CoDeSysControl Compact for SIL2

Document Version 3.4.2.0

CONTENT

1 CmpMgr

This implementation of the component manager is used typically in environments with the possibility to load and unload components dynamically. But the components must not loaded dynamically, they can be linked static too.

Compiler Switch

- #define STATIC_LINK All components are linked statically to the component manager during compile time.
- #define MIXED_LINK All components are linked static, but can be extended by dynamically linked components.
- #define DYNAMIC_LINK [Default] All components exists as separate compiled modules. So this modules can be loaded dynamically.
- #define CPLUSPLUS All components are linked together via C++ classes.
- #define CM_SYSTARGET_DISABLE_OVERLOADABLE_FUNCTIONS Switch to disable the possibility to overload the SysTarget functions
- #define CM_DISABLE_API_SIGNATURE_CHECK Switch to disable the signature check of all api functions
- #define CM_DISABLE_API_VERSION_CHECK Switch to disable the version check of all api functions
- #define CM_NO_EXIT Switch to disable definite exit or shutdown process of the runtime system
- #define CM_NO_DYNAMIC_COMPONENTS Switch to disable dynamic loadable components

1.1 CMItf

Interface of the component manager

The component manager component is the central component of the runtime system. The component manager is responsible for:

- startup and shutdown of the runtime system
- loading and unloading all components
- intializing all components
- linking all functions, including the external IEC library functions
- checking the consistency of calling a function (matching function prototypes)

The list of components that should be loaded can be specified in different ways.

If the components are linked static, there are two different ways to specify the component list:

- Static list: The list of static loaded components can be specified as a calling option for the CMInit() interface method. The name of the component on the entry funtion of each component must be specified. See the description of the CMInit() method for detailed information.
- Settings: The settings component has a settings interface, where the component list can be specified. Here only the component name must be specified. In this case, the function MainLoadComponent() must be impleemnted in the Main module and here all static linked components must be added to the list. The advantage of this method is that in the settings you can specify, which static component can be loaded or not. This is quite flexible.

For dynamically linked components, the component list can only be specified in the settings component.

External IEC library functions are managed by the component manager too. Each interface have to specify, if it could be used for IEC code or not in the m4 Interface definition.

An additional feature is the possibility to use all components with a C++ interface. Here you can implement your own component in C++ and can use all other runtime components object oriented with a C++ interface.

1.1.1 Define: MAX_COMPONENT_NAME

Category: Static defines
Type:
Define: MAX_COMPONENT_NAME
Key: 32
Maximum length of component name

1.1.2 Define: MAX_API_NAME

Category: Static defines
Type:
Define: MAX_API_NAME
Key: 64
Maximum length of api function name

1.1.3 Define: CM_API_VERSION_CHECK_MASK

Condition: #ifndef CM_API_VERSION_CHECK_MASK
Category: Static defines
Type:
Define: CM_API_VERSION_CHECK_MASK
Key: UINT32_C
Significant mask to check different versions of an api function. If the significant parts of the version don't match, the api versions don't match!

1.1.4 Define: CM_API_VERSION_CHECK_MINIMUM

Condition: #ifndef CM_API_VERSION_CHECK_MINIMUM
Category: Static defines
Type:
Define: CM_API_VERSION_CHECK_MINIMUM
Key: UINT32_C
Specifies the minimum supported version of all api functions.

1.1.5 Define: CM_EXTLIB_API_VERSION_CHECK_MASK

Condition: #ifndef CM_EXTLIB_API_VERSION_CHECK_MASK
Category: Static defines
Type:
Define: CM_EXTLIB_API_VERSION_CHECK_MASK
Key: UINT32_C
Significant mask to check different versions of an external library api function. If the significant parts of the version don't match, the api versions don't match!

1.1.6 Define: CM_EXTLIB_API_VERSION_CHECK_MINIMUM

Condition: #ifndef CM_EXTLIB_API_VERSION_CHECK_MINIMUM
Category: Static defines
Type:
Define: CM_EXTLIB_API_VERSION_CHECK_MINIMUM
Key: UINT32_C
Specifies the minimum supported version of all external library api functions.

1.1.7 Define: EVT_CmpMgr_PrepareShutdown

Category: Events
Type:
Define: EVT_CmpMgr_PrepareShutdown
Key: MAKE_EVENTID
Event is sent before shutdown of the runtime system

1.1.8 Define: EVT_CmpMgr_PrepareExitComm

Category: Events
Type:
Define: EVT_CmpMgr_PrepareExitComm
Key: MAKE_EVENTID
Event is sent before exit the communication servers during shutdown

1.1.9 Define: EVT_CmpMgr_PrepareExitTasks
Category: Events
Type:
Define: EVT_CmpMgr_PrepareExitTasks
Key: MAKE_EVENTID
Event is sent before exit all tasks during shutdown

1.1.10 Define: EVT_CmpMgr_Exit3
Category: Events
Type:
Define: EVT_CmpMgr_Exit3
Key: MAKE_EVENTID
Event is sent before exit of the runtime system during shutdown

1.1.11 Define: EVT_CmpMgr_PrepareExit
Category: Events
Type:
Define: EVT_CmpMgr_PrepareExit
Key: MAKE_EVENTID
Event is sent before exit of the runtime system during shutdown

1.1.12 Define: EVT_CmpMgr_Exit2
Category: Events
Type:
Define: EVT_CmpMgr_Exit2
Key: MAKE_EVENTID
Event is sent before exit of the runtime system during shutdown

1.1.13 Define: EVT_CmpMgr_DisableOperation
Category: Events
Type:
Define: EVT_CmpMgr_DisableOperation
Key: MAKE_EVENTID
Event is sent to disable operations. Each operation is defined in the corresponding Itf.h file of the component, which is specified by its ComponentID

1.1.14 Define: RTS_CMINIT_OPTION_SETTINGSFILEISOPTIONAL

1.1.15 Define: CMPTYPE_STANDARD
Category: ComponentType
Type:
Define: CMPTYPE_STANDARD
Key: UINT16_C
A component can only be a member of type CMPTYPE_STANDARD or type CMPTYPE_SAFETY. To separate the components in detail, we use the types CMPTYPE_STATIC, CMPTYPE_DYNAMIC, CMPTYPE_SYSTEM or CMPTYPE_IEC.

1.1.16 Define: RTS_CS_UNKNOWN

1.1.17 Define: CM_HOOK_TYPE_NORMAL
Category: COMM_CYCLE_HOOK types
Type:
Define: CM_HOOK_TYPE_NORMAL
Key: 0
- CM_HOOK_TYPE_NORMAL: Cyclic call of COMM_CYCLE_HOOK
- CM_HOOK_TYPE_FLASH_ACCESS: Additional call during flash accesses

1.1.18 Define: CM_CNF_STATIC
Category: ComponentNameFlags
Type:
Define: CM_CNF_STATIC
Key: 0
- CM_CNF_STATIC: Name is stored in a static buffer
- CM_CNF_DYNAMIC: Name is stored in a dynamic buffer

1.1.19 Typedef: EVTPARAM_CmpMgr_Shutdown

Stuctname: EVTPARAM_CmpMgr_Shutdown
Category: Event parameter
Typedef: typedef struct { void* pdummy; } EVTPARAM_CmpMgr_Shutdown;

1.1.20 Typedef: EVTPARAM_CmpMgr_DisableOperation

Stuctname: EVTPARAM_CmpMgr_DisableOperation
Category: Event parameter
Typedef: typedef struct { CMPID cmpId; RTS_UI32 ulOperation; CMPID cmpIdDisabled; RTS_I32 bDisable; } EVTPARAM_CmpMgr_DisableOperation;

1.1.21 Typedef: LicenseFunctions

Stuctname: LicenseFunctions
LicenseFunctions
Typedef: typedef struct tagLicenseFunctions { RTS_IEC_DWORD dwStructSize; RTS_IEC_BYTE *pfGetUserLicenseValue; RTS_IEC_BYTE *pfConfDynLicChallenge; RTS_IEC_BYTE *pfReqDynLicChallenge; RTS_IEC_DWORD dwVersion; } LicenseFunctions;

1.1.22 Typedef: cmaddcomponent_struct

Stuctname: cmaddcomponent_struct
cmaddcomponent
Typedef: typedef struct tagcmaddcomponent_struct { RTS_IEC_STRING *pszComponent; VAR_INPUT RTS_IEC_UDINT udiCmpId; VAR_INPUT RTS_IEC_UDINT udiVersion; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_HANDLE CMAddComponent; VAR_OUTPUT } cmaddcomponent_struct;

1.1.23 Typedef: cmexitcomponent_struct

Stuctname: cmexitcomponent_struct
cmexitcomponent
Typedef: typedef struct tagcmexitcomponent_struct { RTS_IEC_HANDLE hComponent; VAR_INPUT RTS_IEC_RESULT CMExitComponent; VAR_OUTPUT } cmexitcomponent_struct;

1.1.24 Typedef: cmremovecomponent_struct

Stuctname: cmremovecomponent_struct
cmremovecomponent
Typedef: typedef struct tagcmremovecomponent_struct { RTS_IEC_HANDLE hComponent; VAR_INPUT RTS_IEC_RESULT CMRemoveComponent; VAR_OUTPUT } cmremovecomponent_struct;

1.1.25 Typedef: cmregisterlicensefunctions_struct

Stuctname: cmregisterlicensefunctions_struct
cmregisterlicensefunctions
Typedef: typedef struct tagcmregisterlicensefunctions_struct { LicenseFunctions licenseFunctions; VAR_INPUT RTS_IEC_RESULT CMRegisterLicenseFunctions; VAR_OUTPUT } cmregisterlicensefunctions_struct;

1.1.26 Typedef: cmutlwstrcpy_struct

Stuctname: cmutlwstrcpy_struct
cmutlwstrcpy
Typedef: typedef struct tagcmutlwstrcpy_struct { RTS_IEC_WSTRING *pwszDest; VAR_INPUT RTS_IEC_DINT nDestSize; VAR_INPUT RTS_IEC_WSTRING *pwszSrc; VAR_INPUT RTS_IEC_RESULT CMUtlwstrcpy; VAR_OUTPUT } cmutlwstrcpy_struct;

1.1.27 Typedef: cmutlstricmp_struct

Stuctname: cmutlstricmp_struct
cmutlstricmp
Typedef: typedef struct tagcmutlstricmp_struct { RTS_IEC_STRING *pszString1; VAR_INPUT RTS_IEC_STRING *pszString2; VAR_INPUT RTS_IEC_DINT CMUtlStrICmp; VAR_OUTPUT } cmutlstricmp_struct;

1.1.28 Typedef: cmunregistergetuserlicensevalue_struct

Stuctname: cmunregistergetuserlicensevalue_struct
Typedef: typedef struct tagcmunregistergetuserlicensevalue_struct { RTS_IEC_BYTE *pfGetUserLicenseValue; VAR_INPUT RTS_IEC_RESULT CMUnregisterGetUserLicenseValue; VAR_OUTPUT } cmunregistergetuserlicensevalue_struct;

1.1.29 Typedef: cmutlcwstrcpy_struct

Stuctname: cmutlcwstrcpy_struct
cmutlcwstrcpy
Typedef: typedef struct tagcmutlcwstrcpy_struct { RTS_IEC_CWCHAR *pcwszDest; VAR_INPUT RTS_IEC_DINT nDestSize; VAR_INPUT RTS_IEC_CWCHAR *pcwszSrc; VAR_INPUT RTS_IEC_RESULT CMUtlcwstrcpy; VAR_OUTPUT } cmutlcwstrcpy_struct;

1.1.30 Typedef: cmunregisterlicensefunctions_struct

Stuctname: cmunregisterlicensefunctions_struct
cmunregisterlicensefunctions
Typedef: typedef struct tagcmunregisterlicensefunctions_struct { LicenseFunctions licenseFunctions; VAR_INPUT RTS_IEC_RESULT CMUnregisterLicenseFunctions; VAR_OUTPUT } cmunregisterlicensefunctions_struct;

1.1.31 Typedef: cminitcomponent_struct

Stuctname: cminitcomponent_struct
cminitcomponent
Typedef: typedef struct tagcminitcomponent_struct { RTS_IEC_HANDLE hComponent; VAR_INPUT RTS_IEC_RESULT CMInitComponent; VAR_OUTPUT } cminitcomponent_struct;

1.1.32 Typedef: cmgetcomponentbyname_struct

Stuctname: cmgetcomponentbyname_struct
cmgetcomponentbyname
Typedef: typedef struct tagcmgetcomponentbyname_struct { RTS_IEC_STRING *pszComponent; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_HANDLE CMGetComponentByName; VAR_OUTPUT } cmgetcomponentbyname_struct;

1.1.33 Typedef: cmshutdown_struct

Stuctname: cmshutdown_struct
cmshutdown
Typedef: typedef struct tagcmshutdown_struct { RTS_IEC_UDINT dwReason; VAR_INPUT RTS_IEC_RESULT CMShutDown; VAR_OUTPUT } cmshutdown_struct;

1.1.34 Typedef: cmutlsafestrcpy_struct

Stuctname: cmutlsafestrcpy_struct
cmutlsafestrcpy
Typedef: typedef struct tagcmutlsafestrcpy_struct { RTS_IEC_STRING *pszDest; VAR_INPUT RTS_IEC_DINT nDestSize; VAR_INPUT RTS_IEC_STRING *pszSrc; VAR_INPUT RTS_IEC_RESULT CMUtlSafeStrCpy; VAR_OUTPUT } cmutlsafestrcpy_struct;

1.1.35 Typedef: cmregistergetuserlicensevalue_struct

Stuctname: cmregistergetuserlicensevalue_struct
Typedef: typedef struct tagcmregistergetuserlicensevalue_struct { RTS_IEC_BYTE *pfGetUserLicenseValue; VAR_INPUT RTS_IEC_RESULT CMRegisterGetUserLicenseValue; VAR_OUTPUT } cmregistergetuserlicensevalue_struct;

1.1.36 Typedef: cmloadcomponent_struct

Stuctname: cmloadcomponent_struct
Typedef: typedef struct { RTS_IEC_STRING *pszComponent; RTS_IEC_RESULT *pResult; RTS_IEC_HANDLE hComponent; } cmloadcomponent_struct;

1.1.37 CMInit

*RTS_RESULT CMInit (char *pszSettingsFile, StaticComponent *pStaticComponents)*
Called to initialize the component manager

pszSettingsFile [IN]
Pointer to name of the configuration file

pStaticComponents [IN]

Pointer to list of components with name and entry routine to initialize without configuration

Result

Error code:

- ERR_OK
- ERR_FAILED: One or more components failed to load
- ERR_ID_MISMATCH: Signature mismatch of the SysTargetIDs

1.1.38 CMInit2

*RTS_RESULT CMInit2 (char *pszSettingsFile, StaticComponent *pStaticComponents, int bSettingsFileIsOptional)*

Called to initialize the component manager

pszSettingsFile [IN]

Pointer to name of the configuration file

pStaticComponents [IN]

Pointer to list of components with name and entry routine to initialize without configuration

bSettingsFileIsOptional [IN]

Specifies, if the Settingsfile is only used optionally

Result

error code

1.1.39 CMInit3

*RTS_RESULT CMInit3 (char *pszSettingsFile, StaticComponent *pStaticComponents, RTS_UI32 options)*

Called to initialize the component manager

pszSettingsFile [IN]

Pointer to name of the configuration file

pStaticComponents [IN]

Pointer to list of components with name and entry routine to initialize without configuration

options [IN]

Options for the init sequence of the component manager. See category "CMInit options" for details.

Result

error code

1.1.40 CMExit

RTS_RESULT CMExit (void)

Called to deinitialize the component manager

Result

error code

1.1.41 CMSetExit

RTS_RESULT CMSetExit (void)

Set a flag for main loop to exit

Result

error code

1.1.42 CMGetExit

RTS_RESULT CMGetExit (void)

Get a flag to exit main loop

Result

ERR_OK, if exit flag is set ERR_FAILED else

1.1.43 CMDebugOut

*RTS_RESULT CMDebugOut (const char *szFormat, ...)*

Can be used for debug outputs on a console.

Result

ERR_OK or ERR_NOT_SUPPORTED, if SysOut component is not available

1.1.44 CMDebugOutArg

*RTS_RESULT CMDebugOutArg (const char *szFormat, va_list *pargList)*
Can be used for debug outputs on a console.
Result
ERR_OK or ERR_NOT_SUPPORTED, if SysOut component is not available

1.1.45 CMRegisterAPI

*RTS_RESULT CMRegisterAPI (const CMP_EXT_FUNCTION_REF *pExpTable, RTS_UINTPTR dummy, int bExternalLibrary, RTS_UI32 cmpId)*
Called to register a list of component API functions at the component manager
pExpTable [IN]
Table of functions to be registered
CmpId [IN]
Component identifier
bExternalLibrary [IN]
Can be used as external library in the plc program
Result
error code

1.1.46 CMRegisterAPI2

*RTS_RESULT CMRegisterAPI2 (const char *pszAPIName, RTS_VOID_FCTPTR pfAPIFunction, int bExternalLibrary, RTS_UI32 ulSignatureID, RTS_UI32 ulVersion)*
Called to register a component API function at the component manager
pszAPIName [IN]
Name of the API routine
pfAPIFunction [IN]
Function pointer of the API routine
bExternalLibrary [IN]
Can be used as external library in the plc program
ulSignatureID [IN]
SignatureID of the function prototype
ulVersion [IN]
Actual supported implementation version
Result
error code

1.1.47 CMGetAPI3

*RTS_RESULT CMGetAPI3 (char *pszAPIName, RTS_VOID_FCTPTR *ppfAPIFunction, int importOptions, RTS_UI32 *pulSignatureID, RTS_UI32 *pulVersion)*
Called to get an API function of another component
pszAPIName [IN]
Name of the API routine
ppfAPIFunction [OUT]
Returned the function pointer of the API routine
pulSignatureID [INOUT]
SignatureID of the function prototype requested and returns the actual signature implemented
pulVersion [INOUT]
Implementation version requested and returns the actual version implemented
importOptions [IN]
Import options. See CM_IMPORT_xx defines in CmpStd.h for details
Result
error code

1.1.48 CMReleaseAPI

RTS_RESULT CMReleaseAPI (RTS_VOID_FCTPTR pfAPIFunction)
Called to release an API function of another component
pfAPIFunction [IN]
Result
error code

1.1.49 CMLoadComponent

*RTS_HANDLE CMLoadComponent (char *pszComponent, RTS_RESULT *pResult)*

Called to load a component. Can also be called during runtime. ATTENTION: If component has references to other components, referenced components must be loaded first!

pszComponent [IN]
Name of the component

pResult [INOUT]
Pointer to error code

Result
Component handle or RTS_INVALID_HANDLE, if an error is occured

1.1.50 CMInitComponent

RTS_RESULT CMInitComponent (RTS_HANDLE hComponent)

Called to initialize a component after it was loaded during runtime.

hComponent [IN]
Handle of the component

Result
error code

1.1.51 CMExitComponent

RTS_RESULT CMExitComponent (RTS_HANDLE hComponent)

Called to exit a component after it was loaded during runtime. After this it can be unloaded, if no other components references this component

hComponent [IN]
Handle of the component

Result
error code

1.1.52 CMUnloadComponent

RTS_RESULT CMUnloadComponent (RTS_HANDLE hComponent)

Called to unload a component. Can also be called during runtime. ATTENTION: CMExitComponent must be called before unloading the component!

hComponent [IN]
Handle of the component

Result
error code

1.1.53 CMAddComponent

*RTS_HANDLE CMAddComponent (COMPONENT_ENTRY *pComponent, RTS_RESULT *pResult)*

Add a component to the list

pComponent [IN]
Pointer to component description

pResult [OUT]
Pointer to error code

Result
returns a handle to the component

1.1.54 CMRemoveComponent

RTS_RESULT CMRemoveComponent (RTS_HANDLE hComponent)

Remove a component from the list

hComponent [IN]
Handle to the component

Result
error code

1.1.55 CMGetComponentByName

*RTS_HANDLE CMGetComponentByName (char *pszCmpName, RTS_RESULT *pResult)*

Get the component handle of a component specified by name.

pszCmpName [IN]
Name of the component

pResult [INOUT]
Pointer to error code

Result
Component handle or RTS_INVALID_HANDLE, if an error is occured

1.1.56 CMGetCmpId

*RTS_RESULT CMGetCmpId (char *pszCmpName, CMPID *pCmpId)*

Get the component id of a component specified by name. A component id always consists of the unique vendor id as high word and the specific component id as low word. So each component can be assigned to the specific vendor.

pszCmpName [IN]
Name of the component

pCmpId [OUT]
Component id

Result
error code

1.1.57 CMGetCmpName

*RTS_RESULT CMGetCmpName (CMPID CmpId, char *pszCmpName, int iMaxCmpName)*

Get the component name of a component specified by the component id.

pCmpId [IN]
Component id

pszCmpName [OUT]
Name of the component

iMaxCmpName [IN]
Max length of the component name buffer

Result
error code

1.1.58 CMGetComponent

COMPONENT_ENTRY CMGetComponent (CMPID CmpId)*

Get the component description of a component specified by the component id.

pCmpId [IN]
Component id

Result
Pointer to COMPONENT_ENTRY

1.1.59 CMGetNumOfComponents

int CMGetNumOfComponents (void)

Returns the number of registered components

Result
Number of components

1.1.60 CMGetComponentByIndex

COMPONENT_ENTRY CMGetComponentByIndex (int iIndex)*

Returns the registered component specified by index

iIndex [IN]
Index of the component in the list

Result
Pointer to component or NULL if index ist out of range

1.1.61 CMGetFirstComponent

COMPONENT_ENTRY CMGetFirstComponent (RTS_RESULT *pResult)*

Returns the first component entry

pResult [INOUT]

Pointer to error code

Result

Pointer to component or NULL if no component available

1.1.62 CMGetNextComponent

COMPONENT_ENTRY CMGetNextComponent (COMPONENT_ENTRY *pCmp, RTS_RESULT *pResult)*

Returns the registered component specified by index

pCmp [IN]

Pointer to the previous component

pResult [INOUT]

Pointer to error code

Result

Pointer to component or NULL if end of list is reached

1.1.63 CMCreateInstance

IBase CMCreateInstance (CLASSID ClassId, RTS_RESULT *pResult)*

Create an instance of a specified class.

ClassId [IN]

ClassId of the class to create an object

pResult [OUT]

Pointer to error code for result

Result

Pointer to IBase interface of the object

1.1.64 CMDeleteInstance

*RTS_RESULT CMDeleteInstance (IBase *pIBase)*

Delete an instance.

pIBase [IN]

Pointer to IBase interface

Result

error code

1.1.65 CMDeleteInstance2

*RTS_RESULT CMDeleteInstance2 (CLASSID ClassId, IBase *pIBase)*

Delete an instance.

ClassId [IN]

ClassId of the instance

pIBase [IN]

Pointer to IBase interface

Result

error code

1.1.66 CMGetInstance

IBase CMGetInstance (CLASSID ClassId, OBJID ObjId, RTS_RESULT *pResult)*

Get instance of a class.

ClassId [IN]

ClassId of the class to create an object

ObjId [IN]

Index number of the instance

pResult [OUT]

Pointer to error code for result

Result

Pointer to IBase interface of the object

1.1.67 CMRegisterInstance

*RTS_RESULT CMRegisterInstance (CLASSID ClassId, OBJID ObjId, IBase *pIBase)*

Register an exisiting instance to the component manager.

ClassId [IN]
ClassId of the class to create an object
ObjId [IN]
Index number of the instance
pResult [OUT]
Pointer to error code for result
Result
error code

1.1.68 CMUnregisterInstance

*RTS_RESULT CMUnregisterInstance (IBase *pIBase)*
Unregister an existing instance from component manager.
pIBase [IN]
Pointer to IBase interface
Result
error code

1.1.69 CMGetInstanceList

RTS_RESULT CMGetInstanceList (ITFID ItfId, RTS_HANDLE hIBasePool)
Get a list of instances, that implements the specified interface.
ItfId [IN]
Id of the interface
hIBasePool [INOUT]
Must be a MemPool handle. All instances that implements the interface are stored in this pool as result.
Result
error code

1.1.70 CMCallHook

RTS_RESULT CMCallHook (RTS_UI32 ulHook, RTS_UINTPTR ulParam1, RTS_UINTPTR ulParam2, int bReverse)
Called to call all components with the specified hook
ulHook [IN]
Hook (for definition, look into CmpItf.h)
ulParam1 [IN]
First parameter. Hook dependant
ulParam2 [IN]
Second parameter. Hook dependant
bReverse [IN]
Called the components in the opposite order as they were loaded
Result
error code

1.1.71 CMCallHook2

RTS_RESULT CMCallHook2 (RTS_UI16 usComponentTypeMask, RTS_UI32 ulHook, RTS_UINTPTR ulParam1, RTS_UINTPTR ulParam2)
Called to call all components with the specified type provided with the specified hook
usComponentTypeMask [IN]
ComponentType mask. See corresponding category
ulHook [IN]
Hook (for definition, look into CmpItf.h)
ulParam1 [IN]
First parameter. Hook dependant
ulParam2 [IN]
Second parameter. Hook dependant
Result
error code

1.1.72 CMIsDemo

int CMIsDemo (void)

Routine to check, if runtime runs in demo mode

Result

1=Demo mode, 0=No demo

1.1.73 CMInitEnd

RTS_RESULT CMInitEnd (void)

Routine to complete the component manager initialization. Contains calling the hooks from CH_INIT2 to CH_INIT_COMM.

Result

- ERR_OK
- ERR_DUPLICATE: If init end is still done, this error message will be returned

1.1.74 CMCheckSysTargetSignature

RTS_RESULT CMCheckSysTargetSignature (void)

Functtion checks, if the SysTarget component is originally signed and was not modified.

Result

error code: ERR_OK: SysTarget is signed, ERR_FAILED: SysTarget was modified or is unsigned

1.1.75 CMRegisterLicenseFunctions

*RTS_RESULT CMRegisterLicenseFunctions (LicenseFunctions *pLicenseFunctions)*

Register the IEC function pointers from the license manager lib

pLicenseFunctions [IN]

Pointer to all IEC function pointers of the license manager lib

Result

error code

1.1.76 CMUnregisterLicenseFunctions

*RTS_RESULT CMUnregisterLicenseFunctions (LicenseFunctions *pLicenseFunctions)*

Unregister the IEC function pointers from the license manager lib

pLicenseFunctions [IN]

Pointer to all IEC function pointers of the license manager lib

Result

error code

1.1.77 CMGetUserLicenseValue

*RTS_UI32 CMGetUserLicenseValue (RTS_UI32 ulLicenseID, RTS_RESULT *pResult)*

Function to check the license of a feature

ulLicenseID [IN]

HIGHWORD: VendorID, LOWWORD: FeatureID

pResult [OUT]

Pointer to error code

Result

0: Not licensed, !0: Specific license ID

1.1.78 CMReqDynLicChallenge

*RTS_UI32 CMReqDynLicChallenge (RTS_UI32 ulLicenseID, RTS_UI32 ulNewLicenseValue, RTS_RESULT *pResult)*

ulLicenseID [IN]

HIGHWORD: VendorID, LOWWORD: FeatureID

ulNewLicenseValue [IN]

pResult [OUT]

Pointer to error code

Result

1.1.79 CMConfDynLicChallenge

*int CMConfDynLicChallenge (RTS_UI32 ulLicenseID, RTS_UI32 ulNewLicenseValue, RTS_UI32 ulChallenge, RTS_RESULT *pResult)*

ulLicenseID [IN]
HIGHWORD: VendorID, LOWWORD: FeatureID

ulNewLicenseValue [IN]

ulChallenge [IN]

pResult [OUT]
Pointer to error code

Result

1.1.80 CMCallExtraCommCycleHook

RTS_RESULT CMCallExtraCommCycleHook (RTS_UINTPTR ulParam1, RTS_UINTPTR ulParam2)

Function to call the CommCycleHook of all components from outside the CM, if neccessary. This function is intended to be called on singletasking systems (with CmpScheduleTimer or CmpScheduleEmbedded) during long lasting operations. For example SysFlash calls this function to keep the communication alive during writing long memory areas. Before calling the CommCycleHook, the function internally checks the following conditions: - Is SysTask implemented? - Is the less then the configured time since the last call of the CommCycleHook elapsed? If one of this conditions is true, the hook is not called. This allows to use this function without doing these checks locally in the caller. On multitasking systems this function has no effect. Use AsyncServices instead.

ulParam1 [IN]
Type of the COMM_CYCLE_HOOK. See CM_HOOK_TYPE... types.

ulParam2 [IN]
Second parameter. Hook dependant, typically 0

Result
error code

1.1.81 CMIsOperationDisabled

*RTS_RESULT CMIsOperationDisabled (CMPID cmpId, RTS_UI32 ulOperation, CMPID *pCmpIdDisabled)*

Function can be called from each component to disable single operations. If this function is called, the event EVT_CmpMgr_DisableOperation is sent to disable this operation from any component or IEC program.

Result

- ERR_OK: Operation is disabled
- ERR_FAILED: Operation is enabled [default]
- ERR_NOT_SUPPORTED: Cannot be retrieved, because the event is not available

1.1.82 CMProfiling

*RTS_RESULT CMProfiling (char *pszInfo, CMPID cmpId, char *pszComponentName, char *pszModuleName, RTS_I32 nLine, RTS_I32 iID)*

Central function for profiling

Result
Error code

1.1.83 CMGetAPI

*RTS_RESULT CMGetAPI (char *pszAPIName, RTS_VOID_FCTPTR *ppfAPIFunction, RTS_UI32 ulSignatureID)*

Called to get an API function of another component

pszAPIName [IN]
Name of the API routine

ppfAPIFunction [OUT]
Returned the function pointer of the API routine

ulSignatureID [IN]
SignatureID of the function prototype we expected

Result

error code

1.1.84 CMGetAPI2

*RTS_RESULT CMGetAPI2 (char *pszAPIName, RTS_VOID_FCTPTR *ppfAPIFunction, int importOptions, RTS_UI32 ulSignatureID, RTS_UI32 ulVersion)*

Called to get an API function of another component

pszAPIName [IN]
Name of the API routine

ppfAPIFunction [OUT]
Returned the function pointer of the API routine

ulSignatureID [IN]
SignatureID of the function prototype we expected

ulVersion [IN]
Actual implementation version requested

importOptions [IN]
Import options. See CM_IMPORT_xx defines in CmpStd.h for details

Result
error code

1.2 CMUtilsItf

Interface for utility functions of the component manager

The utility functions can be used by all components, because they are always included in the component manager.

1.2.1 Define: RTS_STATIC_STRING_LEN

Category: Static defines
Type:
Define: RTS_STATIC_STRING_LEN
Key: 32
Static length of a string. Is ised for RTS_STRING_CLASS.

1.2.2 Define: CMUTL_CPY_COMM2BUFFER

Category: Copy mode
Type: Int
Define: CMUTL_CPY_COMM2BUFFER
Key: 1
Modes to copy strings. Copy byte packed stream to a word stream. HI byte is filled with 0.

1.2.3 Define: CMUTL_CPY_BUFFER2COMM

Category: Copy mode
Type: Int
Define: CMUTL_CPY_BUFFER2COMM
Key: 2
Modes to copy strings. Copy word stream to a byte packed stream. HI byte is removed.

1.2.4 Define: CMUTL_CPY_DEFAULT

Category: Copy mode
Type: Int
Define: CMUTL_CPY_DEFAULT
Key: 3
Modes to copy strings. Copy stream unchanged.

1.2.5 Define: MAX_LONG_BIN_STRING_LEN

Category:
Type:
Define: MAX_LONG_BIN_STRING_LEN
Key: sizeof(RTS_I32
Maximum length of the long binary string

1.2.6 Define: MAX_LONG_DEC_STRING_LEN

Category:

Type:
Define: MAX_LONG_DEC_STRING_LEN
Key: 11
Maximum length of the long decimal string

1.2.7 Define: MAX_LONG_HEX_STRING_LEN

Category:
Type:
Define: MAX_LONG_HEX_STRING_LEN
Key: 11
Maximum length of the long hexadecimal string

1.2.8 Define: MAX_ULONG_BIN_STRING_LEN

Category:
Type:
Define: MAX_ULONG_BIN_STRING_LEN
Key: sizeof(RTS_UI32
Maximum length of the unsigned long binary string

1.2.9 Define: MAX_ULONG_DEC_STRING_LEN

Category:
Type:
Define: MAX_ULONG_DEC_STRING_LEN
Key: 11
Maximum length of the unsigned long decimal string

1.2.10 Define: MAX_ULONG_HEX_STRING_LEN

Category:
Type:
Define: MAX_ULONG_HEX_STRING_LEN
Key: 11
Maximum length of the long unsigned hexadecimal string

1.2.11 Define: MAX_GUID_STRING_LEN

Category:
Type:
Define: MAX_GUID_STRING_LEN
Key: 38
Maximum legth of the GUID string

1.2.12 Define: RTS_STRING_STATIC

Category: ComponentNameFlags
Type:
Define: RTS_STRING_STATIC
Key: 0x00000001

- RTS_STRING_STATIC: Name is stored in a static buffer
- RTS_STRING_DYNAMIC: Name is stored in a dynamic buffer
- RTS_STRING_ASCII: Ascii string content

1.2.13 Define: CM_STRING_GET_CONTENT

Condition: #ifndef default
Category:
Type:
Define: CM_STRING_GET_CONTENT
Key: pString
Macro to get the string content out of a string class.

1.2.14 Define: CM_STRING_GET_SIZE

Category:
Type:
Define: CM_STRING_GET_SIZE
Key: pString
Macro to get the size in bytes of the content of a string class.

1.2.15 CMUtlSkipWhiteSpace

char CMUtlSkipWhiteSpace (char* psz)*
Called to skip withspaces out of a string.

psz [IN]
Pointer to the string

Result
returns the pointer to the next position in the string with no whitespace

1.2.16 CMUtlLtoa

char CMUtlLtoa (RTS_I32 lValue, char *pszString, int nMaxLen, int iBase, RTS_RESULT *pResult)*

Converts a long value to a string. The representation can be specified by the iBase value:

2: binary (e.g. "10101010101110111100110011011101"). String length must be at least MAX_LONG_BIN_STRING_LEN bytes!

10: decimal (e.g. "2864434397"). String length must be at least MAX_LONG_DEC_STRING_LEN bytes!

16: hexadecimal (e.g. "0xAABBCCDD"). String length must be at least MAX_LONG_HEX_STRING_LEN bytes!

lValue [IN]
Long value to convert

szString [OUT]
Pointer to the string to get integer value as string

iBase [IN]
Base for the conversion: See description

Result
returns the pointer to the next position in the string with no whitespace

1.2.17 CMUtlUltoa

char CMUtlUltoa (RTS_UI32 ulValue, char *pszString, int nMaxLen, int iBase, RTS_RESULT *pResult)*

Converts an unsigned long value to a string. The representation can be specified by the iBase value:

2: binary (e.g. "10101010101110111100110011011101"). String length must be at least MAX_ULONG_BIN_STRING_LEN bytes!

10: decimal (e.g. "2864434397"). String length must be at least MAX_ULONG_DEC_STRING_LEN bytes!

16: hexadecimal (e.g. "0xAABBCCDD"). String length must be at least MAX_ULONG_HEX_STRING_LEN bytes!

ulValue [IN]
Unsigned long value to convert

szString [OUT]
Pointer to the string to get integer value as string

iBase [IN]
Base for the conversion: See description

Result
returns the pointer to the next position in the string with no whitespace

1.2.18 CMUtlStrrev

char CMUtlStrrev (char *psz)*

Reverses content of a string in place

psz [INOUT]
Pointer to a null-terminated string

Result
returns the pointer to the next the same string

1.2.19 CMUtlStrRChr

char CMUtlStrRChr (char *pszString, char *pszBegin, char cFind)*

Scan a string for the last occurrence of a character

pszString [IN]
Pointer to scan

pszBegin [IN]
Pointer of a position into pszString to start the scan

Result
Pointer to the first occurrence in string, or NULL if not found

1.2.20 CMUtlStrICmp

*int CMUtlStrICmp (char *pszString1, char *pszString2)*

Case insensitive string compare.

pszString1 [IN]
Pointer to first string to compare

pszString2 [IN]
Pointer to second string to compare

Result
0 if strings are equal, -1 or 1 if there is a difference between them

1.2.21 CMUtlStrNICmp

*int CMUtlStrNICmp (char *pszString1, char *pszString2, RTS_SIZE n)*

Lower case comparison of two strings specified by length.

pszString1 [IN]
Pointer to first string to compare

pszString2 [IN]
Pointer to second string to compare

n [IN]
Number of characters to compare

Result
0= strings are equal, !0=strings unequal

1.2.22 CMUtlStrIStr

*char * CMUtlStrIStr (const char *pszString, const char *pszStringToFind)*

Find a substring in another string.

pszString [IN]
String to search substring in

pszStringToFind [IN]
Substring to find

Result
NULL= substring not found in string, !NULL= pointer to string, where substring was found

1.2.23 CMUtlStrlwr

char CMUtlStrlwr (char *pszString)*

Lower case conversion of a string.

pszString [IN]
Pointer to conversion string

Result
Returns a pointer to the converted string, NULL if conversion failed

1.2.24 CMUtlStrupr

char CMUtlStrupr (char *pszString)*

Upper case conversion of a string.

pszString [IN]
Pointer to conversion string

Result
Returns a pointer to the converted string, NULL if conversion failed

1.2.25 CMUtlSafeStrCpy

*RTS_RESULT CMUtlSafeStrCpy (char *pszDest, RTS_SIZE nDestSize, const char *pszSrc)*

Safe string copy of a string. The length of the destination buffer is checked to avoid memory overwrites.

pszDest [IN]
Pointer to destination buffer

nDestSize [IN]
Length of destination buffer

pszSrc [IN]
Pointer to source buffer. String must be NUL terminated!

Result
error code

1.2.26 CMUtlSafeStrCpy2

*RTS_RESULT CMUtlSafeStrCpy2 (char *pszDest, RTS_SIZE nDestSize, const char *pszSrc, int nCopyMode)*

Copy a string from one buffer to another, but limit the size of the destination buffer to nDestSize.

pszDest [IN]
Pointer to destination buffer

nDestSize [IN]
Length of destination buffer

pszSrc [IN]
Pointer to source buffer. String must be NUL terminated!

nCopyMode [IN]
Copy mode. See corresponding category "Copy mode"

Result
error code

1.2.27 CMUtlSafeStrNCpy

*RTS_RESULT CMUtlSafeStrNCpy (char *pszDest, RTS_SIZE nDestSize, const char *pszSrc, RTS_SIZE nBytesCopy)*

Copy a string from one buffer to another specified by the number of characters to copy, but limit the size of the destination buffer to nDestSize.

pszDest [IN]
Pointer to destination buffer

nDestSize [IN]
Length of destination buffer

pszSrc [IN]
Pointer to source buffer. String must be NUL terminated!

nBytesCopy [IN]
Number of bytes to copy

Result
error code

1.2.28 CMUtlSafeStrCat

*RTS_RESULT CMUtlSafeStrCat (char *pszDest, RTS_SIZE nDestSize, const char *pszSrc)*

Concatenate two strings, but limit the size of the destination buffer to nDestSize.

pszDest [IN]
Pointer to destination buffer

nDestSize [IN]
Length of destination buffer

pszSrc [IN]
Pointer to source buffer. String must be NUL terminated!

