrAnalogOut

name	**SysIo_WrAnalogOut**	*type*	Function
return value 1	*type*	BOOL	Writes a single analog output
	value	FALSE	Function skipped. An error occurred
		TRUE	Output channel successfully set
input value 1	*name*	**Channel**	
	type	UINT	Analog Output Channel to write
	value	[0 .. 1]	
input value 2	*name*	**Value**	
	type	INT	Value to write to this output
	value	[-32768 .. 32767]	
description	This functions writes a single analog output with new data to the hardware.		

FBESysJ1939.lib

Functions	***Description***
J1939_InitCan	Initializes the Basic-CAN-Interface
J1939_IsRxMsg	Checks whether there are received J1939 frames
J1939_RxMsg	Receives a J1939 message
J1939_TxMsg	Transmits a J1939 message

Hardware Reference

hipecs PLC1010/1020/1030		
	Available CAN Interfaces	
CAN Interface Nr.	0	1
CoDeSys Enumeration	CAN 0	CAN 1
CAN Mode	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939

hipecs CORE10 Modules				
		Available CAN Interfaces		
CAN Interface Nr.	0	1	2	3
CoDeSys Enumeration	CAN 0	CAN 1	CAN 2	CAN 3
CAN Mode	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939	CANopen / Basic CAN / J1939

Data types defined for J1939 library

```
TYPE J1939 :
STRUCT
          Priority        : BYTE;    (* Priority *)
          PGN             : DWORD; (* Parameter Group Number *)
          SAddr           : BYTE;    (* Source Address *)
          Data            : ARRAY [0..7] OF  BYTE;
          Len             : UINT;
END_STRUCT
END_TYPE
```

J1939_InitCan

name	**J1939_InitCAN**			*type*	Function
return value	*type*	BOOL	Returns the result state.		
	value	TRUE	*Init successful*		
		FALSE	*Function skipped. Init failed.*		
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
description	Initializes the Basic J1939 Interface. In order to J1939 add a CANopen master to the system configuration. Set baud rate within CANopen master.				

J1939_IsRxMsg

name	**J1939_IsRxMsg**			*type*	Function
return value	*type*	BOOL	Returns the result state.		

	value	TRUE	*One or more messages available*
		FALSE	*No message available*
input value 1	*name*	**Node**	
	type	SYSCAN_CANNODE	CAN Interface Number
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*
description	Functions check if there is one or more messages in the receiver buffer.		

J1939_RxMsg

name	**J1939_RxMsg**			*type*	Function
return value	*type*	J1939			
	value	[..]	*Returns data in J1939 message format*		
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
description	Receives a message from the CAN Rx buffer				

J1939_TxMsg

name	**J1939_TxMsg**			*type*	Function
return value 1	*type*	BOOL	Send the message to the CAN Tx buffer		
	value	FALSE	Function skipped. An error occurred		
		TRUE	data valid and sent to buffer		
input value 1	*name*	**Node**			
	type	SYSCAN_CANNODE	CAN Interface Number		
	value	CAN 0, CAN 1	*See hardware reference table for valid CAN interface*		
input value 2	*name*	**TxMsg**			
	type	J1939	Message in the J1939 format		
	value	[...]			
description	This functions send a message in the J1939 data type format to the CAN transmit buffer				