Result

error code

1.2.29 CMUtlStrLen

*RTS_SIZE CMUtlStrLen (char *str)*

Calculate the string length

Is able to handle Null pointer

This function works for targets where char has a size of 8 bits as well as for targets where char is 16 bits wide.

str [IN]
pointer to string

Result
Length of string without trailing \0, 0 if str is Null

1.2.30 CMUtlStrToLower

char CMUtlStrToLower (char* psz)*

Convert a string in lower case. String must be NUL terminated!

psz [IN]
Pointer to string to convert

Result
Pointer to psz with the converted content

1.2.31 CMUtlStrToUpper

char CMUtlStrToUpper (char* psz)*

Convert a string in upper case. String must be NUL terminated!

psz [IN]
Pointer to string to convert

Result
Pointer to psz with the converted content

1.2.32 CMUtlvsnprintf

*RTS_RESULT CMUtlvsnprintf (char *pszStr, RTS_SIZE ulStrSize, const char *pszFormat, va_list *pargList)*

Safe sprintf with argument list. NOTES: - If ANSI vsnprintf is not available on the target, vsprintf is used instead and the size of the string can be evaluated only after writing the complete string! So the memory could be overwritten in this case and so the error code is the only way to detect this situation! - RTS_ASSERT() is generated, if string buffer is too short!

pszStr [IN]
Destination buffer

ulStrSize [IN]
Destination buffer size

pszFormat [IN]
Format string

pargList [IN]
Pointer to argument list

Result
Error code:
- ERR_OK: String could be written
- ERR_PARAMETER: NULL pointer used as parameter
- ERR_BUFFERSIZE: If pszStr is too short to get the string value

1.2.33 CMUtlsnprintf

*RTS_RESULT CMUtlsnprintf (char *pszStr, RTS_SIZE ulStrSize, const char *pszFormat, ...)*

Safe sprintf with format string and variable argument list. NOTES: - If ANSI vsnprintf is not available on the target, vsprintf is used instead and the size of the string can be evaluated only after writing the complete string! So the memory could be overwritten in this case and so the error code is the only way to detect this situation! - RTS_ASSERT() is generated, if string buffer is too short!

pszStr [IN]
Destination buffer

ulStrSize [IN]
Destination buffer size
pszFormat [IN]
Format string
... [IN]
Argument list
Result
Error code:
- ERR_OK: String could be written
- ERR_PARAMETER: NULL pointer used as parameter
- ERR_BUFFERSIZE: If pszStr is too short to get the string value

1.2.34 CMUtlSafeMemCpy

*RTS_RESULT CMUtlSafeMemCpy (void *pDest, RTS_SIZE ulDestSize, const void *pSrc, RTS_SIZE ulToCopy)*
Safe memory copy between two buffers. The length of the destination buffer is checked to avoid memory overwrites.
pDest [IN]
Pointer to destination buffer
ulDestSize [IN]
Length of destination buffer
pSrc [IN]
Pointer to source buffer
ulToCopy [IN]
Bytes to copy from source to destination buffer
Result
error code

1.2.35 CMUtlGUIDToString

*RTS_RESULT CMUtlGUIDToString (RTS_GUID *pguid, char *pszGUID, RTS_SIZE nMaxLen)*
Conversion of a GUID into a string.
pguid [IN]
Pointer to the GUID to convert
pszGUID [OUT]
Pointer to string buffer for the result
nMaxLen [IN]
maximum length of the string buffer. Should be at least MAX_GUID_STRING_LEN
Result
error code

1.2.36 CMStringCreate

*RTS_RESULT CMStringCreate (RTS_STRING_CLASS *pString, char *pszComponentName, char *pszInit)*
Create a string and initialize it with a default string.
pString [IN]
Pointer to string class
pszComponentName [IN]
Pointer to the component name, which creates this string
pszInit [IN]
Pointer to init string to copy into content. Can be NULL.
Result
error code

1.2.37 CMStringDelete

*RTS_RESULT CMStringDelete (RTS_STRING_CLASS *pString, char *pszComponentName)*
Create a string and initialize it with a default string.
pString [IN]
Pointer to string class

pszComponentName [IN]
Pointer to the component name, which creates this string
Result
error code

1.2.38 CMStringExtend

*RTS_RESULT CMStringExtend (RTS_STRING_CLASS *pString, char *pszComponentName, RTS_SIZE size)*
Extend a string to a specified size.
pString [IN]
Pointer to string class
pszComponentName [IN]
Pointer to the component name, which creates this string
size [IN]
Size to which the content must be extended
Result
error code

1.2.39 CMUtlwstrcmp

*int CMUtlwstrcmp (const RTS_WCHAR *pwsz1, const RTS_WCHAR *pwsz2)*
Compare two unicode strings
pwsz1 [IN]
Pointer to string 1
pwsz2 [IN]
Pointer to string 2
Result
0: string1 identical to string2 lower 0: string1 less than string2 greater 0: string1 greater than string2

1.2.40 CMUtlwstrncmp

*int CMUtlwstrncmp (const RTS_WCHAR *pwsz1, const RTS_WCHAR *pwsz2, RTS_SIZE nCount)*
Compare two unicode strings with specified number of characters
pwsz1 [IN]
Pointer to string 1
pwsz2 [IN]
Pointer to string 2
nCount [IN]
Number of characters to compare
Result
0: string1 identical to string2 lower 0: string1 less than string2 greater 0: string1 greater than string2

1.2.41 CMUtlwstrcpy

*RTS_RESULT CMUtlwstrcpy (RTS_WCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_WCHAR *pwszSrc)*
Copy one unicode string to another in a safe way
pwszDest [IN]
Destination string
nDestSize [IN]
Size of the destination buffer in unicode characters (not bytes)!
pwszSrc [IN]
Source string
Result
error code

1.2.42 CMUtlwstrcat

*RTS_RESULT CMUtlwstrcat (RTS_WCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_WCHAR *pwszSrc)*
Concatenate two unicode strings in a safe way
pwszDest [IN]

Destination string

nDestSize [IN]
Size of the destination buffer in unicode characters (not bytes)!

pwszSrc [IN]
Source string

Result
error code

1.2.43 CMUtlwstrncpy

*RTS_RESULT CMUtlwstrncpy (RTS_WCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_WCHAR *pwszSrc, RTS_SIZE n)*

Copy number of characters from one unicode string to another in a safe way

pwszDest [IN]
Destination string

nDestSize [IN]
Size of the destination buffer in unicode characters (not bytes)!

pwszSrc [IN]
Source string

n [IN]
Number of characters to copy (not bytes)!

Result
error code

1.2.44 CMUtlwstrlen

*RTS_SIZE CMUtlwstrlen (const RTS_WCHAR *pwsz)*

Get the number of characters (not bytes!) in the content of a string _without_ NUL terminating character.

pwsz [IN]
Pointer to string

Result
Number of characters (not bytes!) in the content of a string _without_ NUL terminating character

1.2.45 CMUtlwtolower

RTS_WCHAR CMUtlwtolower (RTS_WCHAR* pwsz)*

Convert the characters in a unicode string to lower case.

pwsz [IN]
Pointer to string

Result
pwsz is returned

1.2.46 CMUtlwtoupper

RTS_WCHAR CMUtlwtoupper (RTS_WCHAR* pwsz)*

Convert the characters in a unicode string to upper case.

pwsz [IN]
Pointer to string

Result
pwsz is returned

1.2.47 CMUtlStrToW

RTS_RESULT CMUtlStrToW (const char pszSrc, RTS_WCHAR* pwszDest, RTS_SIZE nDestSize)*

Convert a byte character string to a unicode character string.

pszSrc [IN]
Pointer to byte character string

pwszDest [IN]
Pointer to unicode character string

nDestSize [IN]
Size of the destination buffer in unicode characters (not bytes)!

Result

error code

1.2.48 CMUtlWToStr

RTS_RESULT CMUtlWToStr (const RTS_WCHAR pwszSrc, char* pszDest, RTS_SIZE nDestSize)*
Convert a unicode character string to a byte character string .

pwszSrc [IN]
Pointer to unicode character string

pszDest [IN]
Pointer to byte character string

nDestSize [IN]
Size of the destination buffer in bytes

Result
error code

1.2.49 CMUtlwstrchr

RTS_WCHAR CMUtlwstrchr (const RTS_WCHAR *pwsz, RTS_WCHAR w)*
Find a character in a unicode string.

pwsz [IN]
Pointer to unicode string

w [IN]
Character to find

Result
Returns the find position or NULL if not found

1.2.50 CMUtlcwstrcmp

*int CMUtlcwstrcmp (const RTS_CWCHAR *pwsz1, const RTS_CWCHAR *pwsz2)*
Compare two RTS_CWCHAR strings

pwsz1 [IN]
Pointer to string 1

pwsz2 [IN]
Pointer to string 2

Result
0: string1 identical to string2 lower 0: string1 less than string2 greater 0: string1 greater than string2

1.2.51 CMUtlcwstrcpy

*RTS_RESULT CMUtlcwstrcpy (RTS_CWCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_CWCHAR *pwszSrc)*
Copy one RTS_CWCHAR string to another in a safe way

pwszDest [IN]
Destination string

nDestSize [IN]
Size of the destination buffer in RTS_CWCHAR characters!

pwszSrc [IN]
Source string

Result
error code

1.2.52 CMUtlcwstrcat

*RTS_RESULT CMUtlcwstrcat (RTS_CWCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_CWCHAR *pwszSrc)*
Concatenate two RTS_CWCHAR strings in a safe way

pwszDest [IN]
Destination string

nDestSize [IN]
Size of the destination buffer in RTS_CWCHAR characters!

pwszSrc [IN]
Source string

Result

error code

1.2.53 CMUtlcwstrncpy

*RTS_RESULT CMUtlcwstrncpy (RTS_CWCHAR *pwszDest, RTS_SIZE nDestSize, const RTS_CWCHAR *pwszSrc, RTS_SIZE n)*

Copy number of characters from one RTS_CWCHAR string to another in a safe way

pwszDest [IN]
Destination string

nDestSize [IN]
Size of the destination buffer in RTS_CWCHAR characters!

pwszSrc [IN]
Source string

n [IN]
Number of characters to copy (not bytes)!

Result
error code

1.2.54 CMUtlcwstrlen

*RTS_SIZE CMUtlcwstrlen (const RTS_CWCHAR *pwsz)*

Get the number of RTS_CWCHAR characters in the content of a string _without_ NUL terminating character.

pwsz [IN]
Pointer to string

Result
Number of RTS_CWCHAR characters in the content of a string _without_ NUL terminating character

1.2.55 CMUtlcwtolower

RTS_CWCHAR CMUtlcwtolower (RTS_CWCHAR* pwsz)*

Convert the characters in a RTS_CWCHAR string to lower case.

pwsz [IN]
Pointer to string

Result
pwsz is returned

1.2.56 CMUtlcwtoupper

RTS_CWCHAR CMUtlcwtoupper (RTS_CWCHAR* pwsz)*

Convert the characters in a RTS_CWCHAR string to upper case.

pwsz [IN]
Pointer to string

Result
pwsz is returned

1.2.57 CMUtlStrToCW

RTS_RESULT CMUtlStrToCW (const char pszSrc, RTS_CWCHAR* pwszDest, RTS_SIZE nDestSize)*

Convert a byte character string to a RTS_CWCHAR character string.

pszSrc [IN]
Pointer to byte character string

pwszDest [IN]
Pointer to RTS_CWCHAR character string

nDestSize [IN]
Size of the destination buffer in RTS_CWCHAR characters!

Result
error code

1.2.58 CMUtlCWToStr

RTS_RESULT CMUtlCWToStr (const RTS_CWCHAR pwszSrc, char* pszDest, RTS_SIZE nDestSize)*

Convert a RTS_CWCHAR character string to a byte character string .

pwszSrc [IN]
Pointer to RTS_CWCHAR character string

pszDest [IN]
Pointer to byte character string

nDestSize [IN]
Size of the destination buffer in bytes

Result
error code

1.2.59 CMUtlcwstrchr

RTS_CWCHAR CMUtlcwstrchr (const RTS_CWCHAR *pwsz, RTS_CWCHAR cw)*

Find a character in a RTS_CWCHAR string.

pwsz [IN]
Pointer to RTS_CWCHAR string

w [IN]
Character to find

Result
Returns the find position or NULL if not found

1.3 CMBasicChecksItf

Interface to check the runtime environment. Here things like datatype size and byte order are checked, before the runtime system was started.

1.3.1 CMBasicChecks

int CMBasicChecks (void)

Checks to environment and the compiled runtime system for consistency. - Checks the real size of all data types - Checks, if swapping must be done and if swapping is done correctly

Result
TRUE if successful, FALSE if error occurred

1.4 CMLockItf

Interface for a generic locking mechanism

If both SysSem and SysInt components are implemented in the runtime system, a global int-lock implementation can be choosen with PREFER_SYSINT macro. Otherwise, a mutex-based implementation is automatically selected.

If either SysSem or SysInt component is implemented only, then the CMLock implementation is based on an existing component.

1.4.1 CMLockCreate

*RTS_HANDLE CMLockCreate (RTS_RESULT *pResult)*

Function to create a lock object to synchronize data against concurrent task or interrupt access.

pResult [OUT]
Pointer to error code

Result
Handle to the lock object

1.4.2 CMLockDelete

RTS_RESULT CMLockDelete (RTS_HANDLE hLock)

Function to delete a lock object.

hLock [IN]
Handle to the lock object retrieved by CMLockCreate()

Result
Error code

1.4.3 CMLockEnter

RTS_RESULT CMLockEnter (RTS_HANDLE hLock)

Function to enter a lock object. After entering, the accessed data is _safe_ against concurrent access.

hLock [IN]

Handle to the lock object retrieved by CMLockCreate()

Result

Error code

1.4.4 CMLockLeave

RTS_RESULT CMLockLeave (RTS_HANDLE hLock)

Function to leave a lock object. After leaving, the accessed data is _unsafe_ against concurrent access.

hLock [IN]

Handle to the lock object retrieved by CMLockCreate()

Result

Error code

2 CmpAppEmbedded

This component is responsible for handling one single IEC Application. It has reduced memory requirements compared to the CmpApp. It also uses a different application download format to be able to run the application directly from flash if required. This implementation of the application manager needs two components to handle the IEC tasks: CmpIecTask: All IEC tasks of all applications are registered at this component. The IEC cycle is implemented there. CmpSchedule: The IEC task component registered all tasks at the scheduler component for the timing and scheduling of all tasks. Monitoring is done in a separate component too (CmpMonitor).

Compiler Switch

- #define APP_POU_TABLES_ENABLED Management of information for each POU can be enabled with this switch. This can be used in CoDeSys to check during login,

2.1 CmpAppItf

This is the interface of the IEC application manager. The manager is responsible for:
- Handles and collects all IEC applications
- Application specific online communication to CoDeSys (debugging, forcing, etc.)
- Code and data handling (areas)
- Handling of non volatile data (retain handling)

An IEC application is a set of objects which are needed for running a particular instance of the PLC program. Each application is specified by its unique name.

Applications can have relationships among each other. If one application e.g. A2 depends from another application A1, A1 is the parent and A2 the child. So A2 is derived from A1.

Each application has two different kinde of states:
- Application state: run, stop, exception, etc.
- Operating state: storing bootproject, do online-change, etc.

The application state is used to specify, if the application is opertaing normal or not. The operating state is used to specify, which job is executed in the moment on the application.

An IEC application consists of one or more tasks. The IEC tasks are not handled directly by the application manager. This is provided by the the IEC task manager. See CmpIecTask for detailed information.

An application can have code and data. So the application manager needs for each application one or more memory areas in which the code and data will be administrated.

2.1.1 Define: MAX_LEN_APPLNAME

Category: Static defines
Type:
Define: MAX_LEN_APPLNAME
Key: 60
Maximum length fo application name

2.1.2 Define: MAX_PATH_LEN

Condition: #ifndef MAX_PATH_LEN
Category: Static defines
Type:
Define: MAX_PATH_LEN
Key: 255
Maximum length of a file path

2.1.3 Define: APPL_NUM_OF_STATIC_APPLS

Condition: #ifndef APPL_NUM_OF_STATIC_APPLS
Category: Static defines
Type:
Define: APPL_NUM_OF_STATIC_APPLS
Key: 8
Length of application list that is allocated static

2.1.4 Define: APPL_NUM_OF_STATIC_SESSIONIDS

Condition: #ifndef APPL_NUM_OF_STATIC_SESSIONIDS
Category: Static defines
Type:
Define: APPL_NUM_OF_STATIC_SESSIONIDS
Key: 5
Length of session id list that is allocated static

2.1.5 Define: APPL_NUM_OF_STATIC_AREAS

Condition: #ifndef APPL_NUM_OF_STATIC_AREAS
Category: Static defines
Type:
Define: APPL_NUM_OF_STATIC_AREAS
Key: 10
Length of area pointer list that is allocated static in the application object

2.1.6 Define: APP_NUM_OF_STATIC_ASYNC_SERVICES

Condition: #ifndef APP_NUM_OF_STATIC_ASYNC_SERVICES
Category: Static defines
Type:
Define: APP_NUM_OF_STATIC_ASYNC_SERVICES
Key: 8
Number of possible static async services. Can be increased dynamically.

2.1.7 Define: PROJECT_ARCHIVE

Category: File name definitions
Type:
Define: PROJECT_ARCHIVE
Key: Archive.prj
Name of the project archive with all applications in the project

2.1.8 Define: PROJECT_ARCHIVE_INFO

Category: File name definitions
Type:
Define: PROJECT_ARCHIVE_INFO
Key: Archive.inf
Name of the project archive info file, which contains all informations about the archive content

2.1.9 Define: APP_BOOTPROJECT_FILE_EXTENSION

Condition: #ifndef APP_BOOTPROJECT_FILE_EXTENSION
Category: Static defines
Type:
Define: APP_BOOTPROJECT_FILE_EXTENSION
Key: .app
Bootproject file extension

2.1.10 Define: APP_BOOTPROJECT_FILE_EXTENSION_INVALID

Condition: #ifndef APP_BOOTPROJECT_FILE_EXTENSION_INVALID
Category: Static defines
Type:
Define: APP_BOOTPROJECT_FILE_EXTENSION_INVALID
Key: .ap_
Bootproject file extension to invalidate during download

2.1.11 Define: APP_BOOTPROJECT_FILE_EXTENSION_ERROR

Condition: #ifndef APP_BOOTPROJECT_FILE_EXTENSION_ERROR
Category: Static defines
Type:
Define: APP_BOOTPROJECT_FILE_EXTENSION_ERROR
Key: .err
Bootproject file extension if file has an error

2.1.12 Define: APP_BOOTPROJECT_FILE_EXTENSION_CRC

Condition: #ifndef APP_BOOTPROJECT_FILE_EXTENSION_CRC
Category: Static defines
Type:

Define: APP_BOOTPROJECT_FILE_EXTENSION_CRC
Key: .crc
File extension for bootproject crc checksum file

2.1.13 Define: APP_FILE_EXTENSION_SYMBOLS

Condition: #ifndef APP_FILE_EXTENSION_SYMBOLS
Category: Static defines
Type:
Define: APP_FILE_EXTENSION_SYMBOLS
Key: .xml
File extension for symbol file matching to application

2.1.14 Define: APP_BOOTPROJECT_FILE_EXTENSION_SYMBOLS

Condition: #ifndef APP_BOOTPROJECT_FILE_EXTENSION_SYMBOLS
Category: Static defines
Type:
Define: APP_BOOTPROJECT_FILE_EXTENSION_SYMBOLS
Key: .boot.xml
File extension for symbol file matching to bootproject

2.1.15 Define: CCO_SEMAPHORE

Category: Static defines
Type:
Define: CCO_SEMAPHORE
Key: 1
Copy code option for online-change. CCO_DEFAULT defines the option, which mechanism is used to synchronize copy code.

2.1.16 Define: OP_APP_STOP

Category: Operations
Type:
Define: OP_APP_STOP
Key: 1
Operations of the application component. Can be disabled with the event CMPID_CmpMgr::EVT_CmpMgr_DisableOperation!

2.1.17 Define: EVT_PrepareStart

Category: Events
Type:
Define: EVT_PrepareStart
Key: MAKE_EVENTID
Event is sent before start of the specified application

2.1.18 Define: EVT_StartDone

Category: Events
Type:
Define: EVT_StartDone
Key: MAKE_EVENTID
Event is sent after start of the specified application

2.1.19 Define: EVT_PrepareStop

Category: Events
Type:
Define: EVT_PrepareStop
Key: MAKE_EVENTID
Event is sent before stop of the specified application

2.1.20 Define: EVT_StopDone

Category: Events
Type:
Define: EVT_StopDone
Key: MAKE_EVENTID
Event is sent after stop of the specified application

2.1.21 Define: EVT_PrepareReset

Category: Events

Type:
Define: EVT_PrepareReset
Key: MAKE_EVENTID
Event is sent before reset of the specified application, but after the stop of the application! So no IEC user code is executed at this event!

2.1.22 Define: EVT_ResetDone

Category: Events
Type:
Define: EVT_ResetDone
Key: MAKE_EVENTID
Event is sent after reset of the specified application

2.1.23 Define: EVT_PrepareOnlineChange

Category: Events
Type:
Define: EVT_PrepareOnlineChange
Key: MAKE_EVENTID
Event is sent before online change of the specified application

2.1.24 Define: EVT_OnlineChangeDone

Category: Events
Type:
Define: EVT_OnlineChangeDone
Key: MAKE_EVENTID
Event is sent after online change of the specified application

2.1.25 Define: EVT_PrepareDownload

Category: Events
Type:
Define: EVT_PrepareDownload
Key: MAKE_EVENTID
Event is sent before download of the specified application

2.1.26 Define: EVT_DownloadDone

Category: Events
Type:
Define: EVT_DownloadDone
Key: MAKE_EVENTID
Event is sent after download of the specified application

2.1.27 Define: EVT_CodeInitDone

Category: Events
Type:
Define: EVT_CodeInitDone
Key: MAKE_EVENTID
Event is sent after CodeInit. Is called inside the task safe section and only at online-change! (e.g. the copy code for online-change is executed here).

2.1.28 Define: EVT_PrepareDelete

Category: Events
Type:
Define: EVT_PrepareDelete
Key: MAKE_EVENTID
Event is sent before an application is deleted. The application is stopped, if this event is posted.

2.1.29 Define: EVT_DeleteDone

Category: Events
Type:
Define: EVT_DeleteDone
Key: MAKE_EVENTID
Event is sent after an application is deleted. The application is stopped, if this event is posted.
ATTENTION: Right after this event, the APPLICATION structure of the event parameter is deleted!

2.1.30 Define: EVT_PrepareExit

Category: Events

Type:
Define: EVT_PrepareExit
Key: MAKE_EVENTID
Event is sent before an application executes its exit code (deleting the application, reinit at download or reset). The application is in stop, if this event is posted.

2.1.31 Define: EVT_ExitDone

Category: Events
Type:
Define: EVT_ExitDone
Key: MAKE_EVENTID
Event is sent after an application executes its exit code (deleting the application, reinit at download or reset). The application is in stop, if this event is posted.

2.1.32 Define: EVT_CreateBootprojectDone

Category: Events
Type:
Define: EVT_CreateBootprojectDone
Key: MAKE_EVENTID
Event is sent after creating a bootproject of an application successfully

2.1.33 Define: EVT_DenyLoadBootproject

Category: Events
Type:
Define: EVT_DenyLoadBootproject
Key: MAKE_EVENTID
Event is sent to deny loading a bootproject of an application.

2.1.34 Define: EVT_PrepareLoadBootproject

Category: Events
Type:
Define: EVT_PrepareLoadBootproject
Key: MAKE_EVENTID
Event is sent before loading a bootproject of an application

2.1.35 Define: EVT_LoadBootprojectDone

Category: Events
Type:
Define: EVT_LoadBootprojectDone
Key: MAKE_EVENTID
Event is sent after loading a bootproject of an application successfully

2.1.36 Define: EVT_DenyStartBootproject

Category: Events
Type:
Define: EVT_DenyStartBootproject
Key: MAKE_EVENTID
Event is sent to deny starting a bootproject of an application.

2.1.37 Define: EVT_PrepareStartBootproject

Category: Events
Type:
Define: EVT_PrepareStartBootproject
Key: MAKE_EVENTID
Event is sent before starting a bootproject of an application

2.1.38 Define: EVT_StartBootprojectDone

Category: Events
Type:
Define: EVT_StartBootprojectDone
Key: MAKE_EVENTID
Event is sent after starting a bootproject of an application

2.1.39 Define: EVT_DenyStart

Category: Events
Type:

Define: EVT_DenyStart
Key: MAKE_EVENTID
Event is sent to deny starting an application

2.1.40 Define: EVT_DenyStop

Category: Events
Type:
Define: EVT_DenyStop
Key: MAKE_EVENTID
Event is sent to deny stopping an application

2.1.41 Define: EVT_AllBootprojectsLoaded

Category: Events
Type:
Define: EVT_AllBootprojectsLoaded
Key: MAKE_EVENTID
Event is sent after all boot applications have been loaded

2.1.42 Define: EVT_GlobalExitOnResetDone

Category: Events
Type:
Define: EVT_GlobalExitOnResetDone
Key: MAKE_EVENTID
Event is sent on Reset, after global exit and before global init.

2.1.43 Define: EVT_ExitDoneWithConfigAppInfo

Category: Events
Type:
Define: EVT_ExitDoneWithConfigAppInfo
Key: MAKE_EVENTID
Event is sent after an application has executed its exit code. The application is stopped, if this event is posted. The event has got the additional parameter szConfigApp

2.1.44 Define: EVT_CmpApp_Exception

Category: Events
Type:
Define: EVT_CmpApp_Exception
Key: MAKE_EVENTID
Event is sent, if an exception occurred in the context of an application NOTE: In case of a retain mismatch, the RTSEXCPT_RETAIN_IDENTITY_MISMATCH is provided as exception code (see ulException in EVTPARAM_CmpAppException).

2.1.45 Define: EVT_RegisterBootproject

Category: Events
Type:
Define: EVT_RegisterBootproject
Key: MAKE_EVENTID
Event is sent, if a new bootproject is registered

2.1.46 Define: EVT_CreateBootprojectFileFailed

Category: Events
Type:
Define: EVT_CreateBootprojectFileFailed
Key: MAKE_EVENTID
Event is sent, if the bootproject file cannot be created. NOTE: EVT_CreateBootprojectFailed is sent additionally in this case!

2.1.47 Define: EVT_CreateBootprojectFailed

Category: Events
Type:
Define: EVT_CreateBootprojectFailed
Key: MAKE_EVENTID
Event is sent, if the creation of a bootproject failed. This can occur at creation of the bootproject implicit at download or explicit by the user.

2.1.48 Define: EVT_OperatingStateChanged

Category: Events
Type:
Define: EVT_OperatingStateChanged
Key: MAKE_EVENTID
Event is sent, if the operating state has changed

2.1.49 Define: EVT_SourceDownload

Category: Events
Type:
Define: EVT_SourceDownload
Key: MAKE_EVENTID
Event is sent, if the project archive is downloaded (source download)

2.1.50 Define: EVT_Login

Category: Events
Type:
Define: EVT_Login
Key: MAKE_EVENTID
Event is sent, if a client login to the specified application

2.1.51 Define: EVT_Logout

Category: Events
Type:
Define: EVT_Logout
Key: MAKE_EVENTID
Event is sent, if a client logout of the specified application. The event is sent additionally, if a communication error occurred.

2.1.52 Define: EVT_DenyDelete

Category: Events
Type:
Define: EVT_DenyDelete
Key: MAKE_EVENTID
Event is sent to deny deleting an application

2.1.53 Define: EVT_DenyDeleteBootproject

Category: Events
Type:
Define: EVT_DenyDeleteBootproject
Key: MAKE_EVENTID
Event is sent to deny deleting a bootproject of an application

2.1.54 Define: EVT_OEMDownloadServiceTag

Category: Events
Type:
Define: EVT_OEMDownloadServiceTag
Key: MAKE_EVENTID
Event is sent to handle own download/online-change service tags

2.1.55 Define: EVT_OEMRegisteredIecFunction

Category: Events
Type:
Define: EVT_OEMRegisteredIecFunction
Key: MAKE_EVENTID
Event is sent to get all IEC-functions specified in CoDeSys with the following attribute: {attribute 'register_in_runtime'} NOTE: - Only IEC-functions can be registered - The function name must be unique! - The user have to handle the events EVT_PrepareExit and EVT_DeleteDone to check, if the application and so the IEC function will be removed!

2.1.56 Define: EVT_POUTable_Changed

Category: Events
Type:
Define: EVT_POUTable_Changed
Key: MAKE_EVENTID
Event is fired on any change in the POU Tables Manager

2.1.57 Define: EVT_CommCycle

Category: Events
Type:
Define: EVT_CommCycle
Key: MAKE_EVENTID
Event is fired at every communication cycle (idle loop). This can be used in IEC-for background jobs.

2.1.58 Define: EVT_StateChanged

Category: Events
Type:
Define: EVT_StateChanged
Key: MAKE_EVENTID
Event is fired, if the application state changed.

2.1.59 Define: AS_NONE

Category: Application state
Type:
Define: AS_NONE
Key: UINT32_C

- AS_NONE: Unspecified state
- AS_RUN: Application in run
- AS_STOP: Application in stop
- AS_DEBUG_HALT_ON_BP: Application halted on breakpoint
- AS_DEBUG_STEP: Not used actually
- AS_SYSTEM_APPLICATION: State of a system application

2.1.60 Define: OS_NONE

Category: Application operating state
Type:
Define: OS_NONE
Key: UINT32_C

- OS_NONE: Unspecified state (init state)
- OS_PROGRAM_LOADED: Application is completely loaded
- OS_DOWNLOAD: Application download in progress
- OS_ONLINE_CHANGE: Application online-change in progress
- OS_STORE_BOOTPROJECT: Store bootproject in progress
- OS_FORCE_ACTIVE: Force values is active on the application
- OS_EXCEPTION: Application is in exception state (an exception occurred in this application)
- OS_RUN_AFTER_DOWNLOAD: Download code at the end of download is in progress (initialization of the application)
- OS_STORE_BOOTPROJECT_ONLY: Only the bootproject is stored at download
- OS_EXIT: Application exit is still executed (application is no longer active)
- OS_DELETE: Application is deleted (object is available, but the content is stil deleted)
- OS_RESET: Application reset is in progress
- OS_RETAIN_MISMATCH: Retain mismatch occurred during loading the bootproject (retain data does not match to the application)
- OS_BOOTPROJECT_VALID: Bootproject available (bootproject matched to running application in RAM)
- OS_LOAD_BOOTPROJECT: Loading bootproject in progress
- OS_FLOW_ACTIVE: Flow control active
- OS_RUN_IN_FLASH: Application is running in flash

2.1.61 Define: APP_STOP_REASON_UNKNOWN

Category: Stop reason
Type:
Define: APP_STOP_REASON_UNKNOWN
Key: 0
Reason to set the application in stop.

2.1.62 Define: USERDB_OBJECT_PLCLOGIC

Category: Static defines
Type:
Define: USERDB_OBJECT_PLCLOGIC
Key: Device.PlcLogic
Predefined objects in the runtime

2.1.63 Define: TEXT_PROPERTY_PROJECT

2.1.64 Define: AF_SYSTEM_APPLICATION
Category: Application flags
Type:
Define: AF_SYSTEM_APPLICATION
Key: UINT32_C
Flags to specify properties of an application

2.1.65 Define: DLF_CONTINUEDOWNLOAD
Category: Download flags
Type:
Define: DLF_CONTINUEDOWNLOAD
Key: UINT32_C
Download flags that are transmitted with each download.

2.1.66 Define: RTS_RESET
Category: Reset options
Type:
Define: RTS_RESET
Key: 0
Warm reset. All gobal data except retain data is reset to their default values.

2.1.67 Define: RTS_RESET_COLD
Category: Reset options
Type:
Define: RTS_RESET_COLD
Key: 1
Cold reset. All gobal data AND retain data is reset to their default values.

2.1.68 Define: RTS_RESET_ORIGIN
Category: Reset options
Type:
Define: RTS_RESET_ORIGIN
Key: 2
Origin reset. Delete the application, delete all application files (bootproject, etc.), reset all gobal and retain data. After this command, the controller doesn't know anything about the the application.

2.1.69 Define: SRV_LOGIN_APP
Category: Online services
Type:
Define: SRV_LOGIN_APP
Key: 0x01

2.1.70 Define: TAG_DATA_AREA
Category: Online tags
Type:
Define: TAG_DATA_AREA
Key: 0x01

2.1.71 Define: TAG_REPLY_EXTERNAL_REFERENCE_ERROR_LIST
Category: Online reply tags
Type:
Define: TAG_REPLY_EXTERNAL_REFERENCE_ERROR_LIST
Key: 0x01

2.1.72 Typedef: EVTPARAM_CmpApp
Stuctname: EVTPARAM_CmpApp
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION* pApp; } EVTPARAM_CmpApp;

2.1.73 Typedef: EVTPARAM_CmpApp_Reset
Stuctname: EVTPARAM_CmpApp_Reset
Category: Event parameter

Typedef: typedef struct { struct tagAPPLICATION* pApp; RTS_UI16 usResetOption; } EVTPARAM_CmpApp_Reset;

2.1.74 Typedef: EVTPARAM_CmpAppConfig

Stuctname: EVTPARAM_CmpAppConfig
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION* pApp; char* pszConfigApplication; } EVTPARAM_CmpAppConfig;

2.1.75 Typedef: EVTPARAM_CmpAppStop

Stuctname: EVTPARAM_CmpAppStop
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION* pApp; unsigned long ulStopReason; } EVTPARAM_CmpAppStop;

2.1.76 Typedef: EVTPARAM_CmpAppDenyStart

Stuctname: EVTPARAM_CmpAppDenyStart
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION* pApp; int bDeny; } EVTPARAM_CmpAppDenyStart;

2.1.77 Typedef: EVTPARAM_CmpAppDenyStop

Stuctname: EVTPARAM_CmpAppDenyStop
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION* pApp; unsigned long ulStopReason; int bDeny; } EVTPARAM_CmpAppDenyStop;

2.1.78 Typedef: EVTPARAM_CmpAppAllBootAppsLoaded

Stuctname: EVTPARAM_CmpAppAllBootAppsLoaded
Category: Event parameter
Typedef: typedef struct { int nTotalBootApps; int nSuccessfullyLoadedBootApps; } EVTPARAM_CmpAppAllBootAppsLoaded;

2.1.79 Typedef: EVTPARAM_CmpAppDenyLoadBootproject

Stuctname: EVTPARAM_CmpAppDenyLoadBootproject
Category: Event parameter
Typedef: typedef struct { char *pszAppName; int bDeny; } EVTPARAM_CmpAppDenyLoadBootproject;

2.1.80 Typedef: EVTPARAM_CmpAppPrepareLoadBootproject

Stuctname: EVTPARAM_CmpAppPrepareLoadBootproject
Category: Event parameter
Typedef: typedef struct { char *pszAppName; } EVTPARAM_CmpAppPrepareLoadBootproject;

2.1.81 Typedef: EVTPARAM_CmpAppException

Stuctname: EVTPARAM_CmpAppException
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; RTS_HANDLE hIecTask; RTS_UI32 ulException; } EVTPARAM_CmpAppException;

2.1.82 Typedef: EVTPARAM_CmpAppRegisterBootproject

Stuctname: EVTPARAM_CmpAppRegisterBootproject
Category: Event parameter
Typedef: typedef struct { char *pszAppName; } EVTPARAM_CmpAppRegisterBootproject;

2.1.83 Typedef: EVTPARAM_CmpAppOperatingStateChanged

Stuctname: EVTPARAM_CmpAppOperatingStateChanged
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; RTS_UI32 ulPrevOpState; } EVTPARAM_CmpAppOperatingStateChanged;

2.1.84 Typedef: EVTPARAM_CmpAppSourceDownload

Stuctname: EVTPARAM_CmpAppSourceDownload

Category: Event parameter
Typedef: typedef struct { char *pszArchiveName; char bBegin; } EVTPARAM_CmpAppSourceDownload;

2.1.85 Typedef: EVTPARAM_CmpAppComm

Stuctname: EVTPARAM_CmpAppComm
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; RTS_UI32 ulSessionId; } EVTPARAM_CmpAppComm;

2.1.86 Typedef: EVTPARAM_CmpAppDeny

Stuctname: EVTPARAM_CmpAppDeny
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; int bDeny; } EVTPARAM_CmpAppDeny;

2.1.87 Typedef: EVTPARAM_CmpAppDenyDelete

Stuctname: EVTPARAM_CmpAppDenyDelete
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; int bShutdown; int bDeny; } EVTPARAM_CmpAppDenyDelete;

2.1.88 Typedef: EVTPARAM_CmpApp_OEMServiceTag

Stuctname: EVTPARAM_CmpApp_OEMServiceTag
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; RTS_UI32 ulChannelId; RTS_UI32 ulToplevelTag; HEADER_TAG *pHeaderTag; BINTAGREADER *pReader; BINTAGWRITER *pWriter; RTS_RESULT Result; } EVTPARAM_CmpApp_OEMServiceTag;

2.1.89 Typedef: EVTPARAM_CmpApp_OEMRegisteredIecFunction

Stuctname: EVTPARAM_CmpApp_OEMRegisteredIecFunction
Category: Event parameter
Typedef: typedef struct { struct tagAPPLICATION *pApp; struct T_FUNCTION_INFO *pInfo; char *pszName; } EVTPARAM_CmpApp_OEMRegisteredIecFunction;

2.1.90 Typedef: EVTPARAM_CmpApp_POUTable_Create

Stuctname: EVTPARAM_CmpApp_POUTable_Create
Category: Event parameter
Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; } EVTPARAM_CmpApp_POUTable_Create;

2.1.91 Typedef: EVTPARAM_CmpApp_POUTable_Build

Stuctname: EVTPARAM_CmpApp_POUTable_Build
Category: Event parameter
Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; } EVTPARAM_CmpApp_POUTable_Build;

2.1.92 Typedef: EVTPARAM_CmpApp_POUTable_Clear

Stuctname: EVTPARAM_CmpApp_POUTable_Clear
Category: Event parameter
Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; } EVTPARAM_CmpApp_POUTable_Clear;

2.1.93 Typedef: EVTPARAM_CmpApp_POUTable_Remove

Stuctname: EVTPARAM_CmpApp_POUTable_Remove
Category: Event parameter
Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; } EVTPARAM_CmpApp_POUTable_Remove;

2.1.94 Typedef: EVTPARAM_CmpApp_POUTable_AddElement

Stuctname: EVTPARAM_CmpApp_POUTable_AddElement
Category: Event parameter

Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; struct tagPOU_CALL_ENTRY* pElement; RTS_RESULT Result; } EVTPARAM_CmpApp_POUTable_AddElement;

2.1.95 Typedef: EVTPARAM_CmpApp_POUTable_RemoveElement

Stuctname: EVTPARAM_CmpApp_POUTable_RemoveElement
Category: Event parameter

Typedef: typedef struct { struct tagAREA_CALL_ENTRIES_TABLE* pTable; struct tagPOU_CALL_ENTRY* pElement; } EVTPARAM_CmpApp_POUTable_RemoveElement;

2.1.96 Typedef: EVTPARAM_CmpApp_CommCycle

Stuctname: EVTPARAM_CmpApp_CommCycle
Category: Event parameter

Typedef: typedef struct { RTS_UINTPTR ulParam1; RTS_UINTPTR ulParam2; } EVTPARAM_CmpApp_CommCycle;

2.1.97 Typedef: EVTPARAM_CmpApp_StateChanged

Stuctname: EVTPARAM_CmpApp_StateChanged
Category: Event parameter

Typedef: typedef struct { struct tagAPPLICATION *pApp; RTS_UI32 ulPrevState; } EVTPARAM_CmpApp_StateChanged;

2.1.98 Typedef: POU_DESCRIPTOR

Stuctname: POU_DESCRIPTOR

POU Descriptor.

This type might be extending in future somehow that's why CODE_INFO etc. are not used here.

Typedef: typedef struct tagPOU_DESCRIPTOR { POU offset from the start of the code segment RTS_SIZE offset; POU size RTS_SIZE size; } POU_DESCRIPTOR;

2.1.99 Typedef: POU_CALL_ENTRY

Stuctname: POU_CALL_ENTRY

Descriptor for a callable entity.

Typedef: typedef struct tagPOU_CALL_ENTRY { for intrusive doubly-linked list struct tagPOU_CALL_ENTRY* pNext; struct tagPOU_CALL_ENTRY* pPrev; POU information POU_DESCRIPTOR pou; Optional target specific object that can be attached to this entry. void* pTargetSpecificObject; } POU_CALL_ENTRY;

2.1.100 Typedef: AREA_CALL_ENTRIES_TABLE

Stuctname: AREA_CALL_ENTRIES_TABLE

Table of POU entries.

The life-cycle of this object coincides with the life-cycle of the corresponding code area.

Typedef: typedef struct tagAREA_CALL_ENTRIES_TABLE { Handle to the Table Manager RTS_HANDLE hTablesManager; Link to the next table struct tagAREA_CALL_ENTRIES_TABLE* pNext; Code area start address RTS_UI8* pCode; Code area size RTS_SIZE size; Entries list POU_CALL_ENTRY* pHead; POU_CALL_ENTRY* pTail; Table size for quickening certain operations RTS_SIZE elementsCount; Elements allocator (dynamic memory pool) RTS_HANDLE hElementsPool; Table state flags RTS_SIZE flags; Optional target specific object that can be attached to this table. union { void* pPtrValue; RTS_SIZE value; } targetSpecificObject; } AREA_CALL_ENTRIES_TABLE;

2.1.101 Typedef: PROJECT_INFO

Stuctname: PROJECT_INFO
Category: Project information

Contains the project information as specified in the project information dialog in CoDeSys. To use this, the checkbox "Automatically generate POUs for property access" in the project information dialog be enabled.

Typedef: typedef struct { RTS_IEC_STRING stProjectName[81]; RTS_IEC_STRING stTitle[81]; RTS_IEC_STRING stVersion[81]; RTS_IEC_STRING stAuthor[81]; RTS_IEC_STRING stDescription[81]; } PROJECT_INFO;

2.1.102 Typedef: APPLICATION_INFO

Stuctname: APPLICATION_INFO
Category: Application information
Contains the application information as specified in the application property dialog in CoDeSys.
Typedef: typedef struct { RTS_IEC_STRING *pstProjectName; RTS_IEC_STRING *pstAuthor; RTS_IEC_STRING *pstVersion; RTS_IEC_STRING *pstDescription; RTS_IEC_STRING *pstProfile; RTS_IEC_DATE_AND_TIME dtLastChanges; } APPLICATION_INFO;

2.1.103 Typedef: APP_MEMORY_SEGMENT

Stuctname: APP_MEMORY_SEGMENT
Category: Application memory segment
Describes a memory segment of an application.
Typedef: typedef struct _APP_MEMORY_SEGMENT { RTS_IEC_WORD wType; RTS_IEC_WORD wArea; RTS_IEC_DWORD dwOffset; RTS_IEC_DWORD dwSize; RTS_IEC_DWORD dwHighestUsedAddress; } APP_MEMORY_SEGMENT;

2.1.104 Typedef: APP_MEMORY_SEGMENT_INFO

Stuctname: APP_MEMORY_SEGMENT_INFO
Category: Application memory segment information
Describes all memory segments of an application.
Typedef: typedef struct _APP_MEMORY_SEGMENT_INFO { RTS_IEC_DINT diSegments; APP_MEMORY_SEGMENT *pmsList; } APP_MEMORY_SEGMENT_INFO;

2.1.105 Typedef: APPLICATION

Stuctname: APPLICATION
Category: Application description
ATTENTION: Allways add new elements at the end of the structure!!! This structure will be referred to generated IEC task code!
Typedef: typedef struct tagAPPLICATION { struct tagAPPLICATION *pAppParent; RTS_I32 iId; RTS_GUID CodeGuid; RTS_GUID DataGuid; RTS_UI32 ulState; RTS_UI32 ulOpState; RTS_HANDLE hBootproject; RTS_HANDLE hDebugTask; PF_GLOBAL_INIT pfGlobalInit; PF_GLOBAL_EXIT pfGlobalExit; RTS_HANDLE hDummy; Was previously hSessionIdPool, but is removed. Because of backward compatibility, it must be remaining here! RTS_HANDLE hForcePool; RTS_HANDLE hBPPool; RTS_I32 bPersistentForce; char szName[MAX_LEN_APPLNAME]; char szBootprojectName[MAX_PATH_LEN + MAX_LEN_APPLNAME]; RTS_UI32 ulPSVersion; RTS_UI32 ulTargetSettingVersion; ATTENTION: The member variables above are exported via CmpApp.library in IEC! Don't do any changes in this area!!! RTS_UI8 *pcArea[APPL_NUM_OF_STATIC_AREAS]; RTS_UI16 ausAreaType[APPL_NUM_OF_STATIC_AREAS]; RTS_UI32 aulAreaSize[APPL_NUM_OF_STATIC_AREAS]; RTS_UI8 byForcePool[MEM_GET_STATIC_LEN_(0,0)]; RTS_UI8 byBPPool[MEM_GET_STATIC_LEN_(0,0)]; FlowControl flowControl; RTS_GUID SafedCodeGuid; RTS_GUID SafedDataGuid; RetainType rtRetainType; RTS_UI32 ulException; RTS_I32 bInvalidateByRename; RTS_I32 bInvalidateBySetting; RTS_UINTPTR consistencyFlags; RTS_VOID_FCTPTR *ppfGetBooleanProperty; RTS_VOID_FCTPTR *ppfGetTextProperty; RTS_VOID_FCTPTR *ppfGetNumberProperty; RTS_VOID_FCTPTR *ppfGetVersionProperty; RTS_I32 bInitRetains; RTS_UI32 ulFlags; Application flags. See corresponding category. RTS_UI32 ulOffsetExtRefFunctionPointer; Area must be area 0! Offset must be valid for SystemApplications. The offset where to find the function to link functions APPLICATION_INFO ApplicationInfo; POU_REF pouMemorySegmentInfo; POU_REF pouAppInfo; RTS_UI32 ulCRCBootproject; ATTENTION: Always append new elements at the end of this structure! } APPLICATION;

2.1.106 Typedef: COMPACT_CODE_HEADER

Stuctname: COMPACT_CODE_HEADER
Category: Compact download header
This header is sent right at the beginning of each compact download stream
Typedef: typedef struct _COMPACT_CODE_HEADER { RTS_UI32 ulHeaderTag; RTS_UI32 ulHeaderVersion; RTS_UI32 ulHeaderSize; RTS_UI32 ulTotalSize; RTS_UI32 ulDeviceType; RTS_UI32 ulDeviceId; RTS_UI32 ulDeviceVersion; RTS_UI32 ulFlags; RTS_UI32 ulCompilerVersion; RTS_UI32 ulCodeAreaSize; RTS_UI16 usCodeAreaIndex; RTS_UI16 usCodeAreaFlags; RTS_UI32 ulOffsetCode; RTS_UI32 ulSizeCode; RTS_UI32 ulOffsetApplicationInfo; RTS_UI32 ulSizeApplicationInfo; RTS_UI32 ulOffsetAreaTable; RTS_UI32 ulSizeAreaTable; RTS_UI32 ulOffsetFunctionTable; RTS_UI32 ulSizeFunctionTable; RTS_UI32 ulOffsetExternalFunctionTable; RTS_UI32 ulSizeExternalFunctionTable; RTS_UI32 ulOffsetRegisterIecFunctionTable; RTS_UI32 ulSizeRegisterIecFunctionTable; RTS_UI32 ulOffsetSourceCode; RTS_UI32 ulSizeSourceCode; RTS_UI32 ulCrc; } COMPACT_CODE_HEADER;

2.1.107 appnumofactivesessions

*void appnumofactivesessions (appnumofactivesessions_struct *p)*

Retrieves the number of active sessions

pApp [IN]
Pointer to application.

pulNumSessions [OUT]
Number of active sessions.

Result
error code

2.1.108 AppCreateApplication

APPLICATION AppCreateApplication (char* pszAppName, char* pszAppParentName, RTS_RESULT *pResult)*

Creates an application specified by name

pszAppName [IN]
Pointer to name of the application

pszAppParentName [IN]
Pointer to name of the parent application (if a child application should be created). Can be NULL.

pResult [OUT]
Pointer to error code

Result
Pointer to the application description

2.1.109 AppPrepareDownload

RTS_RESULT AppPrepareDownload (APPLICATION pApp, int bOnlineChange, int bCreateBootproject)*

Prepares a new download or online-change

pApp [IN]
Pointer to the specified application description

bOnlineChange [IN]
1=Online change, 0=Download

bCreateBootproject [IN]
1=Create implicitly a bootproject, 0=Create no bootproject

Result
error code

2.1.110 AppAppendCode

*RTS_RESULT AppAppendCode (APPLICATION *pApp, RTS_UI8 *pbyCode, RTS_SIZE ulCodeLen, RTS_SIZE ulCodeOffset, int bLoadBootproject)*

Append IEC code for each code fragment. NOTE: Actually only available for CmpAppEmbedded!

pApp [IN]
Pointer to the specified application description

pbyCode [IN]
Pointer to code part

ulCodeLen [IN]
Code fragment length

ulCodeOffset [IN]
Code offset of the fragment, to write to

bLoadBootproject [IN]
1=Function is called at loading bootproject, 0=Else (e.g. at download sequence)

Result
error code

2.1.111 AppCompleteDownload

RTS_RESULT AppCompleteDownload (APPLICATION pApp, int bOnlineChange, BINTAGWRITER *pWriter)*

Complete the download and init application. NOTE: Actually only available for CmpAppEmbedded!

pApp [IN]
Pointer to the specified application description
bOnlineChange [IN]
1=Online change, 0=Download
pWriter [IN]
Pointer to the online writer. Can be NULL.
Result
error code

2.1.112 AppFindApplicationByName

APPLICATION AppFindApplicationByName (char *pszAppName, RTS_RESULT *pResult)*
Retrieves an application by name
pszAppName [IN]
Pointer to name of the application
pResult [OUT]
Pointer to error code
Result
Pointer to the application description, null if no App could not be found

2.1.113 AppFindApplicationByBootproject

APPLICATION AppFindApplicationByBootproject (char *pszBootprojectName, RTS_RESULT *pResult)*
Retrieves an application by name
pszBootprojectName [IN]
Pointer to name of the bootproject. Must not be the application name!
pResult [OUT]
Pointer to error code
Result
Pointer to the application description

2.1.114 AppFindApplicationBySessionId

APPLICATION AppFindApplicationBySessionId (RTS_UI32 ulSessionId, RTS_RESULT *pResult)*
Retrieves an application by session id. The session id is an unique id that is provided from the login service. There is a relation between the communication channel and the session id.
ulSessionId [IN]
SessionId from login service
pResult [OUT]
Pointer to error code
Result
Pointer to the application description

2.1.115 AppFindApplicationById

APPLICATION AppFindApplicationById (int iId, RTS_RESULT *pResult)*
Retrieves an application by its id.
iId [IN]
Id of the application
pResult [OUT]
Pointer to error code
Result
Pointer to the application description

2.1.116 AppGetNumOfApplications

int AppGetNumOfApplications (void)
Retrieves the number of registered and loaded applications.
Result
Number of registered applications

2.1.117 AppGetApplicationByIndex

APPLICATION AppGetApplicationByIndex (int iIndex, RTS_RESULT *pResult)*
Retrieves an application description sepcified by index
iIndex [IN]
Index of the application list.
pResult [OUT]
Pointer to error code
Result
Pointer to the application description

2.1.118 AppGetApplicationByAreaAddress

*APPLICATION * AppGetApplicationByAreaAddress (void *pAddress, RTS_RESULT *pResult)*
Retrieves an application description specified by area address
pAddress [IN]
area address.
pResult [OUT]
Pointer to error code
Result
Pointer to the application description

2.1.119 AppGetAreaPointer

*RTS_RESULT AppGetAreaPointer (APPLICATION *pApp, int iArea, unsigned char **ppucArea)*
Retrieves the pointer to a memory area specified by index
pApp [IN]
Pointer to the specified application description
iArea [IN]
Area index
ppucArea [OUT]
Pointer pointer to the area
Result
error code

2.1.120 AppGetAreaPointer2

*RTS_RESULT AppGetAreaPointer2 (APPLICATION *pApp, RTS_I32 iArea, RTS_UI8 **ppbyArea, RTS_SIZE *pulSize)*
Retrieves the pointer and size of a memory area specified by index
pApp [IN]
Pointer to the specified application description
iArea [IN]
Area index
ppucArea [OUT]
Pointer pointer to the area. Can be NULL to get only the area size.
pulSize [OUT]
Pointer to size to return the area size. Can be NULL to get only the area address.
Result
error code

2.1.121 AppGetAreaPointerByType

RTS_RESULT AppGetAreaPointerByType (char pszAppName, int iArea, RTS_UI16 usType, unsigned char **ppucArea)*
Retrieves a pointer to the memory area specified by type and index
pszAppName [IN]
Application name
iArea [IN]
Area index
usType [IN]
Area type. See category "Area Types" in SysMemItf.h.
ppucArea [OUT]
Pointer pointer to the area

Result
error code

2.1.122 AppHasAddress

RTS_RESULT AppHasAddress (APPLICATION pApp, RTS_VOID_FCTPTR pAddress)*
Function checks, if the address is a part of the specified application (resides in an area)
pApp [IN]
Application
pAddress [IN]
Pointer to check. Can be a data pointer or a function pointer.
Result
ERR_OK: pAddress is part of the specified application ERR_FAILED: pAddress is not a part of the specified application

2.1.123 AppIsProgramLoaded

*RTS_RESULT AppIsProgramLoaded (APPLICATION *pApp, int* pbProgramLoaded)*
Returns the load state of an application
pApp [IN]
Pointer to the specified application description
pbProgramLoaded [OUT]
Pointer to result. If the application contains a loaded project: *pbProgramLoaded is set to 1, else 0
Result
error code

2.1.124 AppSetProgramLoaded

*RTS_RESULT AppSetProgramLoaded (APPLICATION *pApp, int bProgramLoaded)*
Set the load state of an application
pApp [IN]
Pointer to the specified application description
bProgramLoaded [IN]
1=Application is loaded, 0=Application is not loaded
Result
error code

2.1.125 AppSingleCycle

*RTS_RESULT AppSingleCycle (APPLICATION *pApp)*
Activate a single cycle on the specified application
pApp [IN]
Pointer to the specified application description
Result
error code

2.1.126 AppSetState

*RTS_RESULT AppSetState (APPLICATION *pApp, unsigned long ulState)*
Set the state of an application
pApp [IN]
Pointer to the specified application description
ulState [IN]
State to set. See above for state specifications (prefix: AS_)
Result
error code

2.1.127 AppGetState

*unsigned long AppGetState (APPLICATION *pApp, RTS_RESULT *pResult)*
Returns the state of an application
pApp [IN]
Pointer to the specified application description

pResult [OUT]
Pointer to error code

Result
State of the application. There could be several active states at the same time! See above for state specifications (prefix: AS_)

2.1.128 AppSetOperatingState

*RTS_RESULT AppSetOperatingState (APPLICATION *pApp, unsigned long ulOpState)*

Set the operating state of an application

pApp [IN]
Pointer to the specified application description

ulOpState [IN]
Operating state of an application. This is an information, which job or operating possibilities of the application are set actually. There could be several active opertating states at the same time!

Result
error code

2.1.129 AppResetOperatingState

*RTS_RESULT AppResetOperatingState (APPLICATION *pApp, unsigned long ulOpState)*

Reset a single operating state of an application

pApp [IN]
Pointer to the specified application description

ulOpState [IN]
Operating state to reset

Result
error code

2.1.130 AppGetOperatingState

*unsigned long AppGetOperatingState (APPLICATION *pApp, RTS_RESULT *pResult)*

Get the operating state of an application

pApp [IN]
Pointer to the specified application description

pResult [OUT]
Pointer to error code

Result
Operating state of an application

2.1.131 AppSetCodeGuid

*RTS_RESULT AppSetCodeGuid (APPLICATION *pApp, RTS_GUID *pCodeGuid)*

Sets the code id of the application. The code id is a unique number that specifies the code content of an application.

pApp [IN]
Pointer to the specified application description

pCodeGuid [IN]
Pointer to code guid

Result
error code

2.1.132 AppGetCodeGuid

*RTS_RESULT AppGetCodeGuid (APPLICATION *pApp, RTS_GUID *pCodeGuid)*

Retrieves the code id of the application. The code id is a unique number that specifies the code content of an application.

pApp [IN]
Pointer to the specified application description

pCodeGuid [OUT]
Pointer to code guid

Result
error code

2.1.133 AppSetDataGuid

*RTS_RESULT AppSetDataGuid (APPLICATION *pApp, RTS_GUID *pDataGuid)*

Sets the data id of the application. The data id is a unique number that specifies the data configuration of an application.

pApp [IN]
Pointer to the specified application description

pDataGuid [IN]
Pointer to data guid

Result
error code

2.1.134 AppGetDataGuid

*RTS_RESULT AppGetDataGuid (APPLICATION *pApp, RTS_GUID *pDataGuid)*

Retrieves the data id of the application. The data id is a unique number that specifies the data configuration of an application.

pApp [IN]
Pointer to the specified application description

pDataGuid [OUT]
Pointer to data guid

Result
error code

2.1.135 AppGetBootprojectGuids

*RTS_RESULT AppGetBootprojectGuids (APPLICATION *pApp, RTS_GUID *pCodeGuid, RTS_GUID *pDataGuid)*

Retrieves the code and data guid of the bootproject of the given application. This function can be used to compare the guids of the loaded application with the bootproject.

pApp [IN]
Pointer to the specified application description

pCodeGuid [OUT]
Pointer to code guid

pDataGuid [OUT]
Pointer to data guid

Result
error code

2.1.136 AppExceptionHandler

*RTS_RESULT AppExceptionHandler (APPLICATION *pApp, RTS_UI32 ulException, RegContext Context)*

Exception handling of an application task. Must be called from another component, if an exception occurred in an IEC task. This routine is called typically by the CmplecTask.

pApp [IN]
Pointer to the specified application description, in which the exception was generated.

ulException [IN]
Exception number to see, which exception was generated

Context [IN]
Context of the task. With this context it is possible to investigate the complete callstack to the code position, where the exception occurred

Result
error code

2.1.137 AppReset

RTS_RESULT AppReset (APPLICATION pApp, RTS_UI16 usResetOption, RTS_UI32 ulSessionId)*

Executes a reset on the specified application

pApp [IN]
Pointer to the specified application description

usResetOption [IN]

Reset option. See the category reset options for detailed information

Result

error code

2.1.138 AppResetAllApplications

RTS_RESULT AppResetAllApplications (RTS_UI16 usResetOption)

Resets all applications

usResetOption [IN]

Reset option. See the category reset options for detailed information

Result

error code

2.1.139 AppLoadBootprojects

RTS_RESULT AppLoadBootprojects (void)

Load all registered bootprojects

Result

error code

2.1.140 AppStartBootprojects

RTS_RESULT AppStartBootprojects (void)

Start all registered bootprojects

Result

error code

2.1.141 AppLoadBootproject

*RTS_RESULT AppLoadBootproject (char *pszAppName, char *pszFilePath)*

Load a bootproject with the specified application name

pszAppName [IN]

Pointer to the NUL terminated application name

pszFilePath [IN]

File path for the bootproject

Result

error code

2.1.142 AppStartApplications

RTS_RESULT AppStartApplications (void)

Start all applications

Result

error code

2.1.143 AppStartApplication

*RTS_RESULT AppStartApplication (APPLICATION *pApp)*

Start an application

pApp [IN]

Pointer to the application

Result

error code

2.1.144 AppStopApplications

RTS_RESULT AppStopApplications (RTS_UI32 ulTimeoutMs, RTS_UI32 ulStopReason)

Stop all applications

ulTimeoutMs [IN]

Timeout in milliseconds to wait for stop

ulStopReason [IN]

Stop reason, See corresponding category

Result

error code

2.1.145 AppStopApplication

*RTS_RESULT AppStopApplication (APPLICATION *pApp, RTS_UI32 ulTimeoutMs, RTS_UI32 ulStopReason)*

Stop an application

pApp [IN]
Pointer to the application

ulTimeoutMs [IN]
Timeout in milliseconds to wait for stop

ulStopReason [IN]
Stop reason, See corresponding category

Result
error code

2.1.146 AppExitApplication

*RTS_RESULT AppExitApplication (APPLICATION *pApp, int bShutdown)*

Exit an application (release all tasks, etc.). The application is not deleted!

pApp [IN]
Pointer to the application

bShutdown [IN]
1=Function called at shutdown of the runtime, 0=else

Result
error code

2.1.147 AppDeleteApplication

*RTS_RESULT AppDeleteApplication (APPLICATION *pApp, int bShutdown)*

Delete an application. Bootproject will not deleted!

pApp [IN]
Pointer to the application

bShutdown [IN]
1=Function called at shutdown of the runtime, 0=else

Result
error code

2.1.148 AppDeleteApplications

RTS_RESULT AppDeleteApplications (int bShutdown)

Delete all applications. Bootprojects will not be deleted!

bShutdown [IN]
1=Function called at shutdown of the runtime, 0=else

Result
error code

2.1.149 AppDeleteBootproject

*RTS_RESULT AppDeleteBootproject (APPLICATION *pApp)*

Delete a bootproject (corresponding loaded application will not be affected or deleted!)

pApp [IN]
Pointer to the application

Result
error code

2.1.150 AppAddAddrDataSrv

*RTS_RESULT AppAddAddrDataSrv (RTS_UI32 ulSessionId, PEERADDRESS *pPeerAddr)*

Add the peer address of an existing data server. This can be used from clients, that wants to get symbolic access to IEC variables. If the symbolic infromation is not available here on the controller, the registered data server can be used for that.

ulSessionId [IN]
SessionId of the communication channel

pPeerAddr [IN]

Pointer to the peer address of the data server

Result

error code

2.1.151 AppRemoveAddrDataSrv

RTS_RESULT AppRemoveAddrDataSrv (RTS_UI32 ulSessionId)

Remove a peer address of an existing data server that will be shutdown

ulSessionId [IN]

SessionId of the communication channel

Result

error code

2.1.152 AppGetFirstAddrDataSrv

*RTS_HANDLE AppGetFirstAddrDataSrv (PEERADDRESS **ppPeerAddr, RTS_RESULT *pResult)*

Retrieves the peer address of the first registered data server

ppPeerAddr [OUT]

Pointer pointer to the peer address

pResult [OUT]

Pointer to error code

Result

Handle to get the next peer addresse of a registered data server, RTS_INVALID_HANDLE if not available

2.1.153 AppGetNextAddrDataSrv

*RTS_HANDLE AppGetNextAddrDataSrv (RTS_HANDLE hAddr, PEERADDRESS **ppPeerAddr, RTS_RESULT *pResult)*

Retrieves the peer address of the first registered data server

hAddr [IN]

Handle, that was retrieved by the AppGetFirstAddrDataSrv() function

ppPeerAddr [OUT]

Pointer pointer to the peer address

pResult [OUT]

Pointer to error code

Result

Handle to get the next peer addresse of a registered data server, RTS_INVALID_HANDLE if not available

2.1.154 AppConsistencyCheckBegin

*RTS_HANDLE AppConsistencyCheckBegin (APPLICATION *pApp, RTS_RESULT *pResult)*

Begin a consistency sequence. This can be used to check, if any task of an IEC application (or its father application) was active during this sequence. A typical usage is the task consistent data monitoring during this sequence.

pApp [IN]

Pointer to the application

pResult [OUT]

Pointer to error code

Result

Return a handle to the consistency object

2.1.155 AppConsistencyCheckEnd

RTS_RESULT AppConsistencyCheckEnd (RTS_HANDLE hConsistency)

End a consistency sequence. This can be used to check, if any task of an IEC application (or its father application) was active during this consistency sequence. A typical usage is the task consistent data monitoring.

hConsistency [IN]

Handle to the consistency object that is returned by AppConsistencyBegin()

Result

ERR_OK: No task active during this consistency sequence, ERR_FAILED: Any task of this application was active during this sequence

2.1.156 AppConsistencyCheckAll

RTS_RESULT AppConsistencyCheckAll (void)

Check all applications for consistency, that are called with AppConsistencyCheckBegin()

Result

ERR_OK: No task active during this consistency sequence, ERR_FAILED: Any task of an application was active during this sequence

2.1.157 AppGetFirstApp

APPLICATION AppGetFirstApp (RTS_RESULT *pResult)*

Retrieves the application info of the first application

pResult [OUT]

Pointer to error code

Result

Pointer to the first application

2.1.158 AppGetNextApp

APPLICATION AppGetNextApp (APPLICATION *pAppPrev, RTS_RESULT *pResult)*

Retrieves the application info of the next application

pAppPrev [IN]

Pointer to the first application retrieved by the AppGetFirstApp()

pResult [OUT]

Pointer to error code

Result

Pointer to the next application

2.1.159 AppStoreRetainsInFile

*RTS_RESULT AppStoreRetainsInFile (APPLICATION *pApp, RTS_IEC_STRING* pszFileName)*

Stores the retains of an application in a file

pApp [IN]

Pointer to application

pszFilename [OUT]

Name of retain file

Result

Error code

2.1.160 AppRestoreRetainsFromFile

*RTS_RESULT AppRestoreRetainsFromFile (APPLICATION *pApp, RTS_IEC_STRING* pszFileName)*

Restores the retains of an application from a file

pApp [IN]

Pointer to application

pszFilename [IN]

Name of retain file

Result

Error code

2.1.161 AppRestoreRetainsFromFile2

*RTS_RESULT AppRestoreRetainsFromFile2 (APPLICATION *pApp, RTS_IEC_STRING* pszFileName, int bGenerateException)*

Restores the retains of an application from a file

pApp [IN]

Pointer to application

pszFilename [IN]

Name of retain file

bGenerateException [IN]

Select behaviour on retain mismatch: 1=Generate exception, 0=Only return error code

Result

Error code

2.1.162 AppSaveRetainAreas

*RTS_RESULT AppSaveRetainAreas (APPLICATION *pApp)*
Store retain areas the standard way for the next reboot or reload
pApp [IN]
Pointer to application
Result
Error code

2.1.163 AppRestoreRetainAreas

*RTS_RESULT AppRestoreRetainAreas (APPLICATION *pApp)*
Restore retain areas the standard way
pApp [IN]
Pointer to application
Result
Error code

2.1.164 AppGetAreaSize

RTS_SIZE AppGetAreaSize (APPLICATION pApp, RTS_UI16 usType, RTS_RESULT* pResult)*
This function retuns the size of an application area
pApp [IN]
Pointer to Application
usType [IN]
Area type. See category "Area Types" in SysMemItf.h.
pResult [OUT]
Pointer to Result
Result
Area size

2.1.165 AppGetAreaAddress

RTS_UI8 AppGetAreaAddress (APPLICATION* pApp, RTS_UI16 usType, RTS_RESULT* pResult)*
This function retuns the start address of an application area
pApp [IN]
Pointer to the specified application description
usType [IN]
Area type. See category "Area Types" in SysMemItf.h.
pResult [OUT]
Pointer to Result
Result
Area start address

2.1.166 AppGetSegment

*APP_MEMORY_SEGMENT * AppGetSegment (APPLICATION* pApp, RTS_UI16 usType, RTS_RESULT *pResult)*
This function retuns the segment info specified by type
pApp [IN]
Pointer to Application
usType [IN]
Area type. See category "Area Types" in SysMemItf.h.
pResult [OUT]
Pointer to Result
Result
Pointer to segment info

2.1.167 AppGetSegmentByAddress

*APP_MEMORY_SEGMENT * AppGetSegmentByAddress (APPLICATION* pApp, void *pAddress, RTS_RESULT *pResult)*

This function retuns the segment info specified by address

pApp [IN]
Pointer to Application

pAddress [IN]
Pointer to segment data

pResult [OUT]
Pointer to Result

Result
Pointer to segment info

2.1.168 AppGetSegmentSize

RTS_SIZE AppGetSegmentSize (APPLICATION pApp, RTS_UI16 usType, RTS_RESULT* pResult)*

This function retuns the start size of an IEC segment. All segments resides within an area. This is used to get access for example to the size of the output processimage segment (%Q).

pApp [IN]
Pointer to Application

usType [IN]
Area type. See category "Area Types" in SysMemItf.h.

pResult [OUT]
Pointer to Result

Result
Segment size

2.1.169 AppGetSegmentAddress

*RTS_UI8 HUGEPTR * AppGetSegmentAddress (APPLICATION* pApp, RTS_UI16 usType, RTS_RESULT* pResult)*

This function retuns the start address of an IEC segment. All segments resides within an area. This is used to get access for example to the beginning of the output processimage segment (%Q).

pApp [IN]
Pointer to Application

usType [IN]
Area type. See category "Area Types" in SysMemItf.h.

pResult [OUT]
Pointer to Result

Result
Segment start address

2.1.170 AppRegisterPropAccessFunctions

RTS_RESULT AppRegisterPropAccessFunctions (APPLICATION pApp, RTS_VOID_FCTPTR *ppfGetBooleanProperty, RTS_VOID_FCTPTR *ppfGetTextProperty, RTS_VOID_FCTPTR *ppfGetNumberProperty, RTS_VOID_FCTPTR *ppfGetVersionProperty)*

This function registers the properties access functions

pApp [IN]
Pointer to Application

ppfGetBooleanProperty [IN]
Pointer to Pointer to Boolean Function

ppfGetTextProperty [IN]
Pointer to Pointer to Text Function

ppfGetNumberProperty [IN]
Pointer to Pointer to Number Function

ppfGetVersionProperty [IN]
Pointer to Pointer to Version Function

Result
Error Code

2.1.171 AppGetProjectInformation

RTS_RESULT AppGetProjectInformation (APPLICATION pApp, PROJECT_INFO* pInfo)*

This function returns the project information

pApp [IN]
Pointer to Application
pfGetBooleanProperty [IN]
Pointer to PROJECT_INFORMATION
Result
Error Code

2.1.172 AppGetBooleanProperty

*RTS_RESULT AppGetBooleanProperty (APPLICATION *pApp, BOOLEANPROPERTY *pBooleanProperty)*
With this function you get access to the optional IEC function in the project, which contains boolean project infomration
pApp [IN]
Pointer to Application
pBooleanProperty [INOUT]
wszKey must be set with the key (RTS_IEC_WSTRING!), as defined in the project information dialog in CoDeSys. See category "Property keys" for predefined property keys. bBooleanProperty contains the bool property value.
Result
Error Code

2.1.173 AppGetTextProperty

*RTS_RESULT AppGetTextProperty (APPLICATION *pApp, TEXTPROPERTY *pTextProperty)*
With this function you get access to the optional IEC function in the project, which contains text project infomration
pApp [IN]
Pointer to Application
pTextProperty [INOUT]
wszKey must be set with the key (RTS_IEC_WSTRING!), as defined in the project information dialog in CoDeSys. See category "Property keys" for predefined property keys. wszTextProperty contains the string property value.
Result
Error Code

2.1.174 AppGetNumberProperty

*RTS_RESULT AppGetNumberProperty (APPLICATION *pApp, NUMBERPROPERTY *pNumberProperty)*
With this function you get access to the optional IEC function in the project, which contains text project infomration
pApp [IN]
Pointer to Application
pNumberProperty [INOUT]
wszKey must be set with the key (RTS_IEC_WSTRING!), as defined in the project information dialog in CoDeSys. See category "Property keys" for predefined property keys. udiNumberProperty contains the number property value.
Result
Error Code

2.1.175 AppGetVersionProperty

*RTS_RESULT AppGetVersionProperty (APPLICATION *pApp, VERSIONPROPERTY *pVersionProperty)*
With this function you get access to the optional IEC function in the project, which contains text project infomration
pApp [IN]
Pointer to Application
pVersionProperty [INOUT]
wszKey must be set with the key (RTS_IEC_WSTRING!), as defined in the project information dialog in CoDeSys. See category "Property keys" for predefined property keys. uiMajor, uiMinor, uiServicePack, uiPatch contains the version property values.
Result

Error Code

2.1.176 AppGetCurrent

APPLICATION AppGetCurrent (RTS_RESULT *pResult)*

Get the current application, in which task context the caller is located

pResult [OUT]
Pointer to error code

Result
Pointer to application description

2.1.177 AppCheckFileConsistency

*RTS_RESULT AppCheckFileConsistency (APPLICATION *pApp, RTS_RESULT *pBootprojectConsistency, RTS_RESULT *pArchiveConsistency)*

Check the consistency of the specified application to the files of bootproject and project archive

pApp [IN]
Pointer to Application

pBootprojectConsistency [OUT]
Pointer to result of the bootproject consistency. ERR_OK: Bootproject matches the specified application

pArchiveConsistency [OUT]
Pointer to result of the archive consistency. ERR_OK: Archive matches the specified application

Result
error code

2.1.178 AppGenerateException

*RTS_RESULT AppGenerateException (APPLICATION *pApp, RTS_UI32 ulException)*

Generate an exception on the specified application. NOTE: pApp can be NULL, so the current applicaiton in program download sequence is used!

pApp [IN]
Pointer to Application. Can be NULL!

ulException [IN]
Exception code. See SysExceptItf.h for details.

Result
error code

2.1.179 AppRegisterBootproject

*RTS_RESULT AppRegisterBootproject (char *pszBootproject)*

Function to register a bootproject to reload at the next startup

pszAppName [IN]
Name of the bootproject without ending or application name

Result
error code

2.1.180 AppUnregisterBootproject

*RTS_RESULT AppUnregisterBootproject (char *pszBootproject)*

Function to unregister a bootproject to avoid reload at the next startup

pszAppName [IN]
Name of the bootproject without ending or application name

Result
error code

2.1.181 AppCallGetProperty

*RTS_RESULT AppCallGetProperty (void *pInstance, RTS_VOID_FCTPTR *ppGetMethod, RTS_UI8 *pbyValue, RTS_SIZE ulSize)*

Function call a property of a function block to get the value

pInstance [IN]
Pointer to the FB instance

ppGetMethod [IN]

Pointer to the get method of the property (ADR(__get[PropertyName])

pbyValue [IN]
Pointer to a value buffer to return the value of the property

ulSize [IN]
Size in bytes of the property type

Result
error code

2.1.182 AppCallSetProperty

*RTS_RESULT AppCallSetProperty (void *pInstance, RTS_VOID_FCTPTR *ppSetMethod, RTS_UI8 *pbyValue, RTS_SIZE ulSize)*

Function call a property of a function block to set the value

pInstance [IN]
Pointer to the FB instance

ppSetMethod [IN]
Pointer to the set method of the property (ADR(__get[PropertyName])

pbyValue [IN]
Pointer to a value buffer to set the value of the property

ulSize [IN]
Size in bytes of the property type

Result
error code

2.1.183 AppGetAreaOffsetByAddress

*RTS_RESULT AppGetAreaOffsetByAddress (APPLICATION *pApp, RTS_UINTPTR ulAddress, RTS_UI16 *pusArea, RTS_SIZE *pulOffset)*

Get area number and offset of an application specified by a memory address

pApp [IN]
Pointer to Application

ulAddress [IN]
Memory address

pusArea [OUT]
Pointer to return area number

pulOffset [IN]
Pointer to return area offset

Result
error code

2.1.184 AppSrvGetApplication

APPLICATION AppSrvGetApplication (PROTOCOL_DATA_UNIT *pduSendBuffer, BINTAGREADER *preader, BINTAGWRITER *pwriter, RTS_UI32 ulSessionId)*

Returns the application specified in an online service

pduSendBuffer [IN]
Pointer to the sent data unit

preader [IN]
Pointer to the reader

pwriter [IN]
Pointer to the writer

ulSessionId [IN]
Session id of the given service

Result
Pointer to the application description

2.1.185 AppHasAccessRights

*RTS_RESULT AppHasAccessRights (char *pszApplication, char *pszObject, RTS_UI32 ulRequestedRights, RTS_UI32 ulSessionId, BINTAGWRITER *pWriter)*

Check the access rights to the specified object of the application

pszApplication [IN]

Name of the application

pszObject [IN]
Name of the object

ulRequestedRights [IN]
Requested rights on the object

ulSessionId [IN]
SessionID of the online service

pWriter [IN]
Pointer to the tag writer

Result
Error code: ERR_OK: Has access rights ERR_FAILED: Operation failed ERR_NO_ACCESS_RIGHTS: No access rights to the object

2.1.186 AppGetApplicationInfo

*APPLICATION_INFO * AppGetApplicationInfo (APPLICATION *pApp, RTS_RESULT *pResult)*

Get the detail information of the specified application

pApp [IN]
Pointer to Application

pResult [OUT]
Pointer to error code

Result
Pointer to application info structure

3 CmpBinTagUtil

Provides a reader and a writer for the BinTag structured data format.

3.1 CmpBinTagUtilItf

Interface for the binary tag utility.

3.1.1 Typedef: BTAG_ALIGNMENT

Stuctname: BTAG_ALIGNMENT

This struct defines an alignment property. The meaning of the members is "Align to an address for which the following equation holds: address % sModulus = sRemainder" Eg.: (4,0) aligns to a 4 byte boundary (4,1) aligns to a 4 byte boundary + 1 (1,5,9,13,...) (2,0) aligns to an equal address (2,1) aligns to an odd address (1,0) aligns to any address

Typedef: typedef struct { unsigned short usModulus; unsigned short usRemainder; } BTAG_ALIGNMENT;

3.1.2 Typedef: BINTAGWRITER

Stuctname: BINTAGWRITER

This struct holds the internal state of a writer and should be treated as opaque by an application. Do not alter!

Typedef: typedef struct { RTS_UI8 *pBuffer; RTS_HANDLE hFile; RTS_UI32 ulBufferSize; RTS_UI32 ulPos; BTAG_WRITERTAGINFO tagStack[BTAG_MAX_NESTED_TAGS]; int nStackPos; Points to the index of the current tag. Initially set to -1 (toplevel, no current tag). Must not exceed BTAG_MAX_NESTED_TAGS-1 int iType; RTS_UI32 ulEndServicePos; int bSwapHeader; int bBufferOverflow; PFUPDATECRC pfUpdateCRC; void *pParameterUpdateCRC; }BINTAGWRITER;

3.1.3 Typedef: BINTAGREADER

Stuctname: BINTAGREADER

This struct holds the internal state of a reader and should be treated as opaque by an application. Do not alter!

Typedef: typedef struct { RTS_UI8 *pBuffer; RTS_UI32 ulBufferSize; RTS_HANDLE hFile; RTS_UI32 ulPos; BTAG_READERTAGINFO tagStack[BTAG_MAX_NESTED_TAGS]; int nStackPos; Points to the index of the current tag. Initially set to -1 (toplevel, no current tag). Must not exceed BTAG_MAX_NESTED_TAGS-1 int iType; RTS_UI32 ulStartServicePos; }BINTAGREADER;

3.1.4 BTagSwapHeader

int BTagSwapHeader (HEADER_TAG_EXT pHeader, int bSwap)*

Swap the header of a service.

pHeader [INOUT]

Pass in a HEADER_TAG_EXT struct that should be swapped.

bSwap [IN]

Determines, if the header should be swapped (1) or not (0).

3.1.5 BTagWriterInit2

*int BTagWriterInit2 (BINTAGWRITER *pWriter, RTS_UI8 *pBuffer, RTS_UI32 ulBufferSize, int bSwapHeader)*

Initialize a writer.

pWriter [INOUT]

Pass in a BINTAGWRITER struct that will be initialized to a empty writer.

pBuffer [IN]

The buffer that the writer will write to. The buffer should not be altered until the writer has finished.

ulBufferSize [IN]

The size of the buffer. The writer will fail if a write operation would exceed the buffer.

bSwapHeader [IN]

Clients have to set this flag, if the addressed server has a different byte order. Must be always FALSE for server implementations or if the client has the same byte order as the server.

3.1.6 BTagWriterInit

*int BTagWriterInit (BINTAGWRITER *pWriter, RTS_UI8 *pBuffer, RTS_UI32 ulBufferSize)*

Initialize a writer. Can be used by servers or if the client and the server have the same byte order.

pWriter [INOUT]
Pass in a BINTAGWRITER struct that will be initialized to a empty writer.
pBuffer [IN]
The buffer that the writer will write to. The buffer should not be altered until the writer has finished.
ulBufferSize [IN]
The size of the buffer. The writer will fail if a write operation would exceed the buffer.

3.1.7 BTagWriterStartService

*int BTagWriterStartService (BINTAGWRITER *pWriter, RTS_UI32 ulSessionID, RTS_UI16 usHeaderTag, RTS_UI32 ulServiceGroup, RTS_UI16 usService)*
Start a new service. Must be called after BTagWriterInit.
pWriter [INOUT]
The writer.
ulSessionID [IN]
SessionID of the current session
usHeaderTag [IN]
HeaderTag to identify the protocol handler.
ulServiceGroup [IN]
ulServiceGroup = HIGHWORD: usCustomerId, LOWWORD: usServiceGroup;
usService [IN]
Service

3.1.8 BTagWriterFinishService

*int BTagWriterFinishService (BINTAGWRITER *pWriter, RTS_UI8 **ppBuffer, RTS_UI32 *pulSize)*
Finishes the service. It returns the buffer passed in to BTagWriterInit in ppBuffer and the number of bytes written to the buffer in pulSize;

3.1.9 BTagWriterStartTag

*int BTagWriterStartTag (BINTAGWRITER *pWriter, RTS_UI32 ulTagId, BTAG_ALIGNMENT contentAlignment, RTS_UI32 ulMinLengthSize)*
Start a new tag. A tag cannot be started within a data tag. Data tags are determined by the id of the tag: If bit 7 (that is the highest bit of the least significant byte of the tagid) is set then the tag contains only subtags, otherwise it contains no subtags but only content. Every call to BTagWriterStartTag must be matched with a call to BTagWriterEndTag. All subtags of a tag must be closed before the tag itself may be closed.
pWriter [INOUT]
The writer must have been initialized with a call to BTagWriterInit.
ulTagId [IN]
The id of the tag.
contentAlignment [IN]
How the content should be aligned within this tag. If the tag is not a data tag then contentAlignment must always be (4,0). In any case in this version of the data format the nModulus member must always be 4 since all alignment is done in respect to 4 byte boundaries.
ulMinLengthSize [IN]

3.1.10 BTagWriterAppendString

*int BTagWriterAppendString (BINTAGWRITER *pWriter, const char *pszString)*
Append a C-string to the current tag. The string is written including the trailing end of string char (NUL). The current tag must be a data tag.
pWriter [IN]
Pointer to bintag writer
pszString [IN]
Pointer to NUL terminated string to append
Result
TRUE=succeeded, FALSE=failed

3.1.11 BTagWriterAppendWString

*int BTagWriterAppendWString (BINTAGWRITER *pWriter, const RTS_WCHAR *wszString)*

Append a UTF-16 (widechar) string to the current tag. The string is written including the trailing end of string char (0x0000). The current tag must be a data tag.

3.1.12 BTagWriterAppendBlob

*int BTagWriterAppendBlob (BINTAGWRITER *pWriter, const RTS_UI8 *pBlob, RTS_UI32 ulSize)*

Append ulSize bytes from the buffer pBlob to the current tag. The current tag must be a data tag.

3.1.13 BTagWriterAppendRaw

*int BTagWriterAppendRaw (BINTAGWRITER *pWriter, RTS_UI8 **ppBuffer, RTS_UI32 ulSize)*

Returns a pointer to the current content position and forwards the write position for ulSize bytes. This function provides a buffer of length ulSize within the content of the current tag, in effect giving the caller random access to this buffer. This function is especially usefull, if that buffer has to be passed to another function, that fills it. (see example). ATTENTION: The returned pointer is valid only until the next operation on the bintagwriter is executed. After that the pointer must not be used any more! Do not store that pointer permanently. The current tag must be a data tag.

ppBuffer [OUT]

Is set to point to the buffer, if the function succeeds

// Store a number as hexadecimal string in a bintag writer unsigned char *pBuffer; unsigned char **

3.1.14 BTagWriterAppendFillBytes

*int BTagWriterAppendFillBytes (BINTAGWRITER *pWriter, RTS_UI8 byFillByte, BTAG_ALIGNMENT alignment)*

Add byFillByte to the content until the current content ends on a position that satifies the desired alignment. Eg. a tag must always be closed on a (4,0)-alignment, so after adding a variable length string one should call:

BTAG_ALIGNMENT align = {4,0}; ... BTagWriterAppendFillBytes(pWriter, 0, align); BTagWriterEndTag(pWriter, TAG_

If the alignment property is already fullfilled nothing will be appended.

pWriter [IN]

byFillByte [IN]

The byte to be appended until alignment is achieved.

alignment [IN]

The desired alignment of the next writer position.

3.1.15 BTagWriterAppendDummyBytes

*int BTagWriterAppendDummyBytes (BINTAGWRITER *pWriter, RTS_UI8 byFillByte, RTS_UI32 ulSize)*

Append byFillByte ulSize times.

3.1.16 BTagWriterEndTag

*int BTagWriterEndTag (BINTAGWRITER *pWriter, RTS_UI32 ulTagId)*

Close the current tag. The current tag must have been started with the given tag id. ulTagId is used only as an additional check that all tags are closed in the right order. Since the only element that follows a tag must be a tag again and a tag must always start on a 4 byte boundary the complete size of a tag (header + content) must be dividable by 4. Taking a contentalignment of (4,x) then "contentsize == 4 - x (MOD 4) " must hold. (eg. for (4,1): contentsize % 4 == 3) If this condition does not hold, the function will fail.

3.1.17 BTagWriterSwitchBuffer

*int BTagWriterSwitchBuffer (BINTAGWRITER *pWriter, RTS_UI8 *pNewBuffer, RTS_UI32 ulNewSize, RTS_UI8 **ppOldBuffer)*

Replaces the current writer buffer with a new one. All content in the current buffer is copied to the new buffer. This also means that the size of the new buffer has to be at least the current position of the writer. The purpose of this function is to allow for extension of the buffer if an operation returned a buffer overflow. Then the caller may allocate a new buffer, call BTagWriterSwitchBuffer and retry the failed operation.

pWriter [IN]

pNewBuffer [IN]

The buffer that is to replace the original buffer.

ulNewSize [IN]
Size of the new buffer. Must be greater or at least equal to the current postion of the writer.
ppOldBuffer [OUT]
Is set to the previous buffer.

3.1.18 BTagWriterCreateSavepoint

*int BTagWriterCreateSavepoint (BINTAGWRITER *pWriter, BINTAGSAVEPOINT *pSavepoint)*
Save the current state of the writer. A later call to BTagWriterRestoreSavepoint will reset the writer and its buffer to that state. Any number of savepoints may be created. The caller must use multiple savepoints in a stack-like fashion: If savepoints are created in the order 1,2,3,4 then they must be restored in opposite order only. Not every savepoint must be restored - but if one is restored all savepoints created after that one MUST NOT BE USED any more. Examples: ("sX"="create savepoint X", "rX"="Restore savepoint X", "(x,y,z)"="stack of valid savepoints") The following sequences are valid: () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) r1 () () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) s4 (1,4) r4 (1) whereas this one is invalid: () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) r3 [3 is not valid at this point] The writer does not track the stack of savepoints, therefore it cannot detect errors in the restore order. It's the responsibility of the application to ensure the correct restore order. Failure in doing so will lead to an inconsistent state and may well corrupt the whole document. Besides the restore order savepoints are independent of each other. The application may delete/reuse a savepoint anytime it doesn't need it anymore. Savepoints cannot be used to transfer the state of a writer to a second writer. Any attempt in doing so will lead to undefined behaviour.
pWriter [IN]
pSavePoint [OUT]
Will receive all information needed to restore the current state of the writer.

3.1.19 BTagWriterRestoreSavepoint

*int BTagWriterRestoreSavepoint (BINTAGWRITER *pWriter, BINTAGSAVEPOINT *pSavepoint)*
Restore the state of the writer to one previously saved using BTagWriterCreateSavepoint. See there for a detailed description on how to use these functions.
pWriter [IN]
Must be the same writer that the savepoint was created on.
pSavepoint [IN]
A previously created savepoint.

3.1.20 BTagWriterGetAvailableBuffer

*RTS_UI32 BTagWriterGetAvailableBuffer (BINTAGWRITER *pWriter)*
Reads the available bytes in writer buffer that can be used for data.

3.1.21 BTagWriterFinish

*int BTagWriterFinish (BINTAGWRITER *pWriter, RTS_UI8 **ppBuffer, RTS_UI32 *pulSize)*
Finishes the writer. If not all tags have been closed properly this function will fail. Otherwise it returns the buffer passed in to BTagWriterInit in ppBuffer and the number of bytes written to the buffer in pulSize;

3.1.22 BTagWriteSingleTag

*RTS_RESULT BTagWriteSingleTag (BINTAGWRITER *pWriter, RTS_UI32 ulTag, BTAG_ALIGNMENT align, int bFillBytes, void *pContent, RTS_UI32 ulSize)*
Write a single tag
pWriter [IN]
Pointer to the writer stream
ulTag [IN]
Tag to write
align [IN]
Alignment of the tag
pContent [IN]
Pointer to the tag data
ulSize [IN]
Size in bytes of the tag data
Result
error code

3.1.23 BTagWriteSingleTag2

*RTS_RESULT BTagWriteSingleTag2 (BINTAGWRITER *pWriter, RTS_UI32 ulTag, BTAG_ALIGNMENT align, void **ppContentList, RTS_UI32 *paulSize)*

Write a single tag with more than one content

pWriter [IN]
Pointer to the writer stream

ulTag [IN]
Tag to write

align [IN]
Alignment of the tag

pContent [IN]
Pointer to the tag data

ulSize [IN]
Size in bytes of the tag data

Result
error code

3.1.24 BTagReaderInit

*int BTagReaderInit (BINTAGREADER *pReader, RTS_UI8 *pBuffer, RTS_UI32 ulBufferSize)*

Initialize a reader.

pReader [INOUT]
Pass in a BINTAGREADER structure that will be initialized to an empty reader.

pBuffer [IN]
The buffer that contains the bintag structure to be read. Do not alter the buffer until the reader isn't used any more.

ulBufferSize [IN]
The size of pBuffer. It is expected that the whole buffer is in the bintag structure, possibly containing multiple toplevel tags.

3.1.25 BTagReaderPeekNext

*int BTagReaderPeekNext (BINTAGREADER *pReader)*

Peek at the type of the next element as it would be returned by a call to BTagReaderMoveNext, but do not actually move to it (ie. the readers state will not be changed).

pReader [IN]
An active reader.

pnElementType [OUT]
Will be set to the type of the current element:
- BTAG_ET_STARTTAG: Reader entered a new tag.
- BTAG_ET_ENDTAG: End of the current tag reached. New active tag is the surrounding tag
- BTAG_ET_EOF: End of the buffer reached. No more content/tags can be read

3.1.26 BTagReaderMoveNext

*int BTagReaderMoveNext (BINTAGREADER *pReader, int *pnElementType)*

Let the reader move on to the next element.

pReader [IN]
An active reader.

pnElementType [OUT]
Will be set to the type of the current element:
- BTAG_ET_STARTTAG: Reader entered a new tag.
- BTAG_ET_ENDTAG: End of the current tag reached. New active tag is the surrounding tag
- BTAG_ET_EOF: End of the buffer reached. No more content/tags can be read

3.1.27 BTagReaderSkipContent

*int BTagReaderSkipContent (BINTAGREADER *pReader)*

Jump to the end of the current tag. The next element that will be read by BTagReaderMoveNext will be the ENDTAG of the current tag.

pReader []

3.1.28 BTagReaderGetTagId

*int BTagReaderGetTagId (BINTAGREADER *pReader, RTS_UI32 *pulTagId)*

Get the id of the current tag. Will fail if the reader is positioned at the toplevel.

pulTagId [OUT]

Will be set to the id of the current tag.

3.1.29 BTagReaderGetTagLen

*int BTagReaderGetTagLen (BINTAGREADER *pReader, RTS_UI32 *pulTagLen)*

Get the tag length of the current tag. Will fail if the reader is positioned at the toplevel.

pulTagLen [OUT]

Will be set to the length of the current tag.

3.1.30 BTagReaderIsDataTag

*int BTagReaderIsDataTag (BINTAGREADER *pReader, int *pbIsDataTag)*

Check if the current tag is a data tag (ie. contains raw content) or does contain subtags. Will return an error if the reader is not positioned on a tag (ie. is at the toplevel). However, in that case pbIsData will be set to FALSE.

pReader [IN]

pbIsDataTag [IN]

Is set to TRUE, if the current tag is a data tag, FALSE otherwise.

3.1.31 BTagReaderGetComplexContent

*int BTagReaderGetComplexContent (BINTAGREADER *pReader, RTS_UI8 **ppBuffer, RTS_UI32 *pulSize)*

Get a pointer on the content of the current tag.

pReader [IN]

ppBuffer [OUT]

Will be set to point at the content of the current tag.

pulSize [OUT]

Will contain the size of the content.

3.1.32 BTagReaderGetContent

*int BTagReaderGetContent (BINTAGREADER *pReader, RTS_UI8 **ppBuffer, RTS_UI32 *pulSize)*

Get a pointer on the content of the current tag. Will fail if the reader is on the toplevel (no current tag) or the current tag is not a data tag.

pReader [IN]

ppBuffer [OUT]

Will be set to point at the content of the current tag.

pulSize [OUT]

Will contain the size of the content.

3.1.33 BTagReaderGetString

*int BTagReaderGetString (BINTAGREADER *pReader, char **ppString, RTS_UI32 *pulSize, int bAddEndOfString)*

Like BTagReaderGetContent but treats the content as a string. If a '\0' is found before the end of the tag, pulSize is set to the length up to (and including) the zero. If bAddEndOfString is set, then '\0' is written at the last byte, if '\0' is not found.

pReader [IN]

ppString [Out]

Will be set to the start of the string. Will not necessarily point into the readers buffer. (see note).

pulSize [IN]

The length of the string INCLUDING the trailing '\0', ie. "strlen(ppString)+1".

bAddEndOfString [IN]

Force a trailing zero byte to be added, if it doesn't exist within the bounds of the tag. Will overwrite the last char of the string. Note: This option will alter the content of the buffer!

3.1.34 BTagReaderCreateSavepoint

*int BTagReaderCreateSavepoint (BINTAGREADER *pReader, BINTAGREADERSAVEPOINT *pSavepoint)*

Save the current state of the reader. A later call to BTagReaderRestoreSavepoint will reset the reader and its buffer to that state. Any number of savepoints may be created. The caller must use multiple savepoints in a stack-like fashion: If savepoints are created in the order 1,2,3,4 then they must be restored in opposite order only. Not every savepoint must be restored - but if one is restored all savepoints created after that one MUST NOT BE USED any more. Examples: ("sX"="create savepoint X", "rX"="Restore savepoint X", "(x,y,z)"="stack of valid savepoints") The following sequences are valid: () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) r1 () () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) s4 (1,4) r4 (1) whereas this one is invalid: () s1 (1) s2 (1,2) s3 (1,2,3) r2 (1) r3 [3 is not valid at this point] The reader does not track the stack of savepoints, therefore it cannot detect errors in the restore order. It's the responsibility of the application to ensure the correct restore order. Failure in doing so will lead to an inconsistent state and may well corrupt the whole document. Besides the restore order savepoints are independent of each other. The application may delete/reuse a savepoint anytime it doesn't need it anymore. Savepoints cannot be used to transfer the state of a reader to a second reader. Any attempt in doing so will lead to undefined behaviour.

pReader [IN]

pSavePoint [OUT]

Will receive all information needed to restore the current state of the reader.

3.1.35 BTagReaderRestoreSavepoint

*int BTagReaderRestoreSavepoint (BINTAGREADER *pReader, BINTAGREADERSAVEPOINT *pSavepoint)*

Restore the state of the reader to one previously saved using BTagReaderCreateSavepoint. See there for a detailed description on how to use these functions.

pReader [IN]

Must be the same reader that the savepoint was created on.

pSavepoint [IN]

A previously created savepoint.

3.1.36 BTagReaderGetFirstTag

void BTagReaderGetFirstTag (BINTAGREADER *pReader, RTS_UI32 *pulToplevelTag, RTS_UI32 *pulTag, RTS_UI32 *pulSize, RTS_RESULT *pResult)*

Get the first tag out of a stream

pReader [IN]

Pointer to the reader stream

pulToplevelTag [OUT]

Returns the toplevel tag. -1, if no complex tag

pulTag [OUT]

Returns the tag

pulSize [OUT]

Size of the tag

pResult [OUT]

Pointer to error code

Result

Pointer to the content

3.1.37 BTagReaderGetNextTag

void BTagReaderGetNextTag (BINTAGREADER *pReader, RTS_UI32 *pulToplevelTag, RTS_UI32 *pulTag, RTS_UI32 *pulSize, RTS_RESULT *pResult)*

Get the first tag out of a stream

pReader [IN]

Pointer to the reader stream

pulToplevelTag [OUT]

Returns the toplevel tag. -1, if no complex tag

pulTag [OUT]

Returns the tag

pulSize [OUT]

Size of the tag

pResult [OUT]

Pointer to error code

Result

Pointer to the content

4 CmpBlkDrvUdp

A block driver for udp networks. It assumes that all node addresses are within the same subnet and thus uses only the last ip component for addressing. It is able to communicate with nodes listening on one of up to four ports, which allows for more then one runtime system running on a single host. Detected devices are named "ether 0" through "ether n" where n+1 is the number of detected network interfaces.

4.1 Tasks

4.1.1 Task: BlkDrvUdp

Category: Task
Priority: TASKPRIO_HIGH_END
Description: Block driver communication task.

4.2 CmpBlkDrvItf

Block driver interface. This interface could be implemented by different components.

4.2.1 Define: BLKDRVCOM_MAX_SER_READSIZE

Condition: #ifndef BLKDRVCOM_MAX_SER_READSIZE
Category: Static defines
Type:
Define: BLKDRVCOM_MAX_SER_READSIZE
Key: 1
Maximum number of bytes that are read with one call to SysComRead. Can be set to a bigger value on embedded systems, to minimize the number of calls to SysComRead. Example: 256. Only for BlkDrvCom.

4.2.2 Define: UDP_BIND_ADDRESS

Category: Static defines
Type:
Define: UDP_BIND_ADDRESS
Key: 0
Socket bind option for the CmpBlkDrvUdp.

4.2.3 Define: UDP_PACKET_SORT_NONE

Category: Static defines
Type:
Define: UDP_PACKET_SORT_NONE
Key: 0
Options for the CmpBlkDrvUdp to sort packages after receiving or before sending. Sorting for incoming and outgoing packages can be used seperately or in combination.

5 CmpChannelMgrEmbedded

This component manages layer 4 communication (splitting and reassembling of messages, retransmission, ...). To achieve its tasks it needs either a channel server component.

5.1 CmpChannelMgrItf

Interface for the channel manager.

5.1.1 Define: PKG_LOG_NONE

Category: Package logfilter
Type:
Define: PKG_LOG_NONE
Key: UINT32_C
Package info log entry filters for the CmpChannelMgr

5.1.2 NetworkGetStatus

*int NetworkGetStatus (CHANNELBUFFER *pChBuffer, RTS_UI16 *pusStatus, RTS_UI8 *pbyScalingFactor, RTS_I32 *pnItemsComplete, RTS_I32 *pnTotalItems)*

Get the current status of an active channel.

pChBuffer [IN]
The Channel for which to retrieve the status

pusStatus [OUT]
Is set to the current progress state. The PROGRESS_xxx constants define valied values.

pbyScalingFactor [OUT]
Provides the scaling factor for pnItemsComplete and pnTotalItems. These values have been scaled down by dividing them through 2^ScalingFactor (i.e. they have been right shifted by ScalingFactor bits).

pnItemsComplete [OUT]
Number of items completed (eg. the number of bytes transfered).

pnTotalItems [OUT]
Total number of item. Is set to -1 if unknown. *

5.1.3 NetworkGetChBufferSize

*RTS_UI32 NetworkGetChBufferSize (RTS_UI32 dwCommBufferSize, unsigned short wMaxBlockSize, int *pnNumBlocks)*

Given a fixed size for the communication buffer return the required size of the channel buffer.

dwCommBufferSize [IN]
Requested size of the communication buffer

wMaxBlockSize [IN]
Maximum size of a layer4 block (as returned by RouterGetMaxBlockSize)

pnNumBlocks [OUT]
Required number of block buffers

Result
The size of the channel buffer in bytes.

5.1.4 NetworkInitChannelBuffer

*RTS_UI32 NetworkInitChannelBuffer (CHANNELBUFFER *pChBuffer, RTS_HANDLE hRouter, RTS_UI32 dwBufferSize, unsigned short wChannelId, PEERADDRESS addrSender, unsigned char byChFlags, RTS_UI32 dwMaxCommBuffer)*

Initialize the provided channelbuffer.

pChBuffer [IN]

dwBufferSize [IN]
Size of pChBuffer.

wChannelId [IN]
The id of the channel.

addrSender [IN]
The 3S-address of this channels peer.

byFlags [IN]
Initial channel flags (CF_SERVERCHANNEL for a serverchannel, CF_SENDMODE for a clientchannel)

dwMaxCommBuffer [IN]
Maximum size of the communication buffer to use (usually the requested communication buffer)

Result
The size of the communication buffer for this channelbuffer.

5.1.5 NetworkInitChannelBuffer2

*RTS_UI32 NetworkInitChannelBuffer2 (CHANNELBUFFER *pChBuffer, RTS_HANDLE hRouter, RTS_UI32 dwBufferSize, unsigned short wChannelId, PEERADDRESS addrSender, unsigned char byChFlags, RTS_UI32 dwMaxCommBuffer, unsigned short wMaxBlockSize)*

Initialize the provided channelbuffer with a given maximum Blocksize.

pChBuffer [IN]

dwBufferSize [IN]
Size of pChBuffer.

wChannelId [IN]
The id of the channel.

addrSender [IN]
The 3S-address of this channels peer.

byFlags [IN]
Initial channel flags (CF_SERVERCHANNEL for a serverchannel, CF_SENDMODE for a clientchannel)

dwMaxCommBuffer [IN]
Maximum size of the communication buffer to use (usually the requested communication buffer)

wMaxBlockSize [IN]
Maximum size of one block

Result
The size of the communication buffer for this channelbuffer.

5.1.6 NetworkSendMetaPkg

*int NetworkSendMetaPkg (RTS_HANDLE hRouter, PEERADDRESS addrReceiver, L4METAPACKAGE *pMetaPkg, int nPkgSize)*

Send metaPkg to addrReceiver. Sets the checksum field of the metaPkg before sending.

addrReceiver [IN]
Address of the receiver

pMetaPkg [IN]
The package to be sent

nPkgSize [IN]
Size of pMetaPkg

Result
An error code.

5.1.7 NetworkSendMessage

*int NetworkSendMessage (CHANNELBUFFER *pChBuffer, PROTOCOL_DATA_UNIT pduData)*

Send a message over the provided channel. A channel can't be receiving and sending at the same time. Thus this function will fail with ERR_CHANNELMODE if the channel is currently in receive mode.

Result
- ERR_CHANNELMODE if the channel is in receive mode.
- ERR_CHC_MESSAGESIZE/ERR_CHS_MESSAGESIZE if the size of the data is greater then the CommBuffer size.

6 CmpChannelServer

A layer4 server component. This component is needed by the network component and by applications that want to act as a channel server.

6.1 CmpChannelServerItf

Interface for the channel server.

6.1.1 Define: EVT_ChSChannelClosed

Category: Events
Type:
Define: EVT_ChSChannelClosed
Key: MAKE_EVENTID
Event is sent when a channel is closed.

6.1.2 Typedef: EVTPARAM_ChannelClosed

Stuctname: EVTPARAM_ChannelClosed
Category: Event parameter
Typedef: typedef struct { RTS_UI32 ulChannelHandle; RTS_RESULT errReason; } EVTPARAM_ChannelClosed;

6.1.3 NetServerGetChannel

*int NetServerGetChannel (PEERADDRESS addrPeer, unsigned short wChannelId, CHANNELBUFFER **ppChBuffer)*

Get the buffer for the specified channel id. NOTE: After usage the channelbuffer MUST be released by calling NetServerReleaseChannel. Failing to do so will prevent the channel from being closed and the server will eventually run out of channels. Nevertheless, this function DOES NOT provide exclusive access to the channel. The L4Base component must use appropriate semaphores to ensure exclusive access.

addrPeer [IN]
The second endpoint of the channel

wChannelId [IN]
The id of the channel.

ppChBuffer [OUT]
Is set to the channelbuffer, if the channel exists.

6.1.4 NetServerReleaseChannel

*int NetServerReleaseChannel (CHANNELBUFFER *pChBuffer)*

Release a channel buffer returned by NetServerGetChannel. This buffer may not be used after calling this function. Use NetServerGetChannel to acquire access to this channel again.

pChBuffer [IN]
The channel buffer to release

6.1.5 NetServerMessageReceived

*int NetServerMessageReceived (CHANNELBUFFER *pChBuffer, PROTOCOL_DATA_UNIT pduData)*

Called by the L4Base component whenever a complete L7 message arrived on a server channel

pChBuffer [IN]
Pointer to the channel buffer

pduData [IN]
Content of the message. The server must copy the contents of the message before sending the next message, since the data pointed to by pData will be valid only until the next message has been sent or the channel is closed.

Result
error code

6.1.6 NetServerMessageReceived2

*int NetServerMessageReceived2 (RTS_HANDLE hRouter, CHANNELBUFFER *pChBuffer, PROTOCOL_DATA_UNIT pduData)*

Obsolete: Use NetServerMessageReceived instead. Will be removed in future versions!

Called by the L4Base component whenever a complete L7 message arrived on a server channel

hRouter [IN]
Obsolete parameter, should be set to RTS_INVALID_HANDLE.

pChBuffer [IN]
Pointer to the channel buffer

pduData [IN]
Content of the message. The server must copy the contents of the message before sending the next message, since the data pointed to by pData will be valid only until the next message has been sent or the channel is closed.

Result
error code

7 CmpChecksum

Provides functions to calculate CRC-Checksums (CRC-16, CRC-32) etc.

7.1 Define: INIT_VAL32

Condition: #ifndef UINT32_C
Category:
Type:
Define: INIT_VAL32
Key:
Initial CRC32 value

7.2 CmpChecksumItf

Interface for the checksum utility component.

7.2.1 Typedef: crc16finish_struct

Stuctname: crc16finish_struct

crc16finish

Typedef: typedef struct tagcrc16finish_struct { RTS_IEC_WORD ulCRC; VAR_INPUT RTS_IEC_WORD CRC16Finish; VAR_OUTPUT } crc16finish_struct;

7.2.2 Typedef: crc32update2_struct

Stuctname: crc32update2_struct

crc32update2

Typedef: typedef struct tagcrc32update2_struct { RTS_IEC_DWORD ulCRC; VAR_INPUT RTS_IEC_BYTE *pData; VAR_INPUT RTS_IEC_DWORD ulSize; VAR_INPUT RTS_IEC_DWORD CRC32Update2; VAR_OUTPUT } crc32update2_struct;

7.2.3 Typedef: crc16init_struct

Stuctname: crc16init_struct

crc16init

Typedef: typedef struct tagcrc16init_struct { RTS_IEC_WORD CRC16Init; VAR_OUTPUT } crc16init_struct;

7.2.4 Typedef: crc16update_struct

Stuctname: crc16update_struct

crc16update

Typedef: typedef struct tagcrc16update_struct { RTS_IEC_WORD ulCRC; VAR_INPUT RTS_IEC_BYTE *pData; VAR_INPUT RTS_IEC_DWORD ulSize; VAR_INPUT RTS_IEC_WORD CRC16Update; VAR_OUTPUT } crc16update_struct;

7.2.5 Typedef: crc32update_struct

Stuctname: crc32update_struct

crc32update

Typedef: typedef struct tagcrc32update_struct { RTS_IEC_DWORD ulCRC; VAR_INPUT RTS_IEC_BYTE *pData; VAR_INPUT RTS_IEC_DWORD ulSize; VAR_INPUT RTS_IEC_DWORD CRC32Update; VAR_OUTPUT } crc32update_struct;

7.2.6 Typedef: crc32init_struct

Stuctname: crc32init_struct

crc32init

Typedef: typedef struct tagcrc32init_struct { RTS_IEC_DWORD CRC32Init; VAR_OUTPUT } crc32init_struct;

7.2.7 Typedef: crc32finish_struct

Stuctname: crc32finish_struct

crc32finish

Typedef: typedef struct tagcrc32finish_struct { RTS_IEC_DWORD ulCRC; VAR_INPUT RTS_IEC_DWORD CRC32Finish; VAR_OUTPUT } crc32finish_struct;

7.2.8 CRC16Init

unsigned short CRC16Init (void)

Implementation of a 16-Bit CRC. The CRC is based on the CCITT polynom x^16 + x^12 + x^5 + 1. Since there seem to be different opinions about "the" CCITT CRC-16, here a description of the options used in this api: - Bits are shifted in with MSB first - Input bytes are NOT reversed - The final CRC is NOT reversed - Initial value is 0xFFFF - 16 Zero-bits are implicitly appended to the end of the message - The "checkvalue" is 0xE5CC (ie. the CRC of the ASCII string '123456789')

7.2.9 CRC16Update

*unsigned short CRC16Update (unsigned short usCRC, const unsigned char * pData, RTS_SIZE ulSize)*

Update the CRC with a block of data. Before the first call you have to initialize the CRC by calling CRC16Init.

usCRC [IN]
The previous value of the crc as returned by the last call to CRC16Init or CRC16Update.

pData [IN]
Points at the data which should be added to the crc.

ulSize [IN]
The number of bytes in pData.

Result
Returns the updated crc.

7.2.10 CRC16Finish

unsigned short CRC16Finish (unsigned short usCRC)

For the CRC to be valid it has to be finished after it is updated with all data. Use this as the last step during calculating your CRC.

usCRC [IN]
The current value of the crc to be finished.

Result
The checksum over all data passed in via CRC16Update.

7.2.11 CRC32Init

RTS_UI32 CRC32Init (void)

Implementation of a initial 32-Bit CRC. Initial value is 0xFFFFFFFF.

Result
Returns the initial CRC value.

7.2.12 CRC32Update

*RTS_UI32 CRC32Update (RTS_UI32 ulCRC, const unsigned char *pData, RTS_SIZE ulSize)*

Obsolete: Use CRC32Update2 instead!

Update the CRC with a block of data. Before the first call you have to initialize the crc by calling CRC32Init. Must only be used, if the CRC is calculated in one step and not using several calls of CRC32Update for adding data to the CRC.

ATTENTION: You have to finish the CRC with the function CRC32Finish()!

ulCRC [IN]
The previous value of the crc as returned by the last call to CRC32Init.

pData [IN]
Points at the data for which the CRC should be calculated.

ulSize [IN]
The number of bytes in pData.

Result
Returns the updated CRC.

7.2.13 CRC32Finish

RTS_UI32 CRC32Finish (RTS_UI32 ulCRC)

Obsolete: Use CRC32Finish2 instead!

For the CRC to be valid it has to be finished after it is updated with all data. Use this as the last step during calculating your CRC.

In opposite to the other CRC algorithms, the resulting CRC is swapped to IntelByteOrder.

ulCRC [IN]
The current value of the CRC to be finished.

Result
The checksum over all data passed in via CRC32Update.

7.2.14 CRC32Update2

*RTS_UI32 CRC32Update2 (RTS_UI32 ulCRC, const unsigned char *pData, RTS_SIZE ulSize)*

Update the CRC with a block of data. Before the first call you have to initialize the CRC by calling CRC32Init. ATTENTION: You have to finish the CRC with the function CRC32Finish2()!

ulCRC [IN]
The previous value of the CRC as returned by the last call to CRC32Init or CRC32Update2.

pData [IN]
Points at the data which should be added to the CRC.

ulSize [IN]
The number of bytes in pData.

Result
Returns the updated CRC.

7.2.15 CRC32Finish2

RTS_UI32 CRC32Finish2 (RTS_UI32 ulCRC)

For the CRC to be valid it has to be finished after it is updated with all data. Use this as the last step during calculating your CRC.

ulCRC [IN]
The current value of the CRC to be finished.

Result
The checksum over all data passed in via CRC32Update.

8 CmpCommunicationLib

Provides basic functions and definitions for the communication system.

8.1 CmpCommunicationLibItf

Interface for the communication library.

8.1.1 AddrEquals

int AddrEquals (NODEADDRESS addr1, NODEADDRESS addr2)

Result

Returns TRUE if addr1 == addr2, FALSE else

8.1.2 AddrEqualsBuffer

*int AddrEqualsBuffer (NODEADDRESS addr1, ADDRESSCOMPONENT *pBufAddr2, RTS_UI32 nLenAddr2)*

Result

Returns TRUE, if addr1 equals the address described by pBufAddr2 and nlenAddr2

9 CmpDevice

9.1 CmpDeviceItf

Interface for the device component. This is the first component, that contacts a client to get online access to the target. Here for example the target identification is checked.

9.1.1 Define: SRV_DEV_GET_TARGET_IDENT

9.1.2 Define: TAG_DEV_REPLY_IDENTIFICATION

9.1.3 Define: SETTINGS_FLAG_KILLTASKONRESET

Category: Settings flags
Type:
Define: SETTINGS_FLAG_KILLTASKONRESET
Key: 0x01
Setings flags that re returned by the runtime system

9.1.4 Define: USERDB_OBJECT_DEVICE

Category: Static defines
Type:
Define: USERDB_OBJECT_DEVICE
Key: Device
Predefined objects in the runtime

9.1.5 Typedef: TargetIdent

Stuctname: TargetIdent
Category: Target identification

These values identifies a target completely unique!

Typedef: typedef struct { RTS_UI32 ulTargetType; RTS_UI32 ulTargetId; RTS_UI32 ulTargetVersion; } TargetIdent;

9.1.6 Typedef: DeviceIdentification

Stuctname: DeviceIdentification
Category: Device description

Typedef: typedef struct { TargetIdent targetIdent; RTS_WCHAR *pwszDeviceName; unsigned int nMaxDeviceNameLen; RTS_WCHAR *pwszVendorName; unsigned int nMaxVendorNameLen; RTS_WCHAR *pwszNodeName; unsigned int nMaxNodeNameLen; } DeviceIdentification;

9.1.7 DevGetIdent

*RTS_RESULT DevGetIdent (DeviceIdentification *pDevIdent)*

Retrieves the target identification

pDevIdent [OUT]
Pointer to identification. Is filled by the function. If string pointer are NULL, the real size of the strings is returned.

Result
error code

9.1.8 DevCheckTargetIdent

*RTS_RESULT DevCheckTargetIdent (TargetIdent *pTargetIdentReq, TargetIdent *pTargetIdent)*

Checks the compatibility between a requested identification and the target identification

pTargetIdentReq [IN]
Requested target identification to check

pTargetIdent [IN]
Own target identity. Can be NULL, then the built in target identification is used

Result
error code

10 CmpEventMgr

Manager for all kind of events int the runtime system. Event manager call registered callback functions on the events

10.1 CmpEventMgrItf

This is the interface of the event manager. The manager is responsible to handle all events in the runtime system and to call special registered functions (callbacks) if an event occurred. An event can be sent in any situation, when a state will be changed in the runtime system. An event can be e.g. stop of an Iec application, download of an Iec application, exception occurred in a component, etc.

Typically an event will be sent before a state changed (xxxPrepare) and if the state has changed (xxxDone)

The component that provides and sends an event is called the provider. The component that receives the event is called the consumer of an event.

Each component can define its own event ID. Such an event ID consists of two numbers:

- Event class: 16 bit number that specifies the class of the event
- Event: 16 bit number that specifies the event

The event class and the event is matched to one 32 bit number that is called the event ID. The event, event class and the component ID of the provider makes an event unique. So every provider has to specify at least these three things to sent an event.

Each provider can specify additional parameters for each event that is transferred to the consumer. This is event specific and can be specified by the consumer. Such an optional event parameter must be specified by a component specific parameter ID and with a parameter version number. With this information, a consumer can check, which parameter in which version the provider is sending.

To use an event, first of all the provider has to register the event. After that, the consumer can open this event and can attach its callback routine to this event. Such a callback routine can be:

- C-Function
- Iec-Function
- Iec-Method of a function block
- C++-Method of a C++ class

IMPLEMENTATION NOTE: A provider typically registers its event in the CH_INIT2 hook. The consumer typically registers its callback to special events in the CH_INIT3 hook.

If a provider only wants to register an event if it is really needed by a consumer, the CmpEventMgr sents a special event, if a consumer tries to open an event (see EVT_EventOpen). In this event, the provider can register the event and the consumer can open a valid event.

In opposite, if a consumer wants to register a callback on an event, an event is sent if a provider registers its event (see EVT_EventCreate).

If an event is unregistred by a provider, the event EVT_EventDelete is sent. If an event is closed by a consumer, the event EVT_EventClose is sent.

10.1.1 Define: EVENTMGR_NUM_OF_STATIC_EVENTS

Condition: #ifndef EVENTMGR_NUM_OF_STATIC_EVENTS
Category: Static defines
Type:
Define: EVENTMGR_NUM_OF_STATIC_EVENTS
Key: 50
Maximum number of static allocated events

10.1.2 Define: EVENTMGR_NUM_OF_STATIC_CALLBACKS

Condition: #ifndef EVENTMGR_NUM_OF_STATIC_CALLBACKS
Category: Static defines
Type:
Define: EVENTMGR_NUM_OF_STATIC_CALLBACKS
Key: 10
Maximum number of static allocated callback routines to be registered

10.1.3 Define: EVENTMGR_NUM_OF_STATIC_IECCALLBACKS

Condition: #ifndef EVENTMGR_NUM_OF_STATIC_IECCALLBACKS
Category: Static defines
Type:

Define: EVENTMGR_NUM_OF_STATIC_IECCALLBACKS
Key: 5
Maximum number of static allocated iec callback routines to be registered

10.1.4 Define: MAKE_EVENTID

Condition: #ifndef Event
Category: Convert macro
Type:
Define: MAKE_EVENTID
Key: Class
Macro to create an event ID with the event class and the event

10.1.5 Define: EVTPROVIDER_NONE

Category: Provider component ids
Type:
Define: EVTPROVIDER_NONE
Key: 0
Special priovider ids

10.1.6 Define: EVTCLASS_NONE

Category: Event classes
Type:
Define: EVTCLASS_NONE
Key: 0

All possible event classes:

- EVTCLASS_NONE: No class or invalid
- EVTCLASS_ALL: All classes
- EVTCLASS_INFO: Information
- EVTCLASS_WARNING: Warning
- EVTCLASS_ERROR: Error
- EVTCLASS_EXCEPTION: Exception
- EVTCLASS_VENDOR_SPEC: Vendor specific. Can be used for own event classes

10.1.7 Define: EVT_NONE

Category: Special Events
Type:
Define: EVT_NONE
Key: 0

- EVT_NONE: No event or invalid
- EVT_ALL: All events

10.1.8 Define: EVENT_CALLBACKS_NO_LIMIT

Category: Callback limit
Type:
Define: EVENT_CALLBACKS_NO_LIMIT
Key: UINT32_MAX
No limit of callbacks possible per event

10.1.9 Define: EVT_EventCreate

Category: Events
Type:
Define: EVT_EventCreate
Key: MAKE_EVENTID
Event is sent after a provider creates a new event

10.1.10 Define: EVT_EventDelete

Category: Events
Type:
Define: EVT_EventDelete
Key: MAKE_EVENTID
Event is sent before a provider deletes an event

10.1.11 Define: EVT_EventOpen

Category: Events
Type:

Define: EVT_EventOpen
Key: MAKE_EVENTID
Event is sent before a consumer tries to open an event

10.1.12 Define: EVT_EventClose

Category: Events
Type:
Define: EVT_EventClose
Key: MAKE_EVENTID
Event is sent before a consumer closes an event

10.1.13 Define: EVT_EventCommCycle

Category: Events
Type:
Define: EVT_EventCommCycle
Key: MAKE_EVENTID
Event is sent in every call of the CH_COMM_CYCLE. Can be used for IEC background jobs.

10.1.14 Typedef: EVTPARAM_CmpEventMgr

Stuctname: EVTPARAM_CmpEventMgr
Category: Event parameter
Typedef: typedef struct { EVENTID EventId; CMPID CmpIdProvider; } EVTPARAM_CmpEventMgr;

10.1.15 EventCreate

*RTS_HANDLE EventCreate (EVENTID EventId, CMPID CmpIdProvider, RTS_RESULT *pResult)*

Creates a new event object. If event still exists, a handle to this object will be returned. An event is typically created by the provider of an event in the CH_INIT_DONE hook.

EventId [IN]
Event ID of the event. Contains the class and the event

CmpIdProvider [IN]
Component ID of the provider

pResult [OUT]
Pointer to the error code

Result
Handle to the event object

10.1.16 EventCreate2

*RTS_HANDLE EventCreate2 (EVENTID EventId, CMPID CmpIdProvider, RTS_UI32 nCallbacksPossible, RTS_RESULT *pResult)*

Creates a new event object. If event still exists, a handle to this object will be returned. An event is typically created by the provider of an event in the CH_INIT_DONE hook.

EventId [IN]
Event ID of the event. Contains the class and the event

CmpIdProvider [IN]
Component ID of the provider

nCallbacksPossible [IN]
Maximum number of callbacks possible on this event or EVENT_CALLBACKS_NO_LIMIT for no limit

pResult [OUT]
Pointer to the error code

Result
Handle to the event object

10.1.17 EventDelete

RTS_RESULT EventDelete (RTS_HANDLE hEvent)

Deletes an event specified by handle

hEvent [IN]
Handle to event

Result
error code

10.1.18 EventDeleteAll

RTS_RESULT EventDeleteAll (void)

Delete all registered events and the registered callbacks

Result

error code

10.1.19 EventOpen

*RTS_HANDLE EventOpen (EVENTID EventId, CMPID CmpIdProvider, RTS_RESULT *pResult)*

Opens an existing event object. Can be used to check, if the event was created by the provider. If the event does not exist, an error code is returned.

Typically an event is opened by the consumer of an event in the CH_INIT_DONE2 hook.

EventId [IN]

Event ID of the event. Contains the class and the event

CmpIdProvider [IN]

Component ID of the provider

pResult [OUT]

Pointer to error code

Result

Handle to the event object

10.1.20 EventClose

RTS_RESULT EventClose (RTS_HANDLE hEvent)

Close an event specified by handle

hEvent [IN]

Handle to the event

Result

error code

10.1.21 EventGetEvent

unsigned short EventGetEvent (EVENTID EventId)

Extract the event from eventid

EventId [IN]

Event ID

Result

Event. Is specified in the interface of each component

10.1.22 EventGetClass

unsigned short EventGetClass (EVENTID EventId)

Extract the event class from eventid

EventId [IN]

Event ID

Result

Event class

10.1.23 EventRegisterCallback

*RTS_RESULT EventRegisterCallback (RTS_HANDLE hEvent, ICmpEventCallback *pICallback)*

Register an interface callback function to an event. The interface can be from a C object, a C++ class or a wrapper class for an Iec function block

hEvent [IN]

Handle to event

pICallback [IN]

Pointer to callback interface

Result

error code

10.1.24 EventRegisterCallback2

*RTS_RESULT EventRegisterCallback2 (RTS_HANDLE hEvent, ICmpEventCallback *pICallback, void *pUserParameter)*

Register an interface callback function to an event. The interface can be from a C object, a C++ class or a wrapper class for an Iec function block

hEvent [IN]
Handle to event

pICallback [IN]
Pointer to callback interface

pUserParameter [IN]
Pointer to user parameter, that is transmitted to the callback (see EventParam)

Result
error code

10.1.25 EventRegisterCallback3

*RTS_RESULT EventRegisterCallback3 (RTS_HANDLE hEvent, ICmpEventCallback *pICallback, int bIec, void *pUserParameter)*

Register an interface callback function to an event. The interface can be from a C object, a C++ class or a wrapper class for an Iec function block

hEvent [IN]
Handle to event

pICallback [IN]
Pointer to callback interface

bIec [IN]
1=Iec interface behind the C interface, 0=C interface

pUserParameter [IN]
Pointer to user parameter, that is transmitted to the callback (see EventParam)

Result
error code

10.1.26 EventUnregisterCallback

*RTS_RESULT EventUnregisterCallback (RTS_HANDLE hEvent, ICmpEventCallback *pICallback)*

Unregister a callback interface from an event specified by handle and callback interface

hEvent [IN]
Handle to event

pICallback [IN]
Pointer to callback interface

Result
error code

10.1.27 EventRegisterCallbackFunction

RTS_RESULT EventRegisterCallbackFunction (RTS_HANDLE hEvent, PFEVENTCALLBACKFUNCTION pfCallbackFunction, int bIec)

Register a callback function to an event. Callback function can be a C or Iec function

hEvent [IN]
Handle to event

pfCallbackFunction [IN]
Pointer to callback function

bIec [IN]
1=Iec function, 0=C function

Result
error code

10.1.28 EventRegisterCallbackFunction2

*RTS_RESULT EventRegisterCallbackFunction2 (RTS_HANDLE hEvent, PFEVENTCALLBACKFUNCTION pfCallbackFunction, int bIec, void *pUserParameter)*

Register a callback function to an event. Callback function can be a C or Iec function

hEvent [IN]
Handle to event

pfCallbackFunction [IN]
Pointer to callback function
blec [IN]
1=Iec function, 0=C function
pUserParameter [IN]
Pointer to user parameter, that is transmitted to the callback (see EventParam)
Result
error code

10.1.29 EventUnregisterCallbackFunction

RTS_RESULT EventUnregisterCallbackFunction (RTS_HANDLE hEvent, PFEVENTCALLBACKFUNCTION pfCallbackFunction)
Unregister a callback function from an event specified by handle and callback
hEvent [IN]
Handle to event
pfCallbackFunction [IN]
Pointer to callback function
Result
Error code

10.1.30 EventRegisteredCallbacks

*unsigned long EventRegisteredCallbacks (RTS_HANDLE hEvent, RTS_RESULT *pResult)*
Unregister a callback function from an event specified by handle and callback
hEvent [IN]
Handle to event
pResult [OUT]
Pointer to error code
Result
Number of regis

10.1.31 EventPost

*RTS_RESULT EventPost (RTS_HANDLE hEvent, EventParam *pEventParam)*
Post or sent an event
hEvent [IN]
Handle to event
pEventParam [IN]
Pointer to the event parameters
Result
error code

10.1.32 EventPost2

RTS_RESULT EventPost2 (RTS_HANDLE hEvent, unsigned short usParamId, unsigned short usVersion, void pParameter)*
Sent an event and call synchronously all registered callback handler
hEvent [IN]
Handle to event
usParamId [IN]
Id of the parameter
usVersion [IN]
Version of the parameter
pParameter [IN]
Pointer to the event specific parameter, that is specified by Id
Result
error code

10.1.33 EventPostByEvent

*RTS_RESULT EventPostByEvent (EVENTID EventId, CMPID CmpIdProvider, EventParam *pEventParam)*

Post an event direct without opening the event

EventId [IN]
Event ID of the event. Contains the class and the event

CmpIdProvider [IN]
Component ID of the provider

pEventParam [IN]
Pointer to the event parameters

Result
error code

11 CmplecTask

Component for IEC task management

11.1 CmplecTaskItf

The component CmplecTask provides an interface to create an handle all task of one or more IEC applications.

The following drawing describes the dependencies between the structures of this and other components in this context:

+-----------+ 1 1 +-----------+ 1 n +---------+ | Task_Desc | ----- | Task_Info | ----- | SlotPOU | +-----------+ +-----------+ +-----

Depending on the Scheduler, the task might furthermore be mapped to a hardware ressource or another operating system object.

CmpSchedule (Multitasking):

+-----------+ 1 1 +---------------+ 1 1 +----------------+ | SchedTask | ----- | SYS_TASK_INFO | ----- | OS Task Handle | +----

CmpScheduleTimer:

+-----------+ 1 1 +---------------+ 1 1 +----------------+ | SchedTask | ----- | SYS_TIMER_INFO | ----- | HW Timer Handle |

CmpScheduleEmbedded:

+-----------+ | SchedTask | +-----------+

The Task_Info structure is created and allocated in the IEC data area of the application, while everything else is allocated in the runtime.

Beside the handling of tasks, this component manages also the slots for the tasks, that might be registered by C or IEC code. Those slots are stored within a memory pool, from which they are called at some specific points before and after the IEC task cycle code.

For SIL2 certified systems, those slots are denied, because they potentially cause a call to an unsafe context from the safe context of the IEC task. So we allow those calls only in debug mode.

11.1.1 Define: EVT_AfterReadingInputs

Category: Events
Type:
Define: EVT_AfterReadingInputs
Key: MAKE_EVENTID
Event is sent after reading inputs

11.1.2 Define: EVT_BeforeWritingOutputs

Category: Events
Type:
Define: EVT_BeforeWritingOutputs
Key: MAKE_EVENTID
Event is sent before writing outputs

11.1.3 Define: EVT_BeforeReadingInputs

Category: Events
Type:
Define: EVT_BeforeReadingInputs
Key: MAKE_EVENTID
Event is sent before reading inputs

11.1.4 Define: EVT_AfterWritingOutputs

Category: Events
Type:
Define: EVT_AfterWritingOutputs
Key: MAKE_EVENTID
Event is sent after writing outputs

11.1.5 Define: EVT_IecTask_ReloadInit

Category: Events
Type:
Define: EVT_IecTask_ReloadInit
Key: MAKE_EVENTID

Event is sent at reload of an IOEC task right after the task is deleted

11.1.6 Define: EVT_IecTaskCreateDone

Category: Events
Type:
Define: EVT_IecTaskCreateDone
Key: MAKE_EVENTID
Event is sent, after an IEC-task was created at application download

11.1.7 Define: EVT_IecTaskPrepareDelete

Category: Events
Type:
Define: EVT_IecTaskPrepareDelete
Key: MAKE_EVENTID
Event is sent, before an IEC-task will be deleted (e.g. delete of the application)

11.1.8 Define: EVT_IecTaskDebugLoop

Category: Events
Type:
Define: EVT_IecTaskDebugLoop
Key: MAKE_EVENTID
Event is sent cyclically in the debug loop, if the IEC task is halted on a breakpoint

11.1.9 Define: TaskCyclic

Category: IEC task types
Type:
Define: TaskCyclic
Key: 0x0000

11.1.10 Define: TS_WATCHDOG_ENABLE

Category: Task status definitions
Type:
Define: TS_WATCHDOG_ENABLE
Key: 0x0040

11.1.11 Define: TF_WATCHDOG_ENABLE

Category: Task status bits
Type:
Define: TF_WATCHDOG_ENABLE
Key: 6

11.1.12 Define: ITF_VISU_TASK

11.1.13 Define: IECTASK_TASK_INFO_VERSION

Category: Task Info
Type:
Define: IECTASK_TASK_INFO_VERSION
Key: 2

Currently supported version number of the task info strcuture.

11.1.14 Define: IECTASK_TASK_MAX_PRIO

Category: Task Info
Type:
Define: IECTASK_TASK_MAX_PRIO
Key: 255

Currently supported max number of task priority.

11.1.15 Typedef: EVTPARAM_CmpIecTask

Stuctname: EVTPARAM_CmpIecTask
Category: Event parameter
Typedef: typedef struct { struct tagTask_Desc *pTaskDesc; } EVTPARAM_CmpIecTask;

11.1.16 Typedef: Task_Desc

Stuctname: Task_Desc

Local Task Description

Typedef: typedef struct tagTask_Desc { int iId; int bIgnoreWatchdogInCycle; RTS_HANDLE hSlotPOUPool; RTS_HANDLE hIecTask; RTS_HANDLE hSched; APPLICATION* pAppl; struct tagTask_Info *pInfo; RegContext Context; RTS_SYSTIME tCycleStart; RTS_SYSTIME tAccumulatedCycleTime; RTS_UI32 ulStackSize; unsigned char StaticSlotPool[MEM_GET_STATIC_LEN(NUM_OF_STATIC_IEC_SLOTS, SlotPOU)]; void *pCppInstance; RTS_UI32 ulWatchdogCycleCount; RTS_UI32 ulFlags; int iWatchdogHitCount; #ifdef RTS_ENABLE_TASK_TRACE RTS_HANDLE hTracePacket; RTS_HANDLE hTraceRecord; #endif } Task_Desc;

11.1.17 Typedef: LDATAREF_TYPE

Stuctname: LDATAREF_TYPE

LDATAREF_TYPE

Typedef: typedef struct tagLDATAREF_TYPE { RTS_IEC_UINT POURef; RTS_IEC_UDINT Offset; RTS_IEC_UDINT Size; } LDATAREF_TYPE;

11.1.18 Typedef: SYSIECTASKCONFENTRY

Stuctname: SYSIECTASKCONFENTRY

SYSIECTASKCONFENTRY

Typedef: typedef struct tagSYSIECTASKCONFENTRY { RTS_IEC_USINT byTaskNr; RTS_IEC_USINT byPriority; RTS_IEC_DINT lInterval; LDATAREF_TYPE ldrEvent; RTS_IEC_UINT wIndex; RTS_IEC_UDINT ulNameLen; RTS_IEC_STRING szName[81]; } SYSIECTASKCONFENTRY;

11.1.19 Typedef: SYSIECTASKINFO

Stuctname: SYSIECTASKINFO

SYSIECTASKINFO

Typedef: typedef struct tagSYSIECTASKINFO { RTS_IEC_DWORD dwCount; RTS_IEC_DWORD dwCycleTime; RTS_IEC_DWORD dwCycleTimeMin; RTS_IEC_DWORD dwCycleTimeMax; RTS_IEC_DWORD dwCycleTimeAvg; RTS_IEC_WORD wStatus; RTS_IEC_WORD wMode; } SYSIECTASKINFO;

11.1.20 Typedef: iectaskgetconfig_struct

Stuctname: iectaskgetconfig_struct

This function return the iec task configuration

Typedef: typedef struct tagiectaskgetconfig_struct { RTS_IEC_UDINT udiTaskId; VAR_INPUT SYSIECTASKCONFENTRY *pTaskConfig; VAR_INPUT RTS_IEC_UDINT IecTaskGetConfig; VAR_OUTPUT } iectaskgetconfig_struct;

11.1.21 Typedef: iectaskgetinfo_struct

Stuctname: iectaskgetinfo_struct

iectaskgetinfo

Typedef: typedef struct tagiectaskgetinfo_struct { RTS_IEC_STRING *stTaskName; VAR_INPUT SYSIECTASKINFO *pTaskInfo; VAR_INPUT RTS_IEC_UDINT IecTaskGetInfo; VAR_OUTPUT } iectaskgetinfo_struct;

11.1.22 Typedef: Jitter_Distribution

Stuctname: Jitter_Distribution

Jitter_Distribution

Typedef: typedef struct tagJitter_Distribution { RTS_IEC_WORD wRangeMax; RTS_IEC_WORD wCountJitterNeg; RTS_IEC_WORD wCountJitterPos; } Jitter_Distribution;

11.1.23 Typedef: iectaskgetprofiling_struct

Stuctname: iectaskgetprofiling_struct

iectaskgetprofiling

Typedef: typedef struct tagiectaskgetprofiling_struct { RTS_IEC_BOOL IecTaskGetProfiling; VAR_OUTPUT } iectaskgetprofiling_struct;

11.1.24 Typedef: iectaskgetcurrent_struct

Stuctname: iectaskgetcurrent_struct

iectaskgetcurrent

Typedef: typedef struct tagiectaskgetcurrent_struct { RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IecTaskGetCurrent; VAR_OUTPUT } iectaskgetcurrent_struct;

11.1.25 Typedef: iectaskenablewatchdog_struct

Stuctname: iectaskenablewatchdog_struct

iectaskenablewatchdog

Typedef: typedef struct tagiectaskenablewatchdog_struct { RTS_IEC_BYTE *hIecTask; VAR_INPUT RTS_IEC_UDINT IecTaskEnableWatchdog; VAR_OUTPUT } iectaskenablewatchdog_struct;

11.1.26 Typedef: iectaskgetinfo2_struct

Stuctname: iectaskgetinfo2_struct

iectaskgetinfo2

Typedef: typedef struct tagiectaskgetinfo2_struct { RTS_IEC_BYTE *hIecTask; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT Task_Info *IecTaskGetInfo2; VAR_OUTPUT } iectaskgetinfo2_struct;

11.1.27 Typedef: iectaskgetnext_struct

Stuctname: iectaskgetnext_struct

iectaskgetnext

Typedef: typedef struct tagiectaskgetnext_struct { RTS_IEC_STRING *pszAppName; VAR_INPUT RTS_IEC_BYTE *hPrevIecTask; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IecTaskGetNext; VAR_OUTPUT } iectaskgetnext_struct;

11.1.28 Typedef: iectaskgetfirst_struct

Stuctname: iectaskgetfirst_struct

iectaskgetfirst

Typedef: typedef struct tagiectaskgetfirst_struct { RTS_IEC_STRING *pszAppName; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IecTaskGetFirst; VAR_OUTPUT } iectaskgetfirst_struct;

11.1.29 Typedef: iectaskreload_struct

Stuctname: iectaskreload_struct

Reload a specified IEC task. Reload means here: Delete the task at the actual position and create it newly.

Typedef: typedef struct tagiectaskreload_struct { RTS_IEC_BYTE *hIecTask; VAR_INPUT RTS_IEC_UDINT udiTimeoutMs; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IecTaskReload; VAR_OUTPUT } iectaskreload_struct;

11.1.30 Typedef: iectaskdisablewatchdog_struct

Stuctname: iectaskdisablewatchdog_struct

iectaskdisablewatchdog

Typedef: typedef struct tagiectaskdisablewatchdog_struct { RTS_IEC_BYTE *hIecTask; VAR_INPUT RTS_IEC_UDINT IecTaskDisableWatchdog; VAR_OUTPUT } iectaskdisablewatchdog_struct;

11.1.31 Typedef: iectaskenablescheduling_struct

Stuctname: iectaskenablescheduling_struct

iectaskenablescheduling

Typedef: typedef struct tagiectaskenablescheduling_struct { RTS_IEC_HANDLE hIecTask; VAR_INPUT RTS_IEC_RESULT IecTaskEnableScheduling; VAR_OUTPUT } iectaskenablescheduling_struct;

11.1.32 Typedef: iectaskdisablescheduling_struct

Stuctname: iectaskdisablescheduling_struct

iectaskdisablescheduling

Typedef: typedef struct tagiectaskdisablescheduling_struct { RTS_IEC_HANDLE hIecTask; VAR_INPUT RTS_IEC_RESULT IecTaskDisableScheduling; VAR_OUTPUT } iectaskdisablescheduling_struct;

11.1.33 Typedef: iectaskresetstatistics_struct

Stuctname: iectaskresetstatistics_struct

iectaskresetstatistics

Typedef: typedef struct tagiectaskresetstatistics_struct { RTS_IEC_HANDLE hIecTask; VAR_INPUT RTS_IEC_RESULT IecTaskResetStatistics; VAR_OUTPUT } iectaskresetstatistics_struct;

11.1.34 __sys__setup__tasks

void __sys__setup__tasks (sys_setup_tasks_struct p)*

External function is called by internal plc code to setup a plc task. If a task can not be created, an exception is thrown.

p [IN]
Pointer to task configuration entry. Is an implicit generated plc data structure.

Result
no result

11.1.35 __sys__register__slot__pou

void __sys__register__slot__pou (sys_register_slot_pou_struct p)*

Register an IEC function function to a specific slot.

Slots are called in numbered order at specific positions in the task cycle

On SIL2 Runtimes, this call is only allowed in safety mode.

p [IN]
IEC function call parameters.

Result
Error code

11.1.36 __sys__unregister__slot__pou

void __sys__unregister__slot__pou (sys_register_slot_pou_struct p)*

Unregister an IEC function function from a specific slot.

On SIL2 Runtimes, this call is only allowed in safety mode.

p [IN]
IEC function call parameters.

Result
Error code

11.1.37 __sys__rts__cycle

void __sys__rts__cycle (sys_rts_cycle_struct p)*

This function is obsolete and may not be used in SIL2 Runtime! (SIL2 Note: An Exception is generated if function is called in SIL2 Runtime!)

If supported for backward compatibility, it executes the callbacks from a specific slot.

p [IN]
IEC function call parameters.

Result
Error code

11.1.38 __sys__rts__cycle__2

void __sys__rts__cycle__2 (sys_rts_cycle_2_struct p)*

This function executes a specific range of registered slots.

p [IN]
IEC function call parameters.

Result
nothing

11.1.39 IecTaskCreate

*RTS_HANDLE IecTaskCreate (APPLICATION *pApp, Task_Info *pTaskInfo, RTS_RESULT *pResult)*

Create a new IEC Task

IEC Tasks itself are used by the scheduler of the runtime. They don't essentially need a corresponding OS task or timer. They might be handled by the scheduler in a completely different way.

For example: The embedded scheduler calls the task code directly in the comm-cycle.

The Task registers itself at the scheduler, by calling the function SchedAddTask().

When the define RTS_COMPACT is set, no semaphores are used.

When the define RTS_SIL2 is set, no dynamic memory allocation is used.

pApp [IN]
Pointer to application that contains the task
pTaskInfo [IN]
Pointer to task information
pResult [OUT]
Pointer to error code
Result
Handle to newly created task

11.1.40 IecTaskGetDesc

*Task_Desc * IecTaskGetDesc (RTS_HANDLE hIecTask)*

Return task description for a task, specified by it's task handle.

hIecTask [IN]
Handle to task
Result
Task Description or NULL if task handle was NULL or invalid

11.1.41 IecTaskDebugLoop

RTS_RESULT IecTaskDebugLoop (RTS_HANDLE hIecTask)
Is called in debug loop, when IEC task is halted on breakpoint
hIecTask [IN]
Handle to IEC task
Result
ERR_OK

11.1.42 IecTaskDebugEnter

RTS_RESULT IecTaskDebugEnter (RTS_HANDLE hIecTask)
Is called before entering debug loop, when IEC task is halted on breakpoint
hIecTask [IN]
Handle to IEC task
Result
ERR_OK

11.1.43 IecTaskDebugLeave

RTS_RESULT IecTaskDebugLeave (RTS_HANDLE hIecTask)
Is called after leaving debug loop, when IEC task was halted on breakpoint
hIecTask [IN]
Handle to IEC task
Result
ERR_OK

11.1.44 IecTaskSetContext

*RTS_RESULT IecTaskSetContext (RTS_HANDLE hIecTask, RegContext *pContext)*
Set context of the actual task
hIecTask [IN]
Handle to IEC task

pContext [IN]
Pointer to register context
Result
ERR_OK

11.1.45 IecTaskGetContext

*RTS_RESULT IecTaskGetContext (RTS_HANDLE hIecTask, RegContext *pContext)*
Get context of the actual task
hIecTask [IN]
Handle to IEC task
pContext [IN]
Pointer to register context
Result
ERR_OK

11.1.46 IecTaskExceptionHandler

RTS_RESULT IecTaskExceptionHandler (RTS_HANDLE hIecTask, RTS_UI32 ulException, RegContext Context)
Exception handler for the specified Iec task
hIecTask [IN]
Handle to IEC task
ulException [IN]
Exception
pContext [IN]
Pointer to register context
Result
ERR_OK

11.1.47 IecTaskEnableScheduling

RTS_RESULT IecTaskEnableScheduling (RTS_HANDLE hIecTask)

Enable scheduling for one specified task.

hIecTask [IN]
Handle of task to enable scheduling
Result
Error code

11.1.48 IecTaskDisableScheduling

RTS_RESULT IecTaskDisableScheduling (RTS_HANDLE hIecTask)
Disable scheduling for the specified task
hIecTask [IN]
Handle of task to disable scheduling
Result
Error code

11.1.49 IecTaskEnableWatchdog

RTS_RESULT IecTaskEnableWatchdog (RTS_HANDLE hIecTask)

Enable watchdog for the specified task

hIecTask [IN]
Handle of the task
Result
Error code

11.1.50 IecTaskDisableWatchdog

RTS_RESULT IecTaskDisableWatchdog (RTS_HANDLE hIecTask)

Disable watchdog for the specified task

hIecTask [IN]
Handle of the task
Result
Error code

11.1.51 IecTaskCheckWatchdog

*RTS_RESULT IecTaskCheckWatchdog (RTS_HANDLE hIecTask, RTS_SYSTIME *ptActUs)*

Check if the watchdog of a task expired.

This function is called by the scheduler, as well as from the task context.

If the watchdog expired, an exception is generated and (depending on the scheduler) the function will return ERR_FAILED. Note, that this function does not essentially return, because some schedulers are not able to recover from this error in a way that the function can return.

Handling of the Sensitivity:
- 0,1,2: Exception in first cycle if it exceeds WD time * sensitivity
- 3-MAX_INT: Exception in (sensitivity-1)'th cycle if time exceeded, or in first if time exceeded by WD time * sensitivity

hIecTask [IN]
Handle of the task
ptActUs [IN]
Optional pointer to actual time tick in microseconds. Can be NULL.
Result
Error code

11.1.52 IecFreeTasks

*RTS_RESULT IecFreeTasks (APPLICATION *pApp)*
Frees all IEC tasks
pApp [IN]
Pointer to specified application
Result
Error code

11.1.53 IecTaskCycle

RTS_RESULT IecTaskCycle (Task_Desc pTask)*

Main cycle of an Iec task

Adopts the "running state" of the task, based on the current state of it's application and starts the IEC task cycle:

- ITF_DONT_SCHEDULE: Return ERR_PENDING
- OS_PROGRAM_LOADED: If not set, ERR_FAILED is returned
- AS_SINGLE_CYCLE: if task has TS_SINGLE_CYCLE flag set, set also TS_STOP
- AS_RUN: reset TS_STOP
- Application status != AS_RUN, AS_SINGLE_CYCLE: set TS_STOP

Additionally this functions measures the time around the IEC task cycle and calls IecTaskCheckWatchdog() to create a watchdog exception if the time exceeded.

Note: This function may throw an exception, when program for the task was not, yet, loaded, or when the task configuration contains no pointer for the cycle code.

pTask [IN]
Pointer to task description
Result
Error code

11.1.54 IecTaskUpdateJitterTime

RTS_RESULT IecTaskUpdateJitterTime (Task_Desc pTask, RTS_SYSTIME *ptNow)*
Calculate jitter time of task
pTask [IN]
Pointer to task description

ptNow [IN]
Pointer to actual microsecond time tick
Result
Error code

11.1.55 IecTaskUpdateCycleTime

RTS_RESULT IecTaskUpdateCycleTime (Task_Desc pTask, RTS_SYSTIME *ptActUs)*
Calculate cycle time of task.
pTask [IN]
Pointer to task description
ptActUs [IN]
Pointer to actual time tick in microseconds
Result
Error code

11.1.56 IecTasksResetAllowed

*RTS_RESULT IecTasksResetAllowed (APPLICATION *pApp)*
FUnction to check, if reset is allowed in the actual state on this application
pApp [IN]
Pointer to specified application
Result
ERR_OK: Reset allowed, ERR_NOT_SUPPORTED: Not allowed (e.g. if one task is halted on a breakpoint)

11.1.57 IecTasksPrepareReset

*RTS_RESULT IecTasksPrepareReset (APPLICATION *pApp, int bResetOrigin)*
Prepare reset for all IEC tasks specified by application
pApp [IN]
Pointer to specified application
Result
Error code

11.1.58 IecTasksResetDone

*RTS_RESULT IecTasksResetDone (APPLICATION *pApp, int bResetOrigin)*
Initialize all IEC tasks after reset specified by application
pApp [IN]
Pointer to specified application
Result
Error code

11.1.59 IecTasksWaitForStop

*RTS_RESULT IecTasksWaitForStop (APPLICATION *pApp, RTS_UI32 ulTimeoutMs, unsigned long ulStopReason)*
Wait, if all Iec-Tasks has recognized the stop status of the application
pApp [IN]
Pointer to specified application
ulTimeoutMs [IN]
Timeout in milliseconds to wait for stop. Some timeouts are predefined (see CmpStd.h):
- RTS_TIMEOUT_INFINITE: Endless wait
- RTS_TIMEOUT_DEFAULT: Use default wait time
- RTS_TIMEOUT_NO_WAIT: No wait

ulStopReason [IN]
Stop reason. See corresponding category in CmpAppItf.h
Result
Error code

11.1.60 IecTaskInitOutputs

*RTS_RESULT IecTaskInitOutputs (APPLICATION *pApp)*

Init all outputs and write to the periphery. NOTE: Application must be in stop before calling this function!

pApp [IN]
Pointer to specified application

Result
Error code

11.1.61 IecTaskEnterExclusiveSection

RTS_RESULT IecTaskEnterExclusiveSection (void)

After this call no IEC task will be rescheduled, that is if it is not already running. May be used for executing onlinechange code. Each call must be matched with a call to TaskLeaveExclusiveSection.

Result
Error code

11.1.62 IecTaskLeaveExclusiveSection

RTS_RESULT IecTaskLeaveExclusiveSection (void)

Leave an exclusiv section, that has been started by TaskEnterExclusiveSection

Result
Error code

11.1.63 IecTaskEnterExclusiveSection2

*RTS_RESULT IecTaskEnterExclusiveSection2 (APPLICATION *pApp)*

Enter an exolusive section. After this call, no IEC task will be rescheduled of the specified application, if it is not already running. Each call must be matched with a call to TaskLeaveExclusiveSection.

Result
Error code

11.1.64 IecTaskLeaveExclusiveSection2

*RTS_RESULT IecTaskLeaveExclusiveSection2 (APPLICATION *pApp)*

Leave an exclusive section of the specified application, that has been entered by TaskEnterExclusiveSection

Result
Error code

11.1.65 IecTaskRegisterSlotCallbacks

*RTS_RESULT IecTaskRegisterSlotCallbacks (APPLICATION *pApp, RTS_I32 nSlotNr, PF_SLOT_CALLBACK pfSlotCallback, int bIecCallback)*

Register the given callback to all existing tasks of the application

On SIL2 Runtimes, this call is ignored in safety mode.

pApp [IN]
Pointer to an application

nSlotNr [IN]
Slotnumber

pfSlotCallback [IN]
Pointer to Slot Callback

bIecCallback [IN]
Defines if the function pointer is an IEC or a C function

Result
Error code

11.1.66 IecTaskUnregisterSlotCallbacks

*RTS_RESULT IecTaskUnregisterSlotCallbacks (APPLICATION *pApp, RTS_I32 nSlotNr, PF_SLOT_CALLBACK pfSlotCallback, int bIecCallback)*

Unregister the given callback from all existing tasks of the application

On SIL2 Runtimes, this call is ignored in safety mode.

pApp [IN]
Pointer to an application
nSlotNr [IN]
Slotnumber
pfSlotCallback [IN]
Pointer to Slot Callback
blecCallback [IN]
Defines if the function pointer is an IEC or a C function
Result
Error code

11.1.67 IecTaskRegisterSlotCallback

*RTS_RESULT IecTaskRegisterSlotCallback (Task_Desc *pTask, RTS_I32 nSlotNr, PF_SLOT_CALLBACK pfSlotCallback, int bIecCallback)*

Register the given callback to one specific IEC task.

On SIL2 Runtimes, this call throws an exception in safety mode.

pTask [IN]
Handle IEC Task
nSlotNr [IN]
Slotnumber
pfSlotCallback [IN]
Pointer to Slot Callback
blecCallback [IN]
Defines if the function pointer is an IEC or a C function
Result
Error code

11.1.68 IecTaskUnregisterSlotCallback

*RTS_RESULT IecTaskUnregisterSlotCallback (Task_Desc *pTask, RTS_I32 nSlotNr, PF_SLOT_CALLBACK pfSlotCallback, int bIecCallback)*

Unregister the given callback from the specific IEC task.

On SIL2 Runtimes, this call allows only IEC tasks in safety mode.

pTask [IN]
Handle IEC Task
nSlotNr [IN]
Slotnumber
pfSlotCallback [IN]
Pointer to Slot Callback
blecCallback [IN]
Defines if the function pointer is an IEC or a C function
Result
Error code

11.1.69 IecTaskGetHandle

*RTS_HANDLE IecTaskGetHandle (char *pszAppName, char *pszTaskName, RTS_RESULT *pResult)*

Get task, based on the application- and the task name.

pszAppName [IN]
Pointer to application name
pszTaskName [IN]
Pointer to task name
pResult [OUT]
Result of operation
Result

Task handle

11.1.70 IecTaskGetCurrent

*RTS_HANDLE IecTaskGetCurrent (RTS_RESULT *pResult)*

Is called to get the task description of the current running task

pResult [OUT]
Pointer to error code

Result
Task handle

11.1.71 IecTaskGetByIndex

*Task_Desc * IecTaskGetByIndex (APPLICATION* pApp, int iIndex, RTS_RESULT *pResult)*

Get Task Description of a task by it's application ID and it's task index.

Searches in the pool of all created tasks for the task, with the given index in the specified application.

pApp [IN]
Application object

iIndex [IN]
Index of task within the application

pResult [OUT]
Pointer to error code

Result
Pointer to task description, NULL if failed

11.1.72 IecTaskGetHandleByIndex

RTS_HANDLE IecTaskGetHandleByIndex (APPLICATION pApp, int iIndex, RTS_RESULT *pResult)*

Returns task handle of the task specified by index in an application

pApp [IN]
APPLICATION object

iIndex [IN]
Index of task in the task list of the application

pResult [OUT]
Pointer to error code

Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.73 IecTaskGetById

Task_Desc IecTaskGetById (APPLICATION* pappl, int iId, RTS_RESULT *pResult)*

Returns task handle of the task specified by its unique Id

pApp [IN]
APPLICATION object

iId [IN]
Unique Id of the task

pResult [OUT]
Pointer to error code

Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.74 IecTaskGetNumOfTasks

int IecTaskGetNumOfTasks (APPLICATION pAppl, RTS_RESULT *pResult)*

Returns the number of tasks of the specified applciation

pAppl [IN]
APPLICATION object

pResult [OUT]
Pointer to error code

Result

Error code

11.1.75 IecTaskEnableSchedulingAll

*RTS_RESULT IecTaskEnableSchedulingAll (APPLICATION *pApp)*

Enable scheduling for all tasks of the specified application

pApp [IN]
APPLICATION object

Result
Error code

11.1.76 IecTaskDisableSchedulingAll

*RTS_RESULT IecTaskDisableSchedulingAll (APPLICATION *pApp, RTS_HANDLE hIecTaskToExclude)*

Disable scheduling for all tasks of the specified application except the specified task. Typically this interface is used to disable all tasks except the debug task.

pApp [IN]
APPLICATION object

hIecTaskToExclude [IN]
Handle of task to exclude from scheduling. RTS_INVALID_HANDLE means, that all tasks are disabled

Result
Error code

11.1.77 IecTaskGetFirst

*RTS_HANDLE IecTaskGetFirst (char *pszAppName, RTS_RESULT *pResult)*

Get the first IEC task in the specified application

pszAppName [IN]
Application name, to which the task is bound

pResult [OUT]
Pointer to error code

Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.78 IecTaskGetNext

*RTS_HANDLE IecTaskGetNext (char *pszAppName, RTS_HANDLE hPrevIecTask, RTS_RESULT *pResult)*

Get the successor of an IEC task.

Return the successor of an IEC task, based on an application and a predecessor.

pszAppName [IN]
Application name, to which the task is bound

hPrevIecTask [IN]
Handle to previous task provided by IecTaskGetFirst()

pResult [OUT]
Pointer to error code

Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.79 IecTaskGetFirst2

*RTS_HANDLE IecTaskGetFirst2 (APPLICATION *pApp, RTS_RESULT *pResult)*

Get the first IEC task of the specified application

pApp [IN]
Handle of the application that contains the task

pResult [OUT]
Pointer to error code

Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.80 IecTaskGetNext2

*RTS_HANDLE IecTaskGetNext2 (APPLICATION *pApp, RTS_HANDLE hPrevIecTask, RTS_RESULT *pResult)*

Get the successor of an IEC task.

Return the successor of an IEC task, based on an application and a predecessor.

pApp [IN]
Handle of the application that contains the task
hPrevIecTask [IN]
Handle to previous task provided by IecTaskGetFirst2()
pResult [OUT]
Pointer to error code
Result
Handle to the task or RTS_INVALID_HANDLE if failed

11.1.81 IecTaskReload

*RTS_HANDLE IecTaskReload (RTS_HANDLE hIecTask, RTS_UI32 ulTimeoutMs, RTS_RESULT *pResult)*

Reload a specified IEC task. Reload means here: Delete the task at the actual position and create it newly.

hIecTask [IN]
Handle to the task to reload
ulTimeoutMs [IN]
Timeout in milliseconds to wait, until the task deleted itself. Timeout can be one of the following predefined values: RTS_TIMEOUT_DEFAULT: Default timeout to delete the task RTS_TIMEOUT_NO_WAIT: Immediate deletion of the task
pResult [OUT]
Pointer to error code
Result
Handle to the new created task

11.1.82 IecTaskCalculateId

*int IecTaskCalculateId (APPLICATION *pApp, int iIndex, RTS_RESULT *pResult)*

Calculate a task ID based on an application index and a task index.

This ID corresponds directly to the IDs that are used in the CoDeSys programming system to identify a task.

The valid range of iIndex depends on the cpu. The limit is always the square route of UINT_MAX, because the upper half of the datatype is used for the application ID, and the lower half for the task index.

pApp [IN]
Pointer to the application of the task.
iIndex [IN]
Index of the task within it's application
pResult [OUT]
Pointer to error code
Result
Unique Id of the task

11.1.83 IecTaskWaitTasksActive

*RTS_RESULT IecTaskWaitTasksActive (APPLICATION *pApp, RTS_UI32 ulTimeoutMs)*

Function to make a busy wait, while at least one task of the specified application is active

pApp [IN]
Handle of the application that contains the tasks to check. If pApp == NULL, all IEC tasks are checked.

ulTimeoutMs [IN]
Timeout in milliseconds to wait for deleting the task
Result
Error code:

11.1.84 IecTaskSingleCycle

*RTS_RESULT IecTaskSingleCycle (APPLICATION *pApp)*
Activates a single cycle on every cyclic and freewheeling task of the specified application
pApp [IN]
Handle of the application to do a single cycle
Result
Error code

11.1.85 IecTaskResetStatistics

RTS_RESULT IecTaskResetStatistics (RTS_HANDLE hIecTask)
Reset the task statistics of a task (see Task_Info member e.g. dwCycleTime, dwAverageCycleTime, etc.)
hIecTask [IN]
Handle to the task
Result
Error code

11.1.86 IecTaskDelete

RTS_RESULT IecTaskDelete (RTS_HANDLE hIecTask)
Delete an IEC task
hIecTask [IN]
Handle to task
Result
Error code

11.1.87 IecTaskDelete2

RTS_RESULT IecTaskDelete2 (RTS_HANDLE hIecTask, RTS_UI32 ulTimeoutMs)
Delete an IEC task with timeout
ulTimeoutMs [IN]
Timeout in milliseconds to wait for deleting the task Some timeouts are predefined (see CmpStd.h):
- RTS_TIMEOUT_DEFAULT: Use default wait time
- RTS_TIMEOUT_NO_WAIT: No wait

Result
Error code

12 CmpIoDrvIec

Wrapper component for an IEC driver Every IEC driver is encapsulated by this wrapper, because calling the IEC functions must be done with SysCpuCallIecFuncWithParams(). The IoMgr uses the wrapper interface. And internally the wrapper calls the FB interface methods. o- IoDrvIec: Base.hInstance -> pIIoDrvIec o- IoDrvFB: hInterface -> pIBase BaseIec.hInstance -> pFBInstance

12.1 CmpIoDrvIecItf

Wrapper to access the interface functions of an IEC driver from C.

The registration of a new IEC driver instance is going like this:

. +--------------+ +--------+ +-----------+ . |IEC Driver Lib| |CmpIoMgr| |CmpIoDrvIec| . +--------------+ +--------+ +-----------+

The handle of the I/O driver wrapper looks like this:

. +------------------------+ . |CCmpIoDrvIec | . +------------------------+ . |hInstance | ----. . +------------------------+ | . . |IoDrvGe

The main difference between the registration with IoDrvRegisterInstance() vs. IoDrvRegisterInstance2() is, that the first one was initializing all IEC function pointers by itself and the second is using BaseUpdateConfiguration() for this purpose. IoDrvRegisterInstance is now depricated.

12.1.1 IoDrvRegisterInstance

*RTS_RESULT IoDrvRegisterInstance (iodrviecregisterinstance_struct *p)*

Obsolete! This interface is not used anymore.

Note: On SIL2 Runtimes, this call leads to an exception.

Result
Error code

12.1.2 IoDrvRegisterInstance2

*RTS_RESULT IoDrvRegisterInstance2 (iodrviecregisterinstance2_struct *p)*

Register an IEC I/O driver.

A call to this function creates a new instance of the CmpIoDrvIec component, which acts as a wrapper for the registered IEC I/O driver. It will, by itself, call IoMgrRegisterInstance2() to register the newly created C-Wrapper Interface at the I/O Manager.

p [IN]
IEC Parameter structure

Result
Error code

12.1.3 IoDrvUnregisterInstance

*RTS_RESULT IoDrvUnregisterInstance (iodrviecunregisterinstance_struct *p)*

Unregister an IEC I/O driver.

Delete a driver instance and unregister it from the I/O Manager.

Note: On SIL2 runtimes, this function is only allowed in debug mode. When it is called outside of the debug mode, it will throw an exception.

p [IN]
IEC Parameter structure

Result
Error code

12.1.4 IoDrvGetIecInterface

IBase IoDrvGetIecInterface (RTS_HANDLE hIoDrv, RTS_RESULT *pResult)*

Get an IEC I/O driver interface pointer, based on a handle to it's corresponding C-Wrapper.

hIoDrv [IN]
Handle of a C-Wrapper

Result [OUT]
Result of the function
Result
Pointer to the IEC base interface

12.2 CmpIoDrvItf

Standard IO-driver interface.

THis interface is used for I/O drivers written in C as well as I/O drivers written in IEC. IEC I/O drivers are typically written as a function block, which implementes the IIoDrv Interface.

To be able to access this IEC interface from C (which is for example necessary for the IoMgr), the component CmpIoDrvIec acts as a wrapper between C and IEC and implements exactly this interface.

12.2.1 Define: CT_PROGRAMMABLE

Category: Connector types
Type:
Define: CT_PROGRAMMABLE
Key: 0x1000

These are the connector types which are used most frequently. This list is not complete.

12.2.2 Define: CF_ENABLE

Category: Connector flags
Type:
Define: CF_ENABLE
Key: 0x0001
Flags that specifies diagnostic informations of a connector

12.2.3 Define: CO_NONE

Category: Connector options
Type:
Define: CO_NONE
Key: 0x0000
Options to specify properties of a connector

12.2.4 Define: PVF_FUNCTION

Category: Parameter value flags
Type:
Define: PVF_FUNCTION
Key: 0x0001

Defines the type of the parameter

Actually only PVF_POINTER is currently supported by CoDeSys

12.2.5 Define: FIP_NETX_DEVICE

Category: Fieldbus independent parameters
Type:
Define: FIP_NETX_DEVICE
Key: 0x70000000

12.2.6 Define: TMT_INPUTS

Category: Task map types
Type:
Define: TMT_INPUTS
Key: 0x0001

Types of IO-channels in a task map

12.2.7 Define: DRVPROP_CONSISTENCY

Category: Driver property flags
Type:
Define: DRVPROP_CONSISTENCY

Key: 0x0001

12.2.8 Define: CMLSI_BaseTypeMask

Category: Channel Map List Swapping Information
Type:
Define: CMLSI_BaseTypeMask
Key: 0x00FF

Every Io driver needs information on the base data type of an io channel to perform a correct swapping operation.The basedata type and some additional swapping information can be found in the wDummy Byte of the ChannelMapList struct.

12.2.9 IoDrvCreate

*RTS_HANDLE IoDrvCreate (RTS_HANDLE hIIoDrv, CLASSID ClassId, int iId, RTS_RESULT *pResult)*

Create a new I/O driver instance.

This function is obsolete, because the instance has to be created by the caller before he registers the I/O driver in the I/O Manager.

hIIoDrv [IN]
Handle to the IO-driver interface. Must be 0 and is filled automatically by calling the CAL_IoDrvCreate() macro!

ClassId [IN]
ClassID of the driver. See "Class IDs" section in CmpItf.h or in the Dep.h file of the IO-driver.

iId [IN]
Instance number of the IO-driver

pResult [OUT]
Pointer to error code

Result
Should return RTS_INVALID_HANDLE

12.2.10 IoDrvGetInfo

*RTS_RESULT IoDrvGetInfo (RTS_HANDLE hIoDrv, IoDrvInfo **ppIoDrv)*

Get a driver specific info structure.

This structure contains IDs and names of the driver.

hIoDrv [IN]
Handle to the IO-driver instance

ppIoDrv [OUT]
Pointer to pointer to the driver info. Pointer must be set by the driver to its internal driver info structure!

Result
Error code

12.2.11 IoDrvGetModuleDiagnosis

*RTS_RESULT IoDrvGetModuleDiagnosis (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Update Connector Flags in the device tree.

The driver should write the current diagnostic information (available, no driver, bus error,...) with the function IoDrvSetModuleDiagnosis() to the I/O connector.

This function can be used by other components or from the IEC application to update the diagnostic flags of the connector. To update the status from the driver, it has to call this function manually.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, that the diagostic information is requested

Result
Error code

12.2.12 IoDrvIdentify

*RTS_RESULT IoDrvIdentify (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Identify plugable I/O card or slave.

If the configurator supports scanning of modules, this function can be used our of a communication service to identify a module on the bus or locally on the PLC. This This might be done by a blinking LED or whatever the hardware supports.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, that should identify itself physically

Result
Error code

12.2.13 IoDrvReadInputs

*RTS_RESULT IoDrvReadInputs (RTS_HANDLE hIoDrv, IoConfigConnectorMap *pConnectorMapList, int nCount)*

Read inputs for one task

This function is called cyclically from every task that is using inputs. A part of the task map list, which contains only the data of one connector are passed to the driver (called connector map list).

If a driver has registered one instance to more than one connector, it might get more than one call with a different subset of the task map list.

The I/O driver should read the data from the local hardware or a buffer and write them to the corresponding IEC variables.

hIoDrv [IN]
Handle to the IO-driver instance

pConnectorMapList [IN]
Pointer to the connector map list

nCount [IN]
Number of entries in the connector map list

Result
Error code

12.2.14 IoDrvScanModules

*RTS_RESULT IoDrvScanModules (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector, IoConfigConnector **ppConnectorList, int *pnCount)*

Scan for submodules of a connector.

This function is executed when the driver is downloaded. It is called over a communication service.

The I/O driver should search for connected subumodules and return them via ppConnectorList.

NOTE: This interface is called synchronously and the buffer for the connector list has to be allocated by the driver.

The buffer might be freed at the next scan or at the next UpdateConfiguration.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, which layout should be scanned

ppConnectorList [IN]
Pointer to the scanned connectors (devices) to return

pnCount [IN]
Pointer to the number of entries in the connector list to return

Result
Error code

12.2.15 IoDrvStartBusCycle

*RTS_RESULT IoDrvStartBusCycle (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Start bus cycle for a specific connector.

The bus cycle task is defined globally for the whole PLC or locally for a specific I/O connector in the CoDeSys project. This call can be used by the I/O driver to flush the I/O data if it was cached before.

This way we can get a better and consistent timing on the bus.

d

Note: This function is called for every connector which has a registered I/O driver and "needsbuscycle" set in the device description (this means that it might also be called for children of the connector).

Depending on the device description, this function might be executed at the beginning or at the end of the task cycle.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, on which the buscycle must be triggered

Result
Error code

12.2.16 IoDrvUpdateConfiguration

*RTS_RESULT IoDrvUpdateConfiguration (RTS_HANDLE hIoDrv, IoConfigConnector *pConnectorList, int nCount)*

Propagate I/O configuration to the drivers.

This call passes the I/O configuration (based on the configuration tree in the CoDeSys programming system) to all registered I/O drivers. Every driver has the chance to pass this tree and to register itself for a specific connector.

The driver can use the I/O Manager Interface to iterate over the I/O Connectors and to read the I/O Parameters. If it decides to handle the I/Os of one of those connectors, it can register it's driver handle (IBase) to the connector in the member hIoDrv.

This function is called when the application is initialized as well as when it is de- or reinitialized. In this case it is called with pConnectorList = NULL.

hIoDrv [IN]
Handle to the IO-driver instance

pConnectorList [IN]
Pointer to the complete connector list

nCount [IN]
Number of entries in the connector list

Result
Error code

12.2.17 IoDrvUpdateMapping

*RTS_RESULT IoDrvUpdateMapping (RTS_HANDLE hIoDrv, IoConfigTaskMap *pTaskMapList, int nCount)*

Propagate the task map lists to the drivers.

This functions gives the drivers a chance to optimize their internal data structures based on the real task map lists. The function is called on every initialization of the application (download, bootproject,...).

hIoDrv [IN]
Handle to the IO-driver instance

pTaskMapList [IN]
Pointer to the task map list of one task

nCount [IN]
Number of entries in the map list

Result
Error code

12.2.18 IoDrvWatchdogTrigger

*RTS_RESULT IoDrvWatchdogTrigger (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Trigger the hardware watchdog of a driver.

This function is depricated and not used anymore.

hIoDrv [IN]
Handle to the IO-driver instance
pConnector [IN]
Pointer to the connector, on which the watchdog should be retriggered
Result
Should return ERR_NOTIMPLEMENTED

12.2.19 IoDrvWriteOutputs

*RTS_RESULT IoDrvWriteOutputs (RTS_HANDLE hIoDrv, IoConfigConnectorMap *pConnectorMapList, int nCount)*

Write outputs for one task

This function is called cyclically from every task that is using outputs. A part of the task map list, which contains only the data of one connector are passed to the driver (called connector map list).

If a driver has registered one instance to more than one connector, it might get more than one call with a different subset of the task map list.

The I/O driver should write out the data to the local hardware, a buffer or a fieldbus.

hIoDrv [IN]
Handle to the IO-driver instance
pConnectorMapList [IN]
Pointer to the connector map list
nCount [IN]
Number of entries in the connector map list
Result
Error code

12.2.20 IoDrvDelete

RTS_RESULT IoDrvDelete (RTS_HANDLE hIoDrv, RTS_HANDLE hIIoDrv)

Delete an I/O driver instance.

This function is obsolete, because the instance has to be deleted by the caller after he unregisters the I/O driver from the I/O Manager.

Delete an IO-driver instance
hIoDrv [IN]
Handle to the IO-driver instance
hIIoDrv [IN]
Handle of the ITFID_ICmpIoDrv interface
Result
Should return ERR_NOTIMPLEMENTED

12.3 CmpIoDrvParameterItf

Interface to access parameters of an IO-driver.

12.3.1 IoDrvReadParameter

*RTS_RESULT IoDrvReadParameter (RTS_HANDLE hDevice, IoConfigConnector *pConnector, IoConfigParameter *pParameter, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Read a driver specific parameters.

These parameters can be read by the application, as well as, by an online service, which is triggered from the device configurator plugin in the CoDeSys programming system.

Note: If the I/O driver returns an error, the I/O Manager may try to read the parameter himself

Note2: On SIL2 runtimes, this interface is not supported.

hDevice [IN]
Handle to the IO-driver instance
pConnector [IN]
Pointer to the connector (might be determined with IoMgrConfigGetConnector).
pParameter [IN]
Pointer to the parameter (might be determined with IoMgrConfigGetParameter)
pData [IN]
Buffer where the read data is stored
ulBitSize [IN]
Size of the part of the parameter data, that should be read
ulBitOffset [IN]
Offset of the part of the parameter, that should be read
Result
Error code

12.3.2 IoDrvWriteParameter

*RTS_RESULT IoDrvWriteParameter (RTS_HANDLE hDevice, IoConfigConnector *pConnector, IoConfigParameter *pParameter, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Write a driver specific parameters.

These parameters can be written by the application, as well as, by an online service, which is triggered from the device configurator plugin in the CoDeSys programming system.

Note: On SIL2 runtimes, this interface is not supported. And if called in safe-mode, on SIL2 Runtimes, an exception is generated.

hDevice [IN]
Handle to the IO-driver instance
pConnector [IN]
Pointer to the connector (might be determined with IoMgrConfigGetConnector).
pParameter [IN]
Pointer to the parameter (might be determined with IoMgrConfigGetParameter)
pData [IN]
Buffer where the read data is stored
ulBitSize [IN]
Size of the part of the parameter data, that should be written
ulBitOffset [IN]
Offset of the part of the parameter, that should be written
Result
Error code

12.4 CmpIoDrvParameter2Itf

Interface to access parameters of an IO-driver by their parameter id

12.4.1 IoDrvReadParameterById

*RTS_RESULT IoDrvReadParameterById (RTS_HANDLE hDevice, IoConfigConnector *pConnector, RTS_UI32 dwParamId, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Result
ERR_OK

12.5 CmpIoDrvProfinetItf

Interface to get access to a ProfiNet IO-driver.

12.5.1 IoDrvProfinet_Nominate

*RTS_RESULT IoDrvProfinet_Nominate (RTS_HANDLE hDevice, IoConfigConnector *pConnector)*

Result
ERR_OK

12.6 CmpIoDrvDPV1C2MasterItf

Interface of a profibus IO-driver for the DPV1 Class 2 Master interface.

12.6.1 Typedef: DPV1_C2_Abort

Stuctname: DPV1_C2_Abort

DPV1_C2_Abort

Typedef: typedef struct tagDPV1_C2_Abort { RTS_IEC_BOOL bEnable; RTS_IEC_BYTE byStationAddress; RTS_IEC_BYTE byDummy[2]; RTS_IEC_UDINT udiConnectID; RTS_IEC_UDINT C_Ref; RTS_IEC_BYTE abyError[4]; RTS_IEC_WORD wOpState; RTS_IEC_BOOL bOldEnable; RTS_IEC_UDINT udiId; } DPV1_C2_Abort;

12.6.2 Typedef: DPV1_C2_Read

Stuctname: DPV1_C2_Read

DPV1_C2_Read

Typedef: typedef struct tagDPV1_C2_Read { RTS_IEC_BOOL bEnable; RTS_IEC_BYTE byStationAddress; RTS_IEC_BYTE bySlotNr; RTS_IEC_BYTE byIndex; RTS_IEC_UDINT udiConnectID; RTS_IEC_UDINT C_Ref; RTS_IEC_WORD wLen; RTS_IEC_BYTE byDummy[2]; RTS_IEC_BYTE *pBuffer; RTS_IEC_BYTE abyError[4]; RTS_IEC_WORD wOpState; RTS_IEC_BOOL bOldEnable; RTS_IEC_UDINT udiId; } DPV1_C2_Read;

12.6.3 Typedef: DPV1_C2_Initiate

Stuctname: DPV1_C2_Initiate

DPV1_C2_Initiate

Typedef: typedef struct tagDPV1_C2_Initiate { RTS_IEC_BOOL bEnable; RTS_IEC_BYTE byStationAddress; RTS_IEC_BYTE byDummy[2]; RTS_IEC_DWORD dwSendTimeoutMs; RTS_IEC_UDINT C_Ref; RTS_IEC_BYTE Res_SAP; RTS_IEC_WORD wOpState; RTS_IEC_BYTE abyError[4]; RTS_IEC_UDINT udiConnectID; RTS_IEC_BOOL bOldEnable; RTS_IEC_UDINT udiId; } DPV1_C2_Initiate;

12.6.4 Typedef: DPV1_C2_Write

Stuctname: DPV1_C2_Write

DPV1_C2_Write

Typedef: typedef struct tagDPV1_C2_Write { RTS_IEC_BOOL bEnable; RTS_IEC_BYTE byStationAddress; RTS_IEC_BYTE bySlotNr; RTS_IEC_BYTE byIndex; RTS_IEC_UDINT udiConnectID; RTS_IEC_UDINT C_Ref; RTS_IEC_WORD wLen; RTS_IEC_BYTE byDummy[2]; RTS_IEC_BYTE *pBuffer; RTS_IEC_BYTE abyError[4]; RTS_IEC_WORD wOpState; RTS_IEC_BOOL bOldEnable; RTS_IEC_UDINT udiId; } DPV1_C2_Write;

12.6.5 IoDrvDPV1_C2_M_Initiate

*RTS_RESULT IoDrvDPV1_C2_M_Initiate (RTS_HANDLE hIoDrv, DPV1_C2_Initiate *pInit)*

Result
ERR_OK

12.7 CmpIoDrvDPV1C1MasterItf

Interface of a profibus IO-driver for the DPV1 Class 1 Master interface.

12.7.1 IoDrvDPV1_C1_M_Read

*RTS_RESULT IoDrvDPV1_C1_M_Read (RTS_HANDLE hIoDrv, DPV1_C1_Read *pRead)*

Result
ERR_OK

13 CmpIOMgrEmbedded

Manager for all IO-drivers (IEC and ANSI-C) and IO-configuration.

13.1 Tasks

13.1.1 Task: IoMgrWatchdogTask

Category: Task
Priority: TASKPRIO_HIGH_BASE
Description: Task to handle IO-watchdog trigger.

13.2 CmpIoMgrItf

Interface of the IO-manager, which manages all IO-driver instances in the runtime system.

There are several data structures involved in the I/O update. All of them are registered in or passed to the I/O Manager. The following is an overview over the most essential structures and how they are passed to the I/O Manager:

. +------------------+ UpdateConfiguration() . +------------------+ | . +-----------------+ |-+ . | IoConfigTaskMap |-+ . +-------+----

Most calls to the I/O Manager are just passed directly through to the drivers. Depending on the kind of driver, this may passed through CmpIoDrvIEC or directly to it.

The following drawing illustrates how a call to e.g. ReadInputs() is passed to the drivers if there is a C and an IEC driver registered and used in one task:

. +--------+ +--------+ +--------+ +-----------+ +--------+ . |IEC Task| |CmpIoMgr| |IoDrvInC| |CmpIoDrvIEC| |IoDrvLib| . +----

The devices, which are configured in the CoDeSys programming system are downloaded in form of connector lists. All connectors, specified for the devices are downloaded in form of a list.

Beside neighbours, this list has although a parent role, that means, that every list member can point to it's parent.

For example:

. _____ . _,.--"" `""`--..__ . ,,' ___ `--.._ . _,i__ ,-ii_ `"`--.._ `-.. . ,;" `'-._,;" `'-._ ``-.._ `-._ . +----+ +----+ +----+ +----+ +-'--+ .

This list represents the following tree:

. ,-. . (A) . /`-'\ . .' | . ,-./ \,-. . ( B ) ( E ) . /`-'\ `-' . .' | . ,-./ \,-. . ( C ) ( D ) . `-' `-'

The API functions within this component await always a pointer to one of these connectors, when the input is a "Connecto List". The count doesn't essentially need to specify the absolut size of the list, but the size, starting at the connector that we passed to the function.

For example, if we want to iterate over the connector list above (in pseudo code):

GetFirst(A, 5) => A (count: 5) GetNext(A,5) => B (count: 4) GetNextChild(B, 4, B) => C (Count: 3) GetNextChild(B, 4,

Note: Because the parameter pnCount is an INOUT, you need to save the size of the list in a different variable when you use for example GetNextChild(). Because this value will be overwritten with the number of remaining entries in the list, starting at the connector that was returned.

There are two different kinds of driver handles, that the I/O Manager needs to differenciate:

- C Drivers
- IEC Drivers

The I/O Manager provides a common function interface for C Components and for IEC Applications. But while the Connectors and Paramter structures are completely the same for both sides, the driver handles are significantly different. Internally the I/O manager works with C driver handles. The abstraction to provide an IEC interface, which is based on Function Blocks, is done by the two components CmpIoDrvC and CmpIoDrvIec.

The component CmpIoDrvC manages all C drivers and CmpIoDrvIec manages all IEC drivers.

13.2.1 Define: NUM_OF_STATIC_DEVICES

Condition: #ifndef NUM_OF_STATIC_DEVICES
Category: Static defines
Type:
Define: NUM_OF_STATIC_DEVICES
Key: 5
Number of static IO-driver instances to store

13.2.2 Define: NUM_OF_STATIC_PARAMETER_STORAGE

Condition: #ifndef NUM_OF_STATIC_PARAMETER_STORAGE
Category: Static defines
Type:
Define: NUM_OF_STATIC_PARAMETER_STORAGE
Key: 10
Number of static parameter to store during writing operation. See setting

13.2.3 Define: IOMGR_PARAMETER_STORAGE_FILE

Condition: #ifndef IOMGR_PARAMETER_STORAGE_FILE
Category: File name definitions
Type:
Define: IOMGR_PARAMETER_STORAGE_FILE
Key: IoConfig.par
Name of the parameter storage to save the changed IO-config parameters

13.2.4 Define: IOMGR_WATCHTDOGTASK_TASK_DELAY

Condition: #ifndef IOMGR_WATCHTDOGTASK_TASK_DELAY
Category: Watchdo definitions
Type:
Define: IOMGR_WATCHTDOGTASK_TASK_DELAY
Key: 100
This is the initial watchdogtask task delay. On the first download of a watchdog time, this time is recalculated

13.2.5 Define: SRV_IOMGRSCANMODULES

Category: Online services
Type:
Define: SRV_IOMGRSCANMODULES
Key: 0x01

13.2.6 Define: TAG_DEVICESCAN_LIST

Category: Online tags
Type:
Define: TAG_DEVICESCAN_LIST
Key: 0x81

13.2.7 Define: EVT_PrepareUpdateConfiguration

Category: Events
Type:
Define: EVT_PrepareUpdateConfiguration
Key: MAKE_EVENTID
Event is sent before updating the IO-configuration

13.2.8 Define: EVT_DoneUpdateConfiguration

Category: Events
Type:
Define: EVT_DoneUpdateConfiguration
Key: MAKE_EVENTID
Event is sent after updating the IO-configuration

13.2.9 Define: EVT_ConfigAppStartedDone

Category: Events
Type:
Define: EVT_ConfigAppStartedDone
Key: MAKE_EVENTID
Event is sent after the config application is started

13.2.10 Define: EVT_PrepareConfigAppStopped

Category: Events
Type:
Define: EVT_PrepareConfigAppStopped
Key: MAKE_EVENTID
Event is sent before the config application is stopped

13.2.11 Define: EVT_PrepareUpdateMapping

Category: Events
Type:
Define: EVT_PrepareUpdateMapping
Key: MAKE_EVENTID
Event is sent before updating the IO-mapping of an application

13.2.12 Define: EVT_DoneUpdateMapping

Category: Events
Type:
Define: EVT_DoneUpdateMapping
Key: MAKE_EVENTID
Event is sent after updating the IO-mapping of an application

13.2.13 Define: EVT_UpdateDiagDone

Category: Events
Type:
Define: EVT_UpdateDiagDone
Key: MAKE_EVENTID
Event is sent after updating the module diagnosis of the specified driver, which supports the property DRVPROP_BACKGROUND_GETDIAG

13.2.14 Typedef: IoConfigParameter

Stuctname: IoConfigParameter

Parameter description. This entry describes completely a parameter of an connector.

Typedef: typedef struct tagIoConfigParameter { RTS_IEC_DWORD dwParameterId; RTS_IEC_BYTE *dwValue; RTS_IEC_WORD wType; RTS_IEC_WORD wLen; RTS_IEC_DWORD dwFlags; RTS_IEC_BYTE *dwDriverSpecific; } IoConfigParameter;

13.2.15 Typedef: IoConfigConnector

Stuctname: IoConfigConnector

Connector information. Each device is described completely as a set of one input- and one or more output-conenctors.

Typedef: typedef struct tagIoConfigConnector { RTS_IEC_WORD wType; RTS_IEC_WORD wOptions; RTS_IEC_DWORD dwFlags; RTS_IEC_HANDLE hIoDrv; RTS_IEC_DWORD dwNumOfParameters; IoConfigParameter *pParameterList; struct tagIoConfigConnector *pFather; } IoConfigConnector;

13.2.16 Typedef: IoConfigChannelMap

Stuctname: IoConfigChannelMap

Mapping information for a single channel. Every I/O-channel is described as a parameter, but with special meanings. The datatype of a channel can be simple (BOOL, BYTE, WORD, etc.) or array of simple types.

Typedef: typedef struct tagIoConfigChannelMap { IoConfigParameter *pParameter; RTS_IEC_BYTE *pbyIecAddress; RTS_IEC_WORD wParameterBitOffset; RTS_IEC_WORD wIecAddressBitOffset; RTS_IEC_WORD wSize; RTS_IEC_WORD wBaseTypeInformation; RTS_IEC_BYTE *dwDriverSpecific; } IoConfigChannelMap;

13.2.17 Typedef: IoConfigConnectorMap

Stuctname: IoConfigConnectorMap

Connector map to describe all IO-channels of one connector

Typedef: typedef struct tagIoConfigConnectorMap { IoConfigConnector *pConnector; RTS_IEC_BYTE *dwIoMgrSpecific; RTS_IEC_DWORD dwNumOfChannels; IoConfigChannelMap *pChannelMapList; } IoConfigConnectorMap;

13.2.18 Typedef: IoConfigTaskMap

Stuctname: IoConfigTaskMap

Mapping description for each task.

Typedef: typedef struct tagIoConfigTaskMap { RTS_IEC_DWORD dwTaskId; RTS_IEC_WORD wType; RTS_IEC_WORD wNumOfConnectorMap; IoConfigConnectorMap *pConnectorMapList; } IoConfigTaskMap;

13.2.19 Typedef: iomgrstartbuscycle_struct

Stuctname: iomgrstartbuscycle_struct

iomgrstartbuscycle

Typedef: typedef struct tagiomgrstartbuscycle_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_UDINT IoMgrStartBusCycle; VAR_OUTPUT } iomgrstartbuscycle_struct;

13.2.20 Typedef: iomgrupdateconfiguration2_struct

Stuctname: iomgrupdateconfiguration2_struct

iomgrupdateconfiguration2

Typedef: typedef struct tagiomgrupdateconfiguration2_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT nCount; VAR_INPUT RTS_IEC_STRING *pszConfigApplication; VAR_INPUT RTS_IEC_UDINT IoMgrUpdateConfiguration2; VAR_OUTPUT } iomgrupdateconfiguration2_struct;

13.2.21 Typedef: iomgrconfiggetconnector_struct

Stuctname: iomgrconfiggetconnector_struct

iomgrconfiggetconnector

Typedef: typedef struct tagiomgrconfiggetconnector_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT RTS_IEC_DWORD dwModuleType; VAR_INPUT RTS_IEC_DWORD dwInstance; VAR_INPUT IoConfigConnector *IoMgrConfigGetConnector; VAR_OUTPUT } iomgrconfiggetconnector_struct;

13.2.22 Typedef: iomgrconfiggetdriver_struct

Stuctname: iomgrconfiggetdriver_struct

iomgrconfiggetdriver

Typedef: typedef struct tagiomgrconfiggetdriver_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_DINT *pbIecDriver; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT IBase *IoMgrConfigGetDriver; VAR_OUTPUT } iomgrconfiggetdriver_struct;

13.2.23 Typedef: iomgrconfiggetconnectorlist_struct

Stuctname: iomgrconfiggetconnectorlist_struct

iomgrconfiggetconnectorlist

Typedef: typedef struct tagiomgrconfiggetconnectorlist_struct { IoConfigConnector **ppConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT RTS_IEC_UDINT IoMgrConfigGetConnectorList; VAR_OUTPUT } iomgrconfiggetconnectorlist_struct;

13.2.24 Typedef: iomgrconfiggetconnectorbydriver_struct

Stuctname: iomgrconfiggetconnectorbydriver_struct

iomgrconfiggetconnectorbydriver

Typedef: typedef struct tagiomgrconfiggetconnectorbydriver_struct { IBase *pIBase; VAR_INPUT RTS_IEC_DINT bIecDriver; VAR_INPUT RTS_IEC_DINT nIndex; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT IoConfigConnector *IoMgrConfigGetConnectorByDriver; VAR_OUTPUT } iomgrconfiggetconnectorbydriver_struct;

13.2.25 Typedef: iomgrconfiggetfirstconnector_struct

Stuctname: iomgrconfiggetfirstconnector_struct

iomgrconfiggetfirstconnector

Typedef: typedef struct tagiomgrconfiggetfirstconnector_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT RTS_IEC_WORD wType; VAR_INPUT IoConfigConnector *IoMgrConfigGetFirstConnector; VAR_OUTPUT } iomgrconfiggetfirstconnector_struct;

13.2.26 Typedef: iomgrconfigsetdiagnosis_struct

Stuctname: iomgrconfigsetdiagnosis_struct

iomgrconfigsetdiagnosis

Typedef: typedef struct tagiomgrconfigsetdiagnosis_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_DWORD dwFlags; VAR_INPUT RTS_IEC_UDINT IoMgrConfigSetDiagnosis; VAR_OUTPUT } iomgrconfigsetdiagnosis_struct;

13.2.27 Typedef: iomgrreadparameter_struct

Stuctname: iomgrreadparameter_struct

iomgrreadparameter

Typedef: typedef struct tagiomgrreadparameter_struct { RTS_IEC_DWORD dwModuleType; VAR_INPUT RTS_IEC_DWORD dwInstance; VAR_INPUT RTS_IEC_DWORD dwParameterId;

VAR_INPUT RTS_IEC_BYTE *pData; VAR_INPUT RTS_IEC_DWORD dwBitSize; VAR_INPUT RTS_IEC_DWORD dwBitOffset; VAR_INPUT RTS_IEC_UDINT IoMgrReadParameter; VAR_OUTPUT } iomgrreadparameter_struct;

13.2.28 Typedef: iomgrwatchdogtrigger_struct

Stuctname: iomgrwatchdogtrigger_struct

iomgrwatchdogtrigger

Typedef: typedef struct tagiomgrwatchdogtrigger_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_UDINT IoMgrWatchdogTrigger; VAR_OUTPUT } iomgrwatchdogtrigger_struct;

13.2.29 Typedef: iomgrconfigresetdiagnosis_struct

Stuctname: iomgrconfigresetdiagnosis_struct

iomgrconfigresetdiagnosis

Typedef: typedef struct tagiomgrconfigresetdiagnosis_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_DWORD dwFlags; VAR_INPUT RTS_IEC_UDINT IoMgrConfigResetDiagnosis; VAR_OUTPUT } iomgrconfigresetdiagnosis_struct;

13.2.30 Typedef: iomgrconfiggetnextconnector_struct

Stuctname: iomgrconfiggetnextconnector_struct

iomgrconfiggetnextconnector

Typedef: typedef struct tagiomgrconfiggetnextconnector_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT RTS_IEC_WORD wType; VAR_INPUT IoConfigConnector *IoMgrConfigGetNextConnector; VAR_OUTPUT } iomgrconfiggetnextconnector_struct;

13.2.31 Typedef: iomgrupdatemapping_struct

Stuctname: iomgrupdatemapping_struct

iomgrupdatemapping

Typedef: typedef struct tagiomgrupdatemapping_struct { IoConfigTaskMap *pTaskMapList; VAR_INPUT RTS_IEC_DINT nCount; VAR_INPUT RTS_IEC_UDINT IoMgrUpdateMapping; VAR_OUTPUT } iomgrupdatemapping_struct;

13.2.32 Typedef: iomgrgetfirstdriverinstance_struct

Stuctname: iomgrgetfirstdriverinstance_struct

iomgrgetfirstdriverinstance

Typedef: typedef struct tagiomgrgetfirstdriverinstance_struct { RTS_IEC_DINT *pbIecDriver; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT IBase *IoMgrGetFirstDriverInstance; VAR_OUTPUT } iomgrgetfirstdriverinstance_struct;

13.2.33 Typedef: iomgrregisterconfigapplication_struct

Stuctname: iomgrregisterconfigapplication_struct

iomgrregisterconfigapplication

Typedef: typedef struct tagiomgrregisterconfigapplication_struct { RTS_IEC_STRING *pszConfigApplication; VAR_INPUT RTS_IEC_UDINT IoMgrRegisterConfigApplication; VAR_OUTPUT } iomgrregisterconfigapplication_struct;

13.2.34 Typedef: iomgrsetdriverproperties_struct

Stuctname: iomgrsetdriverproperties_struct

iomgrsetdriverproperties

Typedef: typedef struct tagiomgrsetdriverproperties_struct { RTS_IEC_HANDLE hIoDrv; VAR_INPUT RTS_IEC_DWORD ulProperties; VAR_INPUT RTS_IEC_UDINT IoMgrSetDriverProperties; VAR_OUTPUT } iomgrsetdriverproperties_struct;

13.2.35 Typedef: iomgrsetdriverproperty_struct

Stuctname: iomgrsetdriverproperty_struct

iomgrsetdriverproperty

Typedef: typedef struct tagiomgrsetdriverproperty_struct { RTS_IEC_HANDLE hIoDrv; VAR_INPUT RTS_IEC_DWORD ulProperty; VAR_INPUT RTS_IEC_DINT bValue; VAR_INPUT RTS_IEC_UDINT IoMgrSetDriverProperty; VAR_OUTPUT } iomgrsetdriverproperty_struct;

13.2.36 Typedef: iomgrgetdriverproperties_struct

Stuctname: iomgrgetdriverproperties_struct

iomgrgetdriverproperties

Typedef: typedef struct tagiomgrgetdriverproperties_struct { RTS_IEC_HANDLE hIoDrv; VAR_INPUT RTS_IEC_DWORD **ppProperties; VAR_INPUT RTS_IEC_UDINT IoMgrGetDriverProperties; VAR_OUTPUT } iomgrgetdriverproperties_struct;

13.2.37 Typedef: iomgrconfiggetparametervaluepointer_struct

Stuctname: iomgrconfiggetparametervaluepointer_struct

iomgrconfiggetparametervaluepointer

Typedef: typedef struct tagiomgrconfiggetparametervaluepointer_struct { IoConfigParameter *pParameter; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IoMgrConfigGetParameterValuePointer; VAR_OUTPUT } iomgrconfiggetparametervaluepointer_struct;

13.2.38 Typedef: iomgrscanmodules_struct

Stuctname: iomgrscanmodules_struct

iomgrscanmodules

Typedef: typedef struct tagiomgrscanmodules_struct { IoConfigConnector *pConnector; VAR_INPUT IoConfigConnector **ppConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT RTS_IEC_UDINT IoMgrScanModules; VAR_OUTPUT } iomgrscanmodules_struct;

13.2.39 Typedef: iomgrgetmoduldiagnosis_struct

Stuctname: iomgrgetmoduldiagnosis_struct

iomgrgetmoduldiagnosis

Typedef: typedef struct tagiomgrgetmoduldiagnosis_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_UDINT IoMgrGetModuleDiagnosis; VAR_OUTPUT } iomgrgetmoduldiagnosis_struct;

13.2.40 Typedef: iomgrunregisterinstance_struct

Stuctname: iomgrunregisterinstance_struct

iomgrunregisterinstance

Typedef: typedef struct tagiomgrunregisterinstance_struct { RTS_IEC_HANDLE hInterface; VAR_INPUT RTS_IEC_UDINT IoMgrUnregisterInstance; VAR_OUTPUT } iomgrunregisterinstance_struct;

13.2.41 Typedef: iomgrconfiggetnextchild_struct

Stuctname: iomgrconfiggetnextchild_struct

iomgrconfiggetnextchild

Typedef: typedef struct tagiomgrconfiggetnextchild_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT IoConfigConnector *pFather; VAR_INPUT IoConfigConnector *IoMgrConfigGetNextChild; VAR_OUTPUT } iomgrconfiggetnextchild_struct;

13.2.42 Typedef: iomgrconfiggetparameter_struct

Stuctname: iomgrconfiggetparameter_struct

iomgrconfiggetparameter

Typedef: typedef struct tagiomgrconfiggetparameter_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_DWORD dwParameterId; VAR_INPUT IoConfigParameter *IoMgrConfigGetParameter; VAR_OUTPUT } iomgrconfiggetparameter_struct;

13.2.43 Typedef: iomgrupdatemapping2_struct

Stuctname: iomgrupdatemapping2_struct

iomgrupdatemapping2

Typedef: typedef struct tagiomgrupdatemapping2_struct { IoConfigTaskMap *pTaskMapList; VAR_INPUT RTS_IEC_DINT nCount; VAR_INPUT RTS_IEC_STRING *pszConfigApplication; VAR_INPUT RTS_IEC_UDINT IoMgrUpdateMapping2; VAR_OUTPUT } iomgrupdatemapping2_struct;

13.2.44 Typedef: iomgrwriteparameter_struct

Stuctname: iomgrwriteparameter_struct

iomgrwriteparameter

Typedef: typedef struct tagiomgrwriteparameter_struct { RTS_IEC_DWORD dwModuleType; VAR_INPUT RTS_IEC_DWORD dwInstance; VAR_INPUT RTS_IEC_DWORD dwParameterId; VAR_INPUT RTS_IEC_BYTE *pData; VAR_INPUT RTS_IEC_DWORD dwBitSize; VAR_INPUT

RTS_IEC_DWORD dwBitOffset; VAR_INPUT RTS_IEC_UDINT IoMgrWriteParameter; VAR_OUTPUT } iomgrwriteparameter_struct;

13.2.45 Typedef: iomgrupdateconfiguration_struct

Stuctname: iomgrupdateconfiguration_struct

iomgrupdateconfiguration

Typedef: typedef struct tagiomgrupdateconfiguration_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT nCount; VAR_INPUT RTS_IEC_UDINT IoMgrUpdateConfiguration; VAR_OUTPUT } iomgrupdateconfiguration_struct;

13.2.46 Typedef: iomgrgetconfigapplication_struct

Stuctname: iomgrgetconfigapplication_struct

iomgrgetconfigapplication

Typedef: typedef struct tagiomgrgetconfigapplication_struct { RTS_IEC_STRING *pszConfigApplication; VAR_INPUT RTS_IEC_DINT *pnMaxLen; VAR_INPUT RTS_IEC_UDINT IoMgrGetConfigApplication; VAR_OUTPUT } iomgrgetconfigapplication_struct;

13.2.47 Typedef: iomgrconfiggetfirstchild_struct

Stuctname: iomgrconfiggetfirstchild_struct

iomgrconfiggetfirstchild

Typedef: typedef struct tagiomgrconfiggetfirstchild_struct { IoConfigConnector *pConnectorList; VAR_INPUT RTS_IEC_DINT *pnCount; VAR_INPUT IoConfigConnector *pFather; VAR_INPUT IoConfigConnector *IoMgrConfigGetFirstChild; VAR_OUTPUT } iomgrconfiggetfirstchild_struct;

13.2.48 Typedef: iomgrconfiggetparametervalueword_struct

Stuctname: iomgrconfiggetparametervalueword_struct

iomgrconfiggetparametervalueword

Typedef: typedef struct tagiomgrconfiggetparametervalueword_struct { IoConfigParameter *pParameter; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_WORD IoMgrConfigGetParameterValueWord; VAR_OUTPUT } iomgrconfiggetparametervalueword_struct;

13.2.49 Typedef: iomgrwriteoutputs_struct

Stuctname: iomgrwriteoutputs_struct

iomgrwriteoutputs

Typedef: typedef struct tagiomgrwriteoutputs_struct { IoConfigTaskMap *pTaskMap; VAR_INPUT RTS_IEC_UDINT IoMgrWriteOutputs; VAR_OUTPUT } iomgrwriteoutputs_struct;

13.2.50 Typedef: iomgrconfiggetparametervaluebyte_struct

Stuctname: iomgrconfiggetparametervaluebyte_struct

iomgrconfiggetparametervaluebyte

Typedef: typedef struct tagiomgrconfiggetparametervaluebyte_struct { IoConfigParameter *pParameter; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE IoMgrConfigGetParameterValueByte; VAR_OUTPUT } iomgrconfiggetparametervaluebyte_struct;

13.2.51 Typedef: iomgrunregisterconfigapplication_struct

Stuctname: iomgrunregisterconfigapplication_struct

iomgrunregisterconfigapplication

Typedef: typedef struct tagiomgrunregisterconfigapplication_struct { RTS_IEC_STRING *pszConfigApplication; VAR_INPUT RTS_IEC_UDINT IoMgrUnregisterConfigApplication; VAR_OUTPUT } iomgrunregisterconfigapplication_struct;

13.2.52 Typedef: iomgrregisterinstance2_struct

Stuctname: iomgrregisterinstance2_struct

iomgrregisterinstance2

Typedef: typedef struct tagiomgrregisterinstance2_struct { RTS_IEC_DWORD dwClassId; VAR_INPUT IBase *pItf; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *IoMgrRegisterInstance2; VAR_OUTPUT } iomgrregisterinstance2_struct;

13.2.53 Typedef: iomgrreadinputs_struct

Stuctname: iomgrreadinputs_struct

iomgrreadinputs

Typedef: typedef struct tagiomgrreadinputs_struct { IoConfigTaskMap *pTaskMap; VAR_INPUT RTS_IEC_UDINT IoMgrReadInputs; VAR_OUTPUT } iomgrreadinputs_struct;

13.2.54 Typedef: iomgrconfiggetparametervaluedword_struct

Stuctname: iomgrconfiggetparametervaluedword_struct

iomgrconfiggetparametervaluedword

Typedef: typedef struct tagiomgrconfiggetparametervaluedword_struct { IoConfigParameter *pParameter; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DWORD IoMgrConfigGetParameterValueDword; VAR_OUTPUT } iomgrconfiggetparametervaluedword_struct;

13.2.55 Typedef: iomgridentify_struct

Stuctname: iomgridentify_struct

iomgridentify

Typedef: typedef struct tagiomgridentify_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_UDINT IoMgrIdentify; VAR_OUTPUT } iomgridentify_struct;

13.2.56 Typedef: iomgrgetnextdriverinstance_struct

Stuctname: iomgrgetnextdriverinstance_struct

iomgrgetnextdriverinstance

Typedef: typedef struct tagiomgrgetnextdriverinstance_struct { IBase *pIBasePrev; VAR_INPUT RTS_IEC_DINT *pbIecDriver; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT IBase *IoMgrGetNextDriverInstance; VAR_OUTPUT } iomgrgetnextdriverinstance_struct;

13.2.57 Typedef: iomgrcopyinputbe_struct

Stuctname: iomgrcopyinputbe_struct

iomgrcopyinputbe

Typedef: typedef struct tagiomgrcopyinputbe_struct { IoConfigChannelMap *pChannel; VAR_INPUT RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_RESULT IoMgrCopyInputBE; VAR_OUTPUT } iomgrcopyinputbe_struct;

13.2.58 Typedef: iomgrcopyinputle_struct

Stuctname: iomgrcopyinputle_struct

iomgrcopyinputle

Typedef: typedef struct tagiomgrcopyinputle_struct { IoConfigChannelMap *pChannel; VAR_INPUT RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_RESULT IoMgrCopyInputLE; VAR_OUTPUT } iomgrcopyinputle_struct;

13.2.59 Typedef: iomgrcopyoutputbe_struct

Stuctname: iomgrcopyoutputbe_struct

iomgrcopyoutputbe

Typedef: typedef struct tagiomgrcopyoutputbe_struct { IoConfigChannelMap *pChannel; VAR_INPUT RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_RESULT IoMgrCopyOutputBE; VAR_OUTPUT } iomgrcopyoutputbe_struct;

13.2.60 Typedef: iomgrcopyoutputle_struct

Stuctname: iomgrcopyoutputle_struct

iomgrcopyoutputle

Typedef: typedef struct tagiomgrcopyoutputle_struct { IoConfigChannelMap *pChannel; VAR_INPUT RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_RESULT IoMgrCopyOutputLE; VAR_OUTPUT } iomgrcopyoutputle_struct;

13.2.61 Typedef: iomgrconfiggetiodriver_struct

Stuctname: iomgrconfiggetiodriver_struct

iomgrconfiggetiodriver

Typedef: typedef struct tagiomgrconfiggetiodriver_struct { IoConfigConnector *pConnector; VAR_INPUT RTS_IEC_DINT *pbIecDriver; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT IBase *IoMgrConfigGetIoDriver; VAR_OUTPUT } iomgrconfiggetiodriver_struct;

13.2.62 Typedef: EVTPARAM_CmpIoMgr

Stuctname: EVTPARAM_CmpIoMgr

Category: Event parameter

Typedef: typedef struct tagEVTPARAM_CmpIoMgr { IoConfigConnector *pConnectorList; RTS_IEC_DINT nCount; } EVTPARAM_CmpIoMgr;

13.2.63 Typedef: EVTPARAM_CmpIoMgrUpdateMapping

Stuctname: EVTPARAM_CmpIoMgrUpdateMapping
Category: Event parameter
Typedef: typedef struct { IoConfigTaskMap *pTaskMapList; int nCount; char *pszConfigApplication; } EVTPARAM_CmpIoMgrUpdateMapping;

13.2.64 Typedef: EVTPARAM_CmpIoMgrUpdateDiag

Stuctname: EVTPARAM_CmpIoMgrUpdateDiag
Category: Event parameter
Typedef: typedef struct { RTS_HANDLE hIoDrv; } EVTPARAM_CmpIoMgrUpdateDiag;

13.2.65 iomgrstartbuscycle

*void iomgrstartbuscycle (iomgrstartbuscycle_struct *p)*
IEC Interface for the function IoMgrStartBusCycle()

13.2.66 iomgrupdateconfiguration2

*void iomgrupdateconfiguration2 (iomgrupdateconfiguration2_struct *p)*
IEC Interface for the function IoMgrUpdateConfiguration2()

13.2.67 iomgrconfiggetconnector

*void iomgrconfiggetconnector (iomgrconfiggetconnector_struct *p)*
IEC Interface for the function IoMgrConfigGetConnector()

13.2.68 iomgrconfiggetdriver

*void iomgrconfiggetdriver (iomgrconfiggetdriver_struct *p)*

This function behaves the same way as IoMgrConfigGetDriver() but uses IEC handles instead of C handles instead.

pConnector [IN]
Pointer to connector
pblecDriver [OUT]
Pointer to return if it is an Iec Driver
Result
Pointer to IBase interface

13.2.69 iomgrconfiggetconnectorlist

*void iomgrconfiggetconnectorlist (iomgrconfiggetconnectorlist_struct *p)*
IEC Interface for the function IoMgrConfigGetConnectorList()

13.2.70 iomgrconfiggetconnectorbydriver

*void iomgrconfiggetconnectorbydriver (iomgrconfiggetconnectorbydriver_struct *p)*
IEC Interface for the function IoMgrConfigGetConnectorByDriver()

13.2.71 iomgrconfiggetfirstconnector

*void iomgrconfiggetfirstconnector (iomgrconfiggetfirstconnector_struct *p)*
IEC Interface for the function IoMgrConfigGetFirstConnector()

13.2.72 iomgrconfigsetdiagnosis

*void iomgrconfigsetdiagnosis (iomgrconfigsetdiagnosis_struct *p)*
IEC Interface for the function IoMgrConfigSetDiagnosis()

13.2.73 iomgrreadparameter

*void iomgrreadparameter (iomgrreadparameter_struct *p)*
IEC Interface for the function IoMgrReadParameter()

13.2.74 iomgrwatchdogtrigger

*void iomgrwatchdogtrigger (iomgrwatchdogtrigger_struct *p)*
IEC Interface for the function IoMgrWatchdogTrigger()

13.2.75 iomgrconfigresetdiagnosis

*void iomgrconfigresetdiagnosis (iomgrconfigresetdiagnosis_struct *p)*
IEC Interface for the function IoMgrConfigResetDiagnosis()

13.2.76 iomgrconfiggetnextconnector

*void iomgrconfiggetnextconnector (iomgrconfiggetnextconnector_struct *p)*
IEC Interface for the function IoMgrConfigGetNextConnector()

13.2.77 iomgrupdatemapping

*void iomgrupdatemapping (iomgrupdatemapping_struct *p)*
IEC Interface for the function IoMgrUpdateMapping()

13.2.78 iomgrgetfirstdriverinstance

*void iomgrgetfirstdriverinstance (iomgrgetfirstdriverinstance_struct *p)*

This function acts in the same way as IoMgrGetFirstDriverInstance(), but uses IEC driver handles instead of C driver handles.

13.2.79 iomgrregisterconfigapplication

*void iomgrregisterconfigapplication (iomgrregisterconfigapplication_struct *p)*
IEC Interface for the function IoMgrRegisterConfigApplication()

13.2.80 iomgrsetdriverproperties

*void iomgrsetdriverproperties (iomgrsetdriverproperties_struct *p)*
IEC Interface for the function IoMgrSetDriverProperties()

13.2.81 iomgrsetdriverproperty

*void iomgrsetdriverproperty (iomgrsetdriverproperty_struct *p)*
IEC Interface for the function IoMgrSetDriverProperty()

13.2.82 iomgrgetdriverproperties

*void iomgrgetdriverproperties (iomgrgetdriverproperties_struct *p)*
IEC Interface for the function IoMgrGetDriverProperties()

13.2.83 iomgrconfiggetparametervaluepointer

*void iomgrconfiggetparametervaluepointer (iomgrconfiggetparametervaluepointer_struct *p)*
IEC Interface for the function IoMgrConfigGetParameterValuePointer()

13.2.84 iomgrscanmodules

*void iomgrscanmodules (iomgrscanmodules_struct *p)*
IEC Interface for the function IoMgrScanModules()

13.2.85 iomgrgetmodulediagnosis

*void iomgrgetmodulediagnosis (iomgrgetmodulediagnosis_struct *p)*
IEC Interface for the function IoMgrGetModuleDiagnosis()

13.2.86 iomgrunregisterinstance

*void iomgrunregisterinstance (iomgrunregisterinstance_struct *p)*
IEC Interface for the function IoMgrUnregisterInstance()

13.2.87 iomgrconfiggetnextchild

*void iomgrconfiggetnextchild (iomgrconfiggetnextchild_struct *p)*

IEC Interface for the function IoMgrConfigGetNextChild()

13.2.88 iomgrconfiggetparameter

*void iomgrconfiggetparameter (iomgrconfiggetparameter_struct *p)*
IEC Interface for the function IoMgrConfigGetParameter()

13.2.89 iomgrupdatemapping2

*void iomgrupdatemapping2 (iomgrupdatemapping2_struct *p)*
IEC Interface for the function IoMgrUpdateMapping2()

13.2.90 iomgrwriteparameter

*void iomgrwriteparameter (iomgrwriteparameter_struct *p)*
IEC Interface for the function IoMgrWriteParameter()

13.2.91 iomgrupdateconfiguration

*void iomgrupdateconfiguration (iomgrupdateconfiguration_struct *p)*
IEC Interface for the function IoMgrUpdateConfiguration()

13.2.92 iomgrgetconfigapplication

*void iomgrgetconfigapplication (iomgrgetconfigapplication_struct *p)*
IEC Interface for the function IoMgrGetConfigApplication()

13.2.93 iomgrconfiggetfirstchild

*void iomgrconfiggetfirstchild (iomgrconfiggetfirstchild_struct *p)*
IEC Interface for the function IoMgrConfigGetFirstChild()

13.2.94 iomgrconfiggetparametervalueword

*void iomgrconfiggetparametervalueword (iomgrconfiggetparametervalueword_struct *p)*
IEC Interface for the function IoMgrConfigGetParameterValueWord()

13.2.95 iomgrwriteoutputs

*void iomgrwriteoutputs (iomgrwriteoutputs_struct *p)*
IEC Interface for the function IoMgrWriteOutputs()

13.2.96 iomgrconfiggetparametervaluebyte

*void iomgrconfiggetparametervaluebyte (iomgrconfiggetparametervaluebyte_struct *p)*
IEC Interface for the function IoMgrConfigGetParameterValueByte()

13.2.97 iomgrunregisterconfigapplication

*void iomgrunregisterconfigapplication (iomgrunregisterconfigapplication_struct *p)*
IEC Interface for the function IoMgrUnregisterConfigApplication()

13.2.98 iomgrregisterinstance2

*void iomgrregisterinstance2 (iomgrregisterinstance2_struct *p)*
IEC Interface for the function IoMgrRegisterInstance2()

p [IN]
IBase interface of the IO-driver

13.2.99 iomgrreadinputs

*void iomgrreadinputs (iomgrreadinputs_struct *p)*
IEC Interface for the function IoMgrReadInputs()

13.2.100 iomgrconfiggetparametervaluedword

*void iomgrconfiggetparametervaluedword (iomgrconfiggetparametervaluedword_struct *p)*
IEC Interface for the function IoMgrConfigGetParameterValueDWord()

13.2.101 iomgridentify

*void iomgridentify (iomgridentify_struct *p)*
IEC Interface for the function IoMgrIdentify()

13.2.102 iomgrgetnextdriverinstance

*void iomgrgetnextdriverinstance (iomgrgetnextdriverinstance_struct *p)*

This function acts in the same way as IoMgrGetNextDriverInstance(), but uses IEC driver handles instead of C driver handles.

IEC Interface for the function IoMgrGetNextDriverInstance()

13.2.103 iomgrcopyinputbe

*void iomgrcopyinputbe (iomgrcopyinputbe_struct *p)*
IEC Interface for the function IoMgrCopyInputBE()

13.2.104 iomgrcopyinputle

*void iomgrcopyinputle (iomgrcopyinputle_struct *p)*
IEC Interface for the function IoMgrCopyInputLE()

13.2.105 iomgrcopyoutputbe

*void iomgrcopyoutputbe (iomgrcopyoutputbe_struct *p)*
IEC Interface for the function IoMgrCopyOutputBE()

13.2.106 iomgrcopyoutputle

*void iomgrcopyoutputle (iomgrcopyoutputle_struct *p)*
IEC Interface for the function IoMgrCopyOutputLE()

13.2.107 iomgrconfiggetiodriver

*void iomgrconfiggetiodriver (iomgrconfiggetiodriver_struct *p)*
IEC Interface for the function IoMgrConfigGetIoDriver()

13.2.108 IoMgrExceptionHandler

*RTS_RESULT IoMgrExceptionHandler (char *pszApplication, RTS_UI32 ulException)*
Exception handler of the IO-manager

pszApplication [IN]
Pointer to the specified application name, in which the exception was generated

ulException [IN]
Exception number of the exception. See SysExceptItf.h for details.

Result
error code

13.2.109 IoMgrRegisterInstance

*RTS_HANDLE IoMgrRegisterInstance (IBase *pIBase, RTS_RESULT *pResult)*

This function has the same behavior as IoMgrRegisterInstance2() but is not able to handle IEC drivers

pIBase [IN]
IBase interface of the IO-driver

pResult [OUT]
Pointer to error code

Result
Handle to the registered instance

13.2.110 IoMgrRegisterInstance2

*RTS_HANDLE IoMgrRegisterInstance2 (IBase *pIBase, int bIecDriver, RTS_RESULT *pResult)*

Register an instance of an IO-driver.

This function registers C as well as IEC drivers in a common device pool.

pIBase [IN]
IBase interface of the IO-driver
bIecDriver [IN]
Specifies if IBase describes a C or an IEC driver
pResult [OUT]
Pointer to error code
Result
Handle to the registered driver instance

13.2.111 IoMgrUnregisterInstance

*RTS_RESULT IoMgrUnregisterInstance (IBase *pIBase)*

Unregister an instance of an IO-driver

Note: For SIL2, this function is only allowed in debug mode, or from a safe context in safety mode.

pIBase [IN]
IBase interface of the IO-driver
Result
error code

13.2.112 IoMgrGetFirstDriverInstance

IBase IoMgrGetFirstDriverInstance (int *pbIecDriver, RTS_RESULT *pResult)*
Get first registered driver instance
pbIecDriver [INOUT]
Pointer to return, if the instance is an IEC- or C-driver: 1= pIBase is an interface of an IEC-driver, 0= C-driver
pResult [OUT]
Pointer to error code
Result
Handle to the first instance

13.2.113 IoMgrGetNextDriverInstance

IBase IoMgrGetNextDriverInstance (IBase *pIBasePrev, int *pbIecDriver, RTS_RESULT *pResult)*
Get the next registered driver instance
pIBasePrev [IN]
Pointer to IBase of the previous interface
pbIecDriver [INOUT]
Pointer to return, if the instance is an IEC- or C-driver: If the instance is an IEC- or C-driver 1= pIBase is an interface of an IEC-driver, 0= C-driver
pResult [OUT]
Pointer to error code
Result
Handle to the first instance

13.2.114 IoMgrSetDriverProperties

*RTS_RESULT IoMgrSetDriverProperties (IBase *pIBase, RTS_UI32 ulProperties)*

Set the properties of a driver.

This function overwrites all existing properties of the driver.

pIBase [IN]
Pointer to IBase interface of the driver
ulProperties [IN]
Properties of the driver. See category "Driver property flags" for detailed information.
Result
error code

13.2.115 IoMgrGetDriverProperties

*RTS_RESULT IoMgrGetDriverProperties (IBase *pIBase, RTS_UI32 **ppulProperty)*

Get a pointer to the properties of a driver.

pIBase [IN]
Pointer to IBase interface of the driver
ppulProperty [INOUT]
Pointer will point to the properties of the driver.
Result
error code

13.2.116 IoMgrSetDriverProperty

*RTS_RESULT IoMgrSetDriverProperty (IBase *pIBase, RTS_UI32 ulProperty, int bValue)*

Modify a Property Bitfield of a driver.

The bits in the mask ulProperty are set or reset, depending on the value of bValue.

pIBase [IN]
Pointer to IBase interface of the driver
bValue [IN]
Value to set.
ulProperty [IN]
Property of the driver. See category "Driver property flags" for detailed information.
Result
error code

13.2.117 IoMgrRegisterConfigApplication

*RTS_RESULT IoMgrRegisterConfigApplication (char *pszConfigApplication)*

Register the name of the application that contains the IO-configuration.

pszConfigApplication [IN]
Pointer to the name of the IO-config application
Result
error code

13.2.118 IoMgrUnregisterConfigApplication

*RTS_RESULT IoMgrUnregisterConfigApplication (char *pszConfigApplication)*

Unregister the name of the application that contains the IO-configuration.

pszConfigApplication [IN]
Pointer to the name of the IO-config application
Result
error code
Result
error code

13.2.119 IoMgrGetConfigApplication

*RTS_RESULT IoMgrGetConfigApplication (char *pszConfigApplication, int *pnMaxLen)*

Return the name of the application that contains the I/O configuration.

If the parameter pszConfigApplication is NULL, only the size of the string is returned.

pszConfigApplication [IN]
Pointer to get the name of the IO-config application
pnMaxLen [INOUT]
IN: size of pszConfigApplication, OUT: Size of config application if psz
Result
error code

13.2.120 IoMgrUpdateConfiguration

*RTS_RESULT IoMgrUpdateConfiguration (IoConfigConnector *pConnectorList, int nCount)*

Interface to inform all IO-drivers about a new IO-configuration.

On initialization, the parameter pConnectorList describes the whole I/O configuration of the IEC application.

On Reset or deletion of the application, this function is called, too, but then with the parameter pConnectorList set to NULL.

pConnectorList [IN]
Pointer to the complete connector list of the IO-configuration
nCount [INOUT]
Number of connectors in the connector list
Result
error code

13.2.121 IoMgrUpdateConfiguration2

*RTS_RESULT IoMgrUpdateConfiguration2 (IoConfigConnector *pConnectorList, int nCount, char *pszConfigApplication)*

Interface to inform all IO-drivers about a new IO-configuration.

On initialization, the parameter pConnectorList describes the whole I/O configuration of the IEC application.

On Reset or deletion of the application, this function is called, too, but then with the parameter pConnectorList set to NULL.

pConnectorList [IN]
Pointer to the complete connector list of the IO-configuration
nCount [IN]
Number of connectors in the connector list
pszConfigApplication [IN]
Pointer to the application name in which context this function is called
Result
error code

13.2.122 IoMgrUpdateMapping

*RTS_RESULT IoMgrUpdateMapping (IoConfigTaskMap *pTaskMapList, int nCount)*

Interface to inform all IO-drivers about a new IO-mapping.

pTaskMapList [IN]
Pointer to the complete task map list
nCount [INOUT]
Number of task map entries in the list
Result
error code

13.2.123 IoMgrUpdateMapping2

*RTS_RESULT IoMgrUpdateMapping2 (IoConfigTaskMap *pTaskMapList, int nCount, char *pszConfigApplication)*

Interface to inform all IO-drivers about a new IO-mapping.

pTaskMapList [IN]
Pointer to the complete task map list
nCount [IN]
Number of task map entries in the list
pszConfigApplication [IN]
Pointer to the application name in which context this function is called
Result
error code

13.2.124 IoMgrReadInputs

*RTS_RESULT IoMgrReadInputs (IoConfigTaskMap *pTaskMap)*

Interface to perform an update of all inputs of one task.

This function is called once out of each IEC-task, in which inputs are referenced. This call is passed to every I/O driver that is used by this task.

Throws Exception if dwDriverSpecific is 0.

pTaskMap [IN]
Pointer to the task map, which references all input channels of the task
Result
error code

13.2.125 IoMgrWriteOutputs

*RTS_RESULT IoMgrWriteOutputs (IoConfigTaskMap *pTaskMap)*

Interface to perform an update of all outputs of one task.

This function is called once out of each IEC-task, in which outputs are referenced. This call is passed to every I/O driver that is used by this task.

Throws Exception if dwDriverSpecific is 0.

pTaskMap [IN]
Pointer to the task map, which references all output channels of the task
Result
error code

13.2.126 IoMgrStartBusCycle

*RTS_RESULT IoMgrStartBusCycle (IoConfigConnector *pConnector)*

This function is called for every connector that has the setting "needs bus cycle" in the device description. It is only called once and only from the context of the so called "buscycle task". This task can be specified globally (= default for all drivers), or specifically for every driver.

pConnector [IN]
Pointer to the connector that needs a bus cycle
Result
error code

13.2.127 IoMgrScanModules

*RTS_RESULT IoMgrScanModules (IoConfigConnector *pConnector, IoConfigConnector **ppConnectorList, int *pnCount)*

Interface to scan submodule on the specified connector.

This interface is optional and may not be implemented by the runtime system.

pConnector [IN]
Pointer to connector
ppConnectorList [OUT]
List of submodule connectors
pnCount [OUT]
Pointer to elements in the connector list
Result
error code

13.2.128 IoMgrGetModuleDiagnosis

*RTS_RESULT IoMgrGetModuleDiagnosis (IoConfigConnector *pConnector)*

Get diagnostic flags of specified connector.

pConnector [IN]
Pointer to connector
Result
error code

13.2.129 IoMgrIdentify

*RTS_RESULT IoMgrIdentify (IoConfigConnector *pConnector)*

Interface to identify a device specified by connector.

This can be used for example to blink a LED on the device to identify it physically.

pConnector [IN]
Pointer to connector

Result
error code

13.2.130 IoMgrNominate

*RTS_RESULT IoMgrNominate (IoConfigConnector *pConnector)*

Interface to nominate a device specified by connector. Nomination can be used for example for ProfiNET devices to configure their IP-addresses.

pConnector [IN]
Pointer to connector

Result
error code

13.2.131 IoMgrWatchdogTrigger

*RTS_RESULT IoMgrWatchdogTrigger (IoConfigConnector *pConnector)*

If enabled in the device description, this function is called periodically to trigger some kind of hardware watchdog.

pConnector [IN]
Pointer to the connector of the device

Result
error code

13.2.132 IoMgrConfigGetParameter

IoConfigParameter IoMgrConfigGetParameter (IoConfigConnector *pConnector, RTS_UI32 dwParameterId)*

Interface to get a parameter on a specified connector specified by ID

pConnector [IN]
Pointer to connector

dwParameterId [IN]
ID of the parameter. Is defined in the device description.

Result
Parameter or NULL if failed

13.2.133 IoMgrConfigGetParameterValueDword

*RTS_UI32 IoMgrConfigGetParameterValueDword (IoConfigParameter *pParameter, RTS_RESULT *pResult)*

Interface to get the DWORD value out of the specified parameter

pParameter [IN]
Pointer to the parameter

pResult [OUT]
Pointer to error code

Result
Value of the parameter or 0 if failed. Please check always pResult additionally!

13.2.134 IoMgrConfigGetParameterValueWord

*unsigned short IoMgrConfigGetParameterValueWord (IoConfigParameter *pParameter, RTS_RESULT *pResult)*

Interface to get the WORD value out of the specified parameter

pParameter [IN]
Pointer to the parameter
pResult [OUT]
Pointer to error code
Result
Value of the parameter or 0 if failed. Please check always pResult additionally!

13.2.135 IoMgrConfigGetParameterValueByte

*unsigned char IoMgrConfigGetParameterValueByte (IoConfigParameter *pParameter, RTS_RESULT *pResult)*

Interface to get the BYTE value out of the specified parameter

pParameter [IN]
Pointer to the parameter
pResult [OUT]
Pointer to error code
Result
Value of the parameter or 0 if failed. Please check always pResult additionally!

13.2.136 IoMgrConfigGetParameterValuePointer

*void * IoMgrConfigGetParameterValuePointer (IoConfigParameter *pParameter, RTS_RESULT *pResult)*

Interface to get the POINTER to the value out of the specified parameter

pParameter [IN]
Pointer to the parameter
pResult [OUT]
Pointer to error code
Result
Pointer to the value or NULL if failed. Please check always pResult additionally!

13.2.137 IoMgrConfigGetFirstConnector

IoConfigConnector IoMgrConfigGetFirstConnector (IoConfigConnector *pConnectorList, int *pnCount, unsigned short wType)*

Get the first connector with the specified wType in the connector list.

pnCount is decreased and contains the rest of elements still remaining in list.

pConnectorList [IN]
Pointer to connector list
pnCount [INOUT]
Pointer to number of elements in list
wType [IN]
Type of the connector
Result
Pointer to the connector or NULL if not found

13.2.138 IoMgrConfigGetNextConnector

IoConfigConnector IoMgrConfigGetNextConnector (IoConfigConnector *pConnectorList, int *pnCount, unsigned short wType)*

Get the next connector with the specified wType in the connector list.

pnCount is decreased and contains the rest of elements still remaining in list.

pConnectorList [IN]
Pointer to connector list
pnCount [INOUT]
Pointer to number of elements in list

wType [IN]
Type of the connector
Result
Pointer to the connector or NULL if not found

13.2.139 IoMgrConfigGetFirstChild

IoConfigConnector IoMgrConfigGetFirstChild (IoConfigConnector *pConnectorList, int *pnCount, IoConfigConnector *pFather)*

Get the first child connector of the specified father connector.

pnCount is decreased and contains the rest of elements still remaining in list.

pConnectorList [IN]
Pointer to connector list
pnCount [INOUT]
Pointer to number of elements in list
pFather [IN]
Pointer to the father connector
Result
Pointer to the child connector or NULL if not found

13.2.140 IoMgrConfigGetNextChild

IoConfigConnector IoMgrConfigGetNextChild (IoConfigConnector *pConnectorList, int *pnCount, IoConfigConnector *pFather)*

Get the next child connector of the specified father connector.

pnCount is decreased and contains the rest of elements still remaining in list.

pConnectorList [IN]
Pointer to connector list
pnCount [INOUT]
Pointer to number of elements in list
pFather [IN]
Pointer to the father connector
Result
Pointer to the child connector or NULL if not found

13.2.141 IoMgrConfigGetConnectorList

*RTS_RESULT IoMgrConfigGetConnectorList (IoConfigConnector **ppConnectorList, int *pnCount)*

Get the actual IO-configuration in form of the connector list.

If the parameter ppConnectorList is NULL, only the size of the list will be returned.

ppConnectorList [OUT]
Pointer to connector list
pnCount [INOUT]
Number of elements in the list
Result
error code

13.2.142 IoMgrConfigGetConnector

IoConfigConnector IoMgrConfigGetConnector (IoConfigConnector *pConnectorList, int *pnCount, RTS_UI32 ulModuleType, RTS_UI32 ulInstance)*

Get the connector specified by ModuleType and ModuleInstance number

pnCount is decreased and contains the rest of elements, still remaining in list!

Might be called with NULL as Connectorlist, then the Connectorlist and pnCount from the last call to UpdateConfiguration is used

Returns NULL if pnCount is NULL, if ulInstance is greater than nCount, no fitting Connector with given Instance and Type is found or (if pConnectorList is Null, the last stored ConnectorList is also NULL)

pConnectorList [IN]
Pointer to connector list
pnCount [INOUT]
Number of elements in the list
ulModuleType [IN]
Module type
ulInstance [IN]
Instance number
Result
Pointer to found connector or NULL

13.2.143 IoMgrConfigSetDiagnosis

*RTS_RESULT IoMgrConfigSetDiagnosis (IoConfigConnector *pConnector, RTS_UI32 ulFlags)*

Interface to set the diagnostic flags in the connector.

Only the bits that are specified with 1 in the passed ulFlags parameter will be set.

pConnector [IN]
Pointer to connector
ulFlags [IN]
Flags to write
Result
error code

13.2.144 IoMgrConfigResetDiagnosis

*RTS_RESULT IoMgrConfigResetDiagnosis (IoConfigConnector *pConnector, RTS_UI32 ulFlags)*

Interface to reset the diagnostic flags in the connector.

Only the bits that are specified with 1 in the passed ulFlags parameter will be resetted.

pConnector [IN]
Pointer to connector
ulFlags [IN]
Flags to write
Result
error code

13.2.145 IoMgrReadParameter

*RTS_RESULT IoMgrReadParameter (RTS_UI32 ulModuleType, RTS_UI32 ulInstance, RTS_UI32 ulParameterId, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Interface to read a specified parameter.

This interface is optional and may not be implemented by the runtime system.

ulModuleType [IN]
Module type
ulInstance [IN]
Instance number
ulParameterId [IN]
ID of the parameter. Is defined in the device description.
pData [IN]
Pointer to read in the parameter value
ulBitSize [IN]
Bits to read
ulBitOffset [IN]
Bitoffset of the parameter value
Result

error code

13.2.146 IoMgrWriteParameter

*RTS_RESULT IoMgrWriteParameter (RTS_UI32 ulModuleType, RTS_UI32 ulInstance, RTS_UI32 ulParameterId, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Interface to write a specified parameter.

This interface is optional and may not be implemented by the runtime system.

ulModuleType [IN]
Module type
ulInstance [IN]
Instance number
ulParameterId [IN]
ID of the parameter. Is defined in the device description.
pData [IN]
Pointer to the parameter write value
ulBitSize [IN]
Bits to write
ulBitOffset [IN]
Bitoffset of the parameter value
Result
error code

13.2.147 IoMgrConfigGetDriver

*IBase * IoMgrConfigGetDriver (IoConfigConnector *pConnector, RTS_RESULT *pResult)*

Return the registered driver interface of a connector

pConnector [IN]
Pointer to connector
pResult [OUT]
Pointer to error code
Result
Pointer to IBase interface

13.2.148 IoMgrConfigGetConnectorByDriver

*IoConfigConnector * IoMgrConfigGetConnectorByDriver (IBase *pIBase, int nIndex, RTS_RESULT *pResult)*

Interface to get the connector of the driver specified by IBase interface

Note: This function is optional and not supported by CmpIoMgrEmbedded!

pIBase [IN]
Pointer to IBase interface of the driver
nIndex [IN]
Index of the connector
pResult [OUT]
Pointer to error code
Result
Pointer to connector

13.2.149 IoMgrCopyInputLE

*void IoMgrCopyInputLE (IoConfigChannelMap *pChannel, char *pAddress)*

Copy an Input from the Device to some local memory and swap to little endian if necessary.

pChannel [IN]
Pointer to one Channel Mapping
pAddress [IN]
Pointer to the Source Address

13.2.150 IoMgrCopyInputBE

*void IoMgrCopyInputBE (IoConfigChannelMap *pChannel, char *pAddress)*

Copy an Input from the Device to some local memory and swap to big endian if necessary

pChannel [IN]
Pointer to one Channel Mapping
pAddress [IN]
Pointer to the Source Address

13.2.151 IoMgrCopyOutputLE

*void IoMgrCopyOutputLE (IoConfigChannelMap *pChannel, char *pAddress)*

Copy an Output from local memory to the Device and swap to little endian if necessary

pChannel [IN]
Pointer to one Channel Mapping
pAddress [IN]
Pointer to the Destination Address

13.2.152 IoMgrCopyOutputBE

*void IoMgrCopyOutputBE (IoConfigChannelMap *pChannel, char *pAddress)*

Copy an Output from local memory to the Device and swap to big endian if necessary

pChannel [IN]
Pointer to one Channel Mapping
pAddress [IN]
Pointer to the DestinationAddress

14 CmpLogEmbedded

This component support to log all events and data in the runtime system

14.1 CmpLogItf

Interface of the runtime system logger component.

The logger can log runtime system messages in form of strings, together with some describing informations, like message categories and IDs. The messages are saved in a local buffer in the RAM and uploaded to the CoDeSys programming system or an external service tool on demand.

The embedded variant of this component does not support backends and is therefore not able to save the log messages into a file.

Neither the log buffer in RAM, nor the communication medium are safe. Therefore one can not essentially rely on the content of the log messages. They can only be used for analytical purposes.

For runtimes with the define RTS_SIL2 defined, the component may be a bit more limited. For example, no events are supported.

14.1.1 Define: LT_HIGHSPEED

Category: Log types
Type:
Define: LT_HIGHSPEED
Key: UINT32_C
Types of a logger instance

14.1.2 Define: LOG_NONE

Category: Log class/filter
Type:
Define: LOG_NONE
Key: UINT32_C
Log entry classes and filters

14.1.3 Define: LOG_ALL

Category: Log filter
Type:
Define: LOG_ALL
Key: UINT32_MAX
Log entry classes and filters

14.1.4 Define: USERDB_OBJECT_LOGGER

Category: Static defines
Type:
Define: USERDB_OBJECT_LOGGER
Key: Device.Logger
Predefined objects in the runtime

14.1.5 Define: LOG_STD_MAX_NUM_OF_ENTRIES

Condition: #ifndef LOG_STD_MAX_NUM_OF_ENTRIES
Category: Static defines
Type:
Define: LOG_STD_MAX_NUM_OF_ENTRIES
Key: 500
Default maximimum number of log entries in a logger instance

14.1.6 Define: LOG_STD_MAX_NUM_OF_FILES

Condition: #ifndef LOG_STD_MAX_NUM_OF_FILES
Category: Static defines
Type:
Define: LOG_STD_MAX_NUM_OF_FILES
Key: 3
Default maximimum number of files for a logger with a file backend

14.1.7 Define: LOG_STD_MAX_FILE_SIZE

Condition: #ifndef LOG_STD_MAX_FILE_SIZE

Category: Static defines
Type:
Define: LOG_STD_MAX_FILE_SIZE
Key: 100000
Default maximimum file size for a logger with a file backend

14.1.8 Define: LOG_MAX_INFO_LEN

Condition: #ifndef LOG_MAX_INFO_LEN
Category: Static defines
Type:
Define: LOG_MAX_INFO_LEN
Key: 128
Maximimum lenght of the info string in a logger entry

14.1.9 Define: LOG_USE_SYSINTDISABLEALL_DEF

Condition: #ifndef LOG_USE_SYSINTDISABLEALL_DEF
Category: Static defines
Type:
Define: LOG_USE_SYSINTDISABLEALL_DEF
Key: 0
The method used to protect the logger against reentrance.

14.1.10 Define: EVT_LogAdd

Category: Events
Type:
Define: EVT_LogAdd
Key: MAKE_EVENTID
Event is sent, after a new log entry added to the logger

14.1.11 Define: LOG_GET_ENTRIES

Category: Online services
Type:
Define: LOG_GET_ENTRIES
Key: 0x01
Get all log entries

14.1.12 Define: LOG_GET_COMPONENT_NAMES

Category: Online services
Type:
Define: LOG_GET_COMPONENT_NAMES
Key: 0x02
Get component name specified by component id

14.1.13 Define: LOG_GET_LOGGER_LIST

Category: Online services
Type:
Define: LOG_GET_LOGGER_LIST
Key: 0x03
Get all registered logger names

14.1.14 Define: TAG_LOGGER_NAME

Category: Online service tags
Type:
Define: TAG_LOGGER_NAME
Key: 0x01

14.1.15 Typedef: EVTPARAM_CmpLogAdd

Stuctname: EVTPARAM_CmpLogAdd
Category: Event parameter
Typedef: typedef struct { RTS_HANDLE hLog; CMPID CmpId; RTS_I32 iClassID; RTS_RESULT iErrorID; RTS_I32 iInfoID; char *pszInfo; va_list *pargList; } EVTPARAM_CmpLogAdd;

14.1.16 Typedef: LogOptions

Stuctname: LogOptions
Category: Logger

Options of a logger (part of the configuration).

Typedef: typedef struct tagLogOptions { char szName[32]; RTS_I32 bEnable; RTS_I32 iType; RTS_I32 iFilter; RTS_I32 iMaxEntries; RTS_I32 iMaxFileSize; RTS_I32 iMaxFiles; char *pszPath; } LogOptions;

14.1.17 Typedef: LogEntry

Stuctname: LogEntry
Category: Logger

One logger Entry, including the message.

Typedef: typedef struct tagLogEntry { LogTimestamp tTimestamp; CMPID CmpId; RTS_UI32 iClassID; RTS_RESULT iErrorID; RTS_UI32 iInfoID; char szInfo[LOG_MAX_INFO_LEN]; } LogEntry;

14.1.18 Typedef: LogItf

Stuctname: LogItf
Category: Logger

Backend interface of one logger.

Typedef: typedef struct { ICmpLogBackend *pIBackend; CLASSID ClassId; RTS_HANDLE hLogBackend; RTS_I32 iLastDumpedIndex; } LogItf;

14.1.19 Typedef: Logger

Stuctname: Logger
Category: Logger

Configuration of one logger.

Typedef: typedef struct { LogOptions lo; RTS_UI32 ulStartTimestamp; RTS_I32 iIndex; RTS_I32 iFirstIndex; RTS_I32 iRegInterfaces; RTS_I32 iDumpSync; LogEntry *pLog; LogItf tLogItf[LOG_DEFAULT_NUM_OF_ITF]; } Logger;

14.1.20 LogCreate

*RTS_HANDLE LogCreate (LogOptions *pOptions, RTS_RESULT *pResult)*

Create a logger

pOptions [IN]
Options for logger

pResult [OUT]
Pointer to get the result

Result
Handle to the logger, or RTS_INVALID_HANDLE if failed

14.1.21 LogOpen

*RTS_HANDLE LogOpen (char *pszName, RTS_RESULT *pResult)*

Open a logger with the specified name. Logger must exist!

pszName [IN]
Logger name

pResult [OUT]
Pointer to get the result

Result
Handle to the logger, or RTS_INVALID_HANDLE if logger does not exist

14.1.22 LogClose

RTS_RESULT LogClose (RTS_HANDLE hLog)

Close the handle to a logger

hLog [IN]
Handle to logger

Result
error code

14.1.23 LogDelete

RTS_RESULT LogDelete (RTS_HANDLE hLog)

Delete a logger

hLog [IN]

Handle to logger

Result
error code

14.1.24 LogGetOptions

*RTS_RESULT LogGetOptions (RTS_HANDLE hLog, LogOptions **ppOptions)*

Get options of logger

hLog [IN]
Handle to logger

ppOptions [OUT]
Pointer to pointer to log options

Result
error code

14.1.25 LogEnable

RTS_RESULT LogEnable (RTS_HANDLE hLog)

Enable logging

hLog [IN]
Handle to logger

Result
error code

14.1.26 LogDisable

RTS_RESULT LogDisable (RTS_HANDLE hLog)

Disable logging

hLog [IN]
Handle to logger

Result
error code

14.1.27 LogSetFilter

RTS_RESULT LogSetFilter (RTS_HANDLE hLog, RTS_I32 iFilter)

Set filter of logger

hLog [IN]
Handle to logger

iFilter [IN]
Logger filter

Result
error code

14.1.28 LogGetFilter

RTS_I32 LogGetFilter (RTS_HANDLE hLog)

Get filter of logger

hLog [IN]
Handle to logger

Result
Filter

14.1.29 LogAdd

*RTS_RESULT LogAdd (RTS_HANDLE hLog, CMPID CmpId, RTS_I32 iClassID, RTS_RESULT iErrorID, RTS_I32 iInfoID, char *pszInfo, ...)*

Add a new log entry to the log buffer.

If the buffer is full when this function is called, the oldest log entry in the buffer will be overwritten.

If the Class ID contains LOG_INFO_TIMESTAMP_RELATIVE, there is an additional tag, called "TimeRel" added to the text of the log entry. This will limit the message size of the entry by the size of this tag.

If the Class ID contains the flag LOG_USER_NOTIFY, the log message will be shown in form of a message box at the next log in of CoDeSys or instantly if CoDeSys is still loged in.

The interface supports a minimum of 8 variable arguments. Depending on the C-Library, this might be more.

LT_TIMESTAMP_RTC, LT_TIMESTAMP_RTC_HIGHRES is not supported in SIL2 Runtime.

hLog [IN]
Handle to logger
CmpId [IN]
Component id
iClassID [IN]
ClassID of entry (Info, Warning, Error, etc.)
iErrorID [IN]
Error code if available
iInfoID [IN]
ID of info text to enable multiple language error texts
pszInfo [IN]
String to info text (in english or informations coded in XML)
Result
error code

14.1.30 LogAddArg

*RTS_RESULT LogAddArg (RTS_HANDLE hLog, CMPID CmpId, RTS_I32 iClassID, RTS_RESULT iErrorID, RTS_I32 iInfoID, char *pszInfo, va_list *pargList)*

Add a new log entry to the log buffer.

The behavior is the same as the behavior of "LogAdd()", except that the parameter is a pointer to a variable argument list instead of a direct variable argument list, passed to the function.

hLog [IN]
Handle to logger
CmpId [IN]
Component id
iClassID [IN]
ClassID of entry (Info, Warning, Error, etc.)
iErrorID [IN]
Error code if available
iInfoID [IN]
ID of info text to enable multiple language error texts
pszInfo [IN]
String to info text (in english or informations coded in XML)
pargList [IN]
Pointer to argument list, format is specified in pszInfo
Result
error code

14.1.31 LogDumpAll

RTS_RESULT LogDumpAll (int iOptions)
Dump all entries from all log files
iOptions [IN]

One or multiple of the following options:

- LT_DUMP_ASYNC: If dump is done from an asychronous event
- LT_DUMP_ALWAYS: If dump should be forced (always)
- LT_DUMP_ON_CLOSE: If dump is called from closing the logger instance
- LT_DUMP_ON_REQUEST: If dump is forced from a request

Result
error code

14.1.32 LogDumpEntries

RTS_RESULT LogDumpEntries (RTS_HANDLE hLog)
Dump all entries from the last still dumped entry
hLog [IN]
Handle to the logger
Result
error code

14.1.33 LogRegisterInterface

*RTS_RESULT LogRegisterInterface (RTS_HANDLE hLog, CLASSID ClassId, ICmpLogBackend *pIBackend)*
Register a new logger interface
hLog [IN]
Handle to logger
ClassId [IN]
ClassID of the backend component
pIBackend [IN]
Pointer to the backend interface
Result
error code

14.1.34 LogUnregisterInterface

*RTS_RESULT LogUnregisterInterface (RTS_HANDLE hLog, ICmpLogBackend *pIBackend)*
Unregister a logger interface
hLog [IN]
Handle to logger
pIBackend [IN]
Pointer to the backend interface
Result
error code

14.1.35 LogGetEntryByIndex

*RTS_HANDLE LogGetEntryByIndex (RTS_HANDLE hLog, int iIndex, LogEntry *pLogEntry, RTS_RESULT *pResult)*
Get the first logentry of a logger
hLog [IN]
Handle to logger
iIndex [IN]
Index of entry to get. 0 is the first entry.
pLogEntry [IN]
Pointer to log entry
pResult [IN]
Pointer to result
Result
Handle to next log entry or RTS_INVALID_HANDLE, if end of logger is reached

14.1.36 LogGetEntryByQueueIndex

*RTS_HANDLE LogGetEntryByQueueIndex (RTS_HANDLE hLog, int iQueueIndex, LogEntry *pLogEntry, RTS_RESULT *pResult)*
Get the first logentry of a logger
hLog [IN]
Handle to logger
iQueueIndex [IN]
Index of entry to get. -1 get the first entry.
pLogEntry [IN]
Pointer to log entry
pResult [IN]
Pointer to result
Result

Handle to next log entry or RTS_INVALID_HANDLE, if end of logger is reached

14.1.37 LogGetFirstEntry

*RTS_HANDLE LogGetFirstEntry (RTS_HANDLE hLog, LogEntry *pLogEntry, RTS_RESULT *pResult)*

Get the first logentry of a logger

hLog [IN]
Handle to logger

pLogEntry [IN]
Pointer to log entry

pResult [IN]
Pointer to result

Result
Handle to next log entry or RTS_INVALID_HANDLE, if end of logger is reached

14.1.38 LogGetNextEntry

*RTS_HANDLE LogGetNextEntry (RTS_HANDLE hLog, RTS_HANDLE hEntry, LogEntry *pLogEntry, RTS_RESULT *pResult)*

Get the next logentry of a logger

hLog [IN]
Handle to logger

hEntry [IN]
Handle to log entry (is returned by LogGetFirstEntry or LogGetNextEntry)

pLogEntry [IN]
Pointer to log entry

pResult [IN]
Pointer to result

Result
Handle to next log entry or RTS_INVALID_HANDLE, if end of logger is reached

14.1.39 LogGetEntry

*RTS_RESULT LogGetEntry (RTS_HANDLE hLog, RTS_HANDLE hEntry, LogEntry *pLogEntry)*

Get an entry specified by handle

hLog [IN]
Handle to logger

hEntry [IN]
Handle to log entry (is returned by LogGetFirstEntry or LogGetNextEntry)

pLogEntry [IN]
Pointer to log entry

Result
error code

14.1.40 LogGetFirstLogger

*RTS_HANDLE LogGetFirstLogger (RTS_RESULT *pResult)*

Get the first registered logger

pResult [OUT]
Pointer to error code

Result
Handle to the first logger

14.1.41 LogGetNextLogger

*RTS_HANDLE LogGetNextLogger (RTS_HANDLE hLogger, RTS_RESULT *pResult)*

Get the next registered logger

hLogger [IN]
Handle to previous logger

pResult [OUT]
Pointer to error code

Result

Handle to the first logger

14.1.42 LogGetName

*RTS_RESULT LogGetName (RTS_HANDLE hLog, char *pszLoggerName, int nMaxLen)*

Get the logger name of the logger specified by handle

hLogger [IN]
Handle to the logger

pszLogger [INOUT]
Pointer to logger name

nMaxLen [IN]
Max length of pszLogger

Result
error code

14.1.43 LogGetUserNotify

*RTS_RESULT LogGetUserNotify (LogEntry **ppLogEntryUserNotify)*

Get the last log entry of class LOG_USER_NOTIFY

ppLogEntryUserNotify [OUT]
Returns the pointer to the user notify entry

Result
Error code:
- ERR_OK: There is still an unread log entry of the type LOG_USER_NOTIFY
- ERR_NO_OBJECT: No pending log entry of the type LOG_USER_NOTIFY

15 CmpMemPool

System component that provides heap memory access.

Compiler Switch

- #define MEMPOOL_DISABLE_HEAP_MEMORY Switch to disable dynamic memory

15.1 CmpMemPoolItf

Interface of the memory pool manager to handle static and dynamic memory blocks

A MemPool has the following structure:

. . ---------- Pool Control Block ---------- Block Control Block . |Blocksize | data size of single block |nRefCount | F

15.1.1 Typedef:

Typedef: typedef struct tagRTS_MEM_LIST_POOL RTS_MEM_LIST_POOL; typedef struct tagRTS_MEM_LIST_POOL_BLOCK RTS_MEM_LIST_POOL_BLOCK; typedef struct tagRTS_MEM_LIST_ELEMENT RTS_MEM_LIST_ELEMENT; Single-linked list element. struct tagRTS_MEM_LIST_ELEMENT { RTS_MEM_LIST_ELEMENT* next; };

15.1.2 MemPoolCreateDynamic

*RTS_HANDLE MemPoolCreateDynamic (char *pszComponentName, RTS_SIZE ulNumBlocks, RTS_SIZE ulBlockSize, RTS_RESULT *pResult)*

Create a dynamic pool (consists of heap memory)

pszComponentName [IN]
Component name

ulBlockSize [IN]
Size of each memory block in the pool

pResult [OUT]
Pointer to error code

Result
Handle to the memory pool

15.1.3 MemPoolCreateStatic

RTS_HANDLE MemPoolCreateStatic (RTS_SIZE ulBlockSize, RTS_SIZE ulMemSize, void pMemory, RTS_RESULT *pResult)*

Create a memory pool from a static memory buffer.

The memory buffer don't has to be aligned in a specific way. Therefore, not all of the memory in the buffer might be used. To get the appropriate additional buffer, the caller is recommended to use the macro MEM_GET_STATIC_LEN(Num, Struct) to get the size of the buffer

For example:

typedef struct { ... } myStruct_s; #define NUM_OF_STATIC_ELEMENTS 0x100 RTS_UI8 s_byMyStaticPool[MEM_G

ulBlockSize [IN]
Size of each memory block in the pool, misaligned to 1, 2, 4 or 8 bytes

ulMemSize [IN]
Complete size of the static memory, misaligned to 1, 2, 4 or 8 bytes

pMemory [IN]
Pointer to the static memory

pResult [OUT]
Pointer to error code

Result
Handle to the memory pool

15.1.4 MemPoolExtendDynamic

*RTS_RESULT MemPoolExtendDynamic (RTS_HANDLE hMemPool, char *pszComponentName, RTS_SIZE ulNumBlocks)*

Extend dynamic an existing pool

hMemPool [IN]
Handle to the pool

pszComponentName [IN]
Component name

ulNumBlocks [IN]
Number of blocks to extend

Result
error code

15.1.5 MemPoolExtendStatic

RTS_RESULT MemPoolExtendStatic (RTS_HANDLE hMemPool, RTS_SIZE ulMemSize, void pMemory)*

Extend an existing pool with a static array

hMemPool [IN]
Handle to the pool

ulMemSize [IN]
Complete size of the static memory

pMemory [IN]
Pointer to the static memory

Result
error code

15.1.6 MemPoolCreateSyncObject

RTS_RESULT MemPoolCreateSyncObject (RTS_HANDLE hMemPool)

Create the internal sync object for synchronizing the pool.

hMemPool [IN]
Handle to the pool

Result
error code

15.1.7 MemPoolDeleteSyncObject

RTS_RESULT MemPoolDeleteSyncObject (RTS_HANDLE hMemPool)

Delete the internal sync object for synchronizing the pool.

hMemPool [IN]
Handle to the pool

Result
error code

15.1.8 MemPoolDelete

*RTS_RESULT MemPoolDelete (RTS_HANDLE hMemPool, char *pszComponentName)*

Delete an existing pool

hMemPool [IN]
Handle to the pool

pszComponentName [IN]
Component name

Result
error code

15.1.9 MemPoolCleanup

*RTS_RESULT MemPoolCleanup (RTS_HANDLE hMemPool, char *pszComponentName, int bReleaseSemaphore)*

Cleanup the pool (delete all allocated heap pool objects)

hMemPool [IN]
Handle to the pool

pszComponentName [IN]
Component name

bReleaseSemaphore [IN]
1=Pool semaphore is released, 0=Only cleanup
Result
error code

15.1.10 MemPoolGetBlock

void MemPoolGetBlock (RTS_HANDLE hMemPool, RTS_RESULT *pResult)*

Get one memory block out of the pool.

SIL2 Implementation: If pPCB is wrong, an Exception is generated!

hMemPool [IN]
Handle to the pool
pResult [OUT]
Pointer to error code
Result
Pointer to the memory block

15.1.11 MemPoolGetBlock2

void MemPoolGetBlock2 (RTS_HANDLE hMemPool, int bDynamic, char *pszComponentName, RTS_RESULT *pResult)*

Get one memory block out of the pool
hMemPool [IN]
Handle to the pool
bDynamic [IN]
1=Block is created dynamically, if the pool is empty, 0=Only use of static pool memory
pszComponentName [IN]
Pointer to the component name for dynamic memory allocation
pResult [OUT]
Pointer to error code
Result
Pointer to the memory block

15.1.12 MemPoolGetPCB

RTS_PCB MemPoolGetPCB (RTS_HANDLE hMemPool, RTS_RESULT *pResult)*

Get one the pool control block of a specified pool
hMemPool [IN]
Handle to the pool
pResult [OUT]
Pointer to error code
Result
Pointer to the pool control block

15.1.13 MemPoolPutBlock

RTS_RESULT MemPoolPutBlock (void pBlock)*

Put a memory block back into the pool (release). Now, the block is in the chain list of free blocks again.
pBlock [IN]
Pointer to the memory block
Result
error code

15.1.14 MemPoolAddUsedBlock

RTS_RESULT MemPoolAddUsedBlock (void pBlock)*

Add used block at the beginning of the pool. Now, the block is in the chain list of used blocks.
pBlock [IN]
Pointer to the memory block
Result

error code

15.1.15 MemPoolAppendUsedBlock

RTS_RESULT MemPoolAppendUsedBlock (void pBlock)*

Add used block at the end of the pool. Now, the block is in the chain list of used blocks.

pBlock [IN]
Pointer to the memory block

Result
error code

15.1.16 MemPoolInsertUsedBlock

RTS_RESULT MemPoolInsertUsedBlock (void pPrevBlock, void* pBlock)*

Insert used block right after the specified block or as a head element of an internal used blocks list.

pPrevBlock [IN]
Pointer to the predecessor block

pBlock [IN]
Pointer to the memory block

Result
error code

15.1.17 MemPoolRemoveUsedBlock

RTS_RESULT MemPoolRemoveUsedBlock (void pBlock)*

Remove used block from the pool. Now, the block is removed from the chain list of used blocks.

pBlock [IN]
Pointer to the memory block

Result
error code

15.1.18 MemPoolAddUsedBlockToPool

RTS_RESULT MemPoolAddUsedBlockToPool (void pBlock, RTS_HANDLE hPool)*

Add used block the beginning of the specified pool. Now, the block is in the chain list of used blocks.

pBlock [IN]
Pointer to the memory block

hPool [IN]
Handle to the pool

Result
error code

15.1.19 MemPoolAppendUsedBlockToPool

RTS_RESULT MemPoolAppendUsedBlockToPool (void pBlock, RTS_HANDLE hPool)*

Add the block to the end of the used block list of hPool

pBlock [IN]
Pointer to the memory block

hPool [IN]
Handle to the pool

Result
error code

15.1.20 MemPoolRemoveUsedBlockFromPool

RTS_RESULT MemPoolRemoveUsedBlockFromPool (void pBlock, RTS_HANDLE hPool)*

Remove used block from the specified pool. Now, the block is removed from the chain list of used blocks.

pBlock [IN]
Pointer to the memory block

hPool [IN]
Handle to the pool

Result
error code

15.1.21 MemPoolLockBlock

RTS_RESULT MemPoolLockBlock (void pBlock)*

Lock the access to a pool to be threadsafe.

pBlock [IN]
Pointer to the memory block

Result
error code

15.1.22 MemPoolUnlockBlock

RTS_RESULT MemPoolUnlockBlock (void pBlock)*

Unlock the access to a pool.

pBlock [IN]
Pointer to the memory block

Result
error code

15.1.23 MemPoolLock

RTS_RESULT MemPoolLock (RTS_HANDLE hMemPool)

Lock the access to the complete pool. SIL2 Implementation is using INT Locks.

hMemPool [IN]
Handle to the memory pool

Result
error code

15.1.24 MemPoolUnlock

RTS_RESULT MemPoolUnlock (RTS_HANDLE hMemPool)

Unlock the access to the complete pool. SIL2 Implementation is using INT Locks.

hMemPool [IN]
Handle to the memory pool

Result
Error code

15.1.25 MemPoolFindBlock

*void * MemPoolFindBlock (RTS_HANDLE hMemPool, RTS_SIZE ulOffset, RTS_SIZE ulSize, void *pToFind, RTS_RESULT *pResult)*

Find a block specified by a value, that is stored in the block.

hMemPool [IN]
Handle to the memory pool

ulOffset [IN]
Byte offset of the value in the block to find

ulSize [IN]
Size in bytes of the value to find in the block

pToFind [IN]
Pointer to the value to find in the block

pResult [OUT]
Pointer to error code

Result
Pointer to the memory block

15.1.26 MemPoolIsValidBlock

*RTS_RESULT MemPoolIsValidBlock (RTS_HANDLE hMemPool, void *pBlock)*

Check a pool memory block, if it is still valid and is not released. NOTE: If the check is successful, a lock is done on this pool!!! So you have to unlock this reference at the end of the usage with MemPoolUnlock()!

hMemPool [IN]
Handle to the memory pool

pBlock [IN]

Pointer to the memory block

Result

Error code

15.1.27 FixedBlocksPoolInit

RTS_RESULT FixedBlocksPoolInit (RTS_FIXED_BLOCKS_POOL pPool, RTS_MEM_REGION* pRegion, RTS_SIZE block_size, RTS_SIZE blocks_number)*

Initializes a pool for allocating objects of fixed size

pPool [IN]

Pointer to a pool object

pRegion [IN]

Pointer to a region of memory that will be used for allocation

block_size [IN]

Size (in bytes) of a memory block

blocks_number [IN]

Number of blocks that should be located in a contiguous chunk

Result

Error code

15.1.28 FixedBlocksPoolAlloc

void FixedBlocksPoolAlloc (RTS_FIXED_BLOCKS_POOL* pPool, RTS_SIZE size, RTS_RESULT* pResult)*

Allocates a memory block out of the pool

pPool [IN]

Pointer to a pool object

size [IN]

Requested size of memory to be allocated. Should be less than the pPool block size

pResult [OUT]

Pointer to an error code

Result

Pointer to the allocated block of memory

15.1.29 FixedBlocksPoolFree

RTS_RESULT FixedBlocksPoolFree (RTS_FIXED_BLOCKS_POOL pPool, void* pMem)*

Puts a memory block back to the pool.

pPool [IN]

Pointer to a pool object

pMem [IN]

Pointer to a memory block to be returned to the pool

Result

error code

15.1.30 FixedBlocksPoolReclaim

RTS_RESULT FixedBlocksPoolReclaim (RTS_FIXED_BLOCKS_POOL pPool)*

Puts all the previously allocated memory blocks back to the pool.

pPool [IN]

Pointer to a pool object

Result

error code

15.1.31 FixedBlocksPoolDestroy

RTS_RESULT FixedBlocksPoolDestroy (RTS_FIXED_BLOCKS_POOL pPool)*

Destroys a pool object.

pPool [IN]

Pointer to a pool object

Result

error code

16 CmpMonitor

This component realizes the monitoring of IEC variable values. This component handles communication service of the group "SG_MONITORING". Therefore, the component registers a handler function at the level 7 server component CmpSrv. The component supports monitoring of simple datatypes, including Bits.

16.1 CmpMonitorItf

Interface of the monitoring component, that provides monitoring of IEC variables.

16.1.1 Define: SRV_MONITORLISTONCE

Category: Online services
Type:
Define: SRV_MONITORLISTONCE
Key: 1

16.1.2 Define: MONITORING_FEATURE_COMPLEX_MONITORING

Category: Features
Type: Int
Define: MONITORING_FEATURE_COMPLEX_MONITORING
Key: 0x00000001
Supported features of the monitoring component

16.1.3 Define: EVT_PrepareWriteVariable

Category: Events
Type:
Define: EVT_PrepareWriteVariable
Key: MAKE_EVENTID
Event is sent to prepare the write operation to a variable

16.1.4 Define: EVT_WriteVariableDone

Category: Events
Type:
Define: EVT_WriteVariableDone
Key: MAKE_EVENTID
Event is sent after the write operation to a variable was done

16.1.5 Define: EVT_PrepareForceVariable

Category: Events
Type:
Define: EVT_PrepareForceVariable
Key: MAKE_EVENTID
Event is sent to prepare the force of variable

16.1.6 Define: EVT_ForceVariableDone

Category: Events
Type:
Define: EVT_ForceVariableDone
Key: MAKE_EVENTID
Event is sent after the force operation of a variable was done

16.1.7 Typedef: EVTPARAM_CmpMonitorWriteVar

Stuctname: EVTPARAM_CmpMonitorWriteVar
Category: Event parameter
Typedef: typedef struct { RTS_UI16 usSize; RTS_UI16 dummy; #ifdef TRG_64BIT RTS_UI32 dummy2; #endif RTS_UI8* pAddress; void* pValue; } EVTPARAM_CmpMonitorWriteVar;

16.1.8 Typedef: EVTPARAM_CmpMonitorWriteVar2

Stuctname: EVTPARAM_CmpMonitorWriteVar2
Category: Event parameter
Typedef: typedef struct { RTS_UI16 usSize; RTS_UI16 dummy; #ifdef TRG_64BIT RTS_UI32 dummy2; #endif RTS_UI8* pAddress; void* pValue; RTS_I32 bDeny; CMPID cmpId; } EVTPARAM_CmpMonitorWriteVar2;

16.1.9 Typedef: EVTPARAM_CmpMonitorForceVar

Stuctname: EVTPARAM_CmpMonitorForceVar
Category: Event parameter
Typedef: typedef struct { RTS_UI16 usForceFlag; RTS_UI16 dummy; #ifdef TRG_64BIT RTS_UI32 dummy2; #endif VarDataRef* pDataRef; APPLICATION* pAppl; } EVTPARAM_CmpMonitorForceVar;

16.1.10 Typedef: EVTPARAM_CmpMonitorForceVar2

Stuctname: EVTPARAM_CmpMonitorForceVar2
Category: Event parameter
Typedef: typedef struct { RTS_UI16 usForceFlag; RTS_UI16 dummy; #ifdef TRG_64BIT RTS_UI32 dummy2; #endif VarDataRef* pDataRef; APPLICATION* pAppl; RTS_UI8* pAddress; RTS_I32 bDeny; CMPID cmpId; } EVTPARAM_CmpMonitorForceVar2;

16.1.11 MonitoringHasFeature

RTS_RESULT MonitoringHasFeature (unsigned long ulFeatures)

Routine to check, if a scheduler has the specified feature.

ulFeatures [IN]
Feature flags, See corresponding category "Features".

Result
ERR_OK if the flags are supported, an error code otherwise

16.1.12 MonitoringReadValue

RTS_RESULT MonitoringReadValue (APPLICATION pappl, BINTAGREADER* preader, RTS_UI8* pbyValue, RTS_UI16 usSize)*

Routine to read a value via a monitoring service. Note: this function is equivalent to calling MonitoringReadValue2 with bIec = 0.

pappl [IN]
Pointer to application.

preader [IN]
the reader specifying the location of the value to monitor.

pbyValue [IN]
pointer to the destination address were the retrieved value is written.

usSize [IN]
size of value to read.

Result
ERR_OK if the value could be retrieved successfully

16.1.13 MonitoringReadValue2

RTS_RESULT MonitoringReadValue2 (APPLICATION pappl, BINTAGREADER* preader, RTS_UI8* pbyValue, RTS_UI16 usSize, int bIec)*

Routine to read a value via a monitoring service.

pappl [IN]
Pointer to application.

preader [IN]
the reader specifying the location of the value to monitor.

pbyValue [IN]
pointer to the destination address were the retrieved value is written.

usSize [IN]
size of value to read.

bIec [IN]
whether the call is done from an IEC task or not

Result
ERR_OK if the value could be retrieved successfully

16.1.14 MonitoringWriteValue

RTS_RESULT MonitoringWriteValue (APPLICATION pappl, BINTAGREADER* preader, RTS_UI8* pbyValue, RTS_UI16 usSize)*

Routine to writa a value via a monitoring service. Note: this function is equivalent to calling MonitoringWriteValue2 with bIec = 0.

pappl [IN]
Pointer to application.
preader [IN]
the reader specifying the location of the value to write.
pbyValue [IN]
pointer to the source address of the value to write.
usSize [IN]
size of value to read.
Result
ERR_OK if the value could be written successfully

16.1.15 MonitoringWriteValue2

RTS_RESULT MonitoringWriteValue2 (APPLICATION pappl, BINTAGREADER* preader, RTS_UI8* pbyValue, RTS_UI16 usSize, int bIec)*
Routine to writa a value via a monitoring service.
pappl [IN]
Pointer to application.
preader [IN]
the reader specifying the location of the value to write.
pbyValue [IN]
pointer to the source address of the value to write.
usSize [IN]
size of value to read.
bIec [IN]
whether the call is done from an IEC task or not
Result
ERR_OK if the value could be written successfully

17 CmpNameServiceServer

Implements the naming services

17.1 CmpNameServiceServerItf

External interface for the naming service server

18 CmpRouter

Implements the routing of packages on layer 3

18.1 CmpRouterItf

Interface for the communication router component.

18.1.1 Typedef: RTR_AddrComponent

Stuctname: RTR_AddrComponent

RTR_AddrComponent

Typedef: typedef struct tagRTR_AddrComponent { RTS_IEC_BYTE Component[2]; } RTR_AddrComponent;

18.1.2 Typedef: RTR_NodeAddress

Stuctname: RTR_NodeAddress

RTR_NodeAddress

Typedef: typedef struct tagRTR_NodeAddress { RTS_IEC_UDINT nAddrComponentCount; RTR_AddrComponent AddrComponents[15]; } RTR_NodeAddress;

18.1.3 Typedef: routergetinstancebyname_struct

Stuctname: routergetinstancebyname_struct

routergetinstancebyname

Typedef: typedef struct tagroutergetinstancebyname_struct { RTS_IEC_STRING *pstName; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *RouterGetInstanceByName; VAR_OUTPUT } routergetinstancebyname_struct;

18.1.4 Typedef: routergetname_struct

Stuctname: routergetname_struct

routergetname

Typedef: typedef struct tagroutergetname_struct { RTS_IEC_BYTE *hRouter; VAR_INPUT RTS_IEC_BYTE *pBuffer; VAR_INPUT RTS_IEC_INT nBufferSize; VAR_INPUT RTS_IEC_UDINT RouterGetName; VAR_OUTPUT } routergetname_struct;

18.1.5 Typedef: routergethostaddress_struct

Stuctname: routergethostaddress_struct

routergethostaddress

Typedef: typedef struct tagroutergethostaddress_struct { RTS_IEC_BYTE *hRouter; VAR_INPUT RTR_NodeAddress *resAddr; VAR_IN_OUT RTS_IEC_UDINT RouterGetHostAddress; VAR_OUTPUT } routergethostaddress_struct;

18.1.6 Typedef: routergetparentaddress_struct

Stuctname: routergetparentaddress_struct

routergetparentaddress

Typedef: typedef struct tagroutergetparentaddress_struct { RTS_IEC_BYTE *hRouter; VAR_INPUT RTR_NodeAddress *resAddr; VAR_IN_OUT RTS_IEC_UDINT RouterGetParentAddress; VAR_OUTPUT } routergetparentaddress_struct;

18.1.7 RouterRegisterNetworkInterface2

*RTS_RESULT RouterRegisterNetworkInterface2 (NETWORKINTERFACEINFO2 *pInterfaceInfo, RTS_HANDLE * phSubnet)*

Called by a blockdriver to register one of it's devices with the router. Allows also to provide some block driver typ specific information.

pDeviceInfo [IN]

Describes the device to register

phSubnet [OUT]

Is set to the subnet handle that refers to this interface. The blockdriver must provide this value in each call to RouterHandleData or RouterUnregisterNetworkInterface.

Result

error code

18.1.8 RouterRegisterNetworkInterface

*RTS_RESULT RouterRegisterNetworkInterface (NETWORKINTERFACEINFO *pInterfaceInfo, RTS_HANDLE * phSubnet)*

Called by a blockdriver to register one of it's devices with the router.

pDeviceInfo [IN]
Describes the device to register

phSubnet [OUT]
Is set to the subnet handle that refers to this interface. The blockdriver must provide this value in each call to RouterHandleData or RouterUnregisterNetworkInterface.

Result
error code

18.1.9 RouterRegisterOnDemandNWInterface

*RTS_RESULT RouterRegisterOnDemandNWInterface (NETWORKINTERFACEINFO *pInterfaceInfo, ONDEMANDNETWORKINTERFACE *pOnDemandInfo, RTS_HANDLE * phSubnet)*

At the moment pOnDemandInfo IS NOT USED by the CmpRouter! Called by an on-demand blockdriver to register one of it's devices with the router. An on-demand blockdriver is a blockdriver that is able to shutdown/open its connection on demand. Eg. for serial blockdrivers, so they can release their hardware interface as long as it isn't needed and open it only, when their is data to be sent. The blockdriver is supposed to be initially closed.

pDeviceInfo [IN]
Describes the device to register

pOnDemandInfo [IN]
Contains additional functions, that allow for opening and closing of the interface.

phSubnet [OUT]
Is set to the subnet handle that refers to this interface. The blockdriver must provide this value in each call to RouterHandleData or RouterUnregisterNetworkInterface.

Result
error code

18.1.10 RouterUnregisterNetworkInterface

RTS_RESULT RouterUnregisterNetworkInterface (RTS_HANDLE hSubnet)

Called by a blockdriver to unregister one of it's devices

hSubnet [IN]
Subnet handle of the interface, which should be unregistered.

Result
error code

18.1.11 RouterHandleData

RTS_RESULT RouterHandleData (RTS_HANDLE phSubnet, NETWORKADDRESS sender, PROTOCOL_DATA_UNIT pduData, int bIsBroadcast)

Called whenever the blockdriver receives a valid data package

phSubnet [IN]
The subnetid assigned to the receiving device during RouterRegisterDevice

addrSender [IN]
The device address of the sender within the subnet

pduData [IN]
The received data package

Result
error code

18.1.12 RouterGetBlkAddresses

*RTS_RESULT RouterGetBlkAddresses (PROTOCOL_DATA_UNIT pduData, PEERADDRESS *pAddrReceiver, PEERADDRESS *pAddrSender, RTS_I32 *piDataOffset)*

Called by a blockdriver to get the addresses of a data package.

pduData [IN]
Data package, from which the addresses should be read.

pAddrReceiver [OUT]
The function returns here the address of the receiver.

pAddrSender [OUT]

The function returns here the address of the sender.

piDataOffset [OUT]
If not NULL, the function returns here the offset (start) of the pay load. .

Result
error code

18.1.13 RouterCompareAddresses

*RTS_BOOL RouterCompareAddresses (PEERADDRESS *pAddr1, PEERADDRESS *pAddr2)*
Called by a blockdriver to compare two peer addresses.

pAddr1 [IN]
First address to compare

pAddr2 [IN]
Second address to compare

Result
TRUE if the addresses are equal, else FALSE

18.1.14 RouterRegisterProtocolHandler

RTS_RESULT RouterRegisterProtocolHandler (int nProtocolId, PFPHHandleData pfPHHandleData)
A protocol handler calls this function to register itself with the router The handler for the addressresolution protocol uses a specialized interface, thus it is an error to register a handler for either PID_ADDRESSREQUEST or PID_ADDRESSNOTIFICATION with this function.

protocolId [IN]
The id of the protocol assigned to this handler

pfPHHandleData [IN]
Pointer to the function to be called by the router whenever a package arrives

Result
error code

18.1.15 RouterCalculateNodeAddr

*RTS_RESULT RouterCalculateNodeAddr (RTS_UI16 usBlkDrvType, RTS_UI8 byNetworkAddressBitSize, NETWORKADDRESS *pNetworkAddr, NODEADDRESS *pNodeAddr)*
Let the router calculate the CoDeSys peer address of a node. This works only for block drivers, which have a unique instance, e. g. the BlkDrvTcp. The given network address conains in this case the ip-address and the port of the node, in the block driver specific format. Additionally this function can be used to get the address of the first router instance of the own runtime system.

usBlkDrvType [IN]
Type of the block driver, see CmpCommunicationLibItf.h. If set to RTS_BLK_DRV_TYPE_NONE, the address of the first router instance is returned. In this case the next two parameters are ignored.

byNetworkAddressBitSize [IN]
Length of the specified network address in bits. Must match to the setting in the blockdriver. Should be 0, if RTS_BLK_DRV_TYPE_NONE is used to get the own router address.

pNetworkAddr [IN]
Networkaddress of the node, for which the peer address should be calcualted. Should be NULL, if RTS_BLK_DRV_TYPE_NONE is used to get the own router address.

pNodeAddr [OUT]
The function returns here the calculated node address.

Result
error code

18.1.16 RouterGetInstanceByName

*RTS_RESULT RouterGetInstanceByName (char *szName, RTS_HANDLE *phRouter)*
Get the handle to the routerinstance with the provided name.

szName [IN]
Name of the router. This parameter may be NULL to request a handle to the default router.

phRouter [OUT]
If a router exists with the specified name, then the handle of the router is written into this parameter. Otherwise it is set to RTS_INVALID_HANDLE.

Result

error code

18.1.17 RouterGetName

*RTS_RESULT RouterGetName (RTS_HANDLE hRouter, char *pszName, int nMaxLen)*

Get the name of a router specified by handle

hRouter [IN]
Handle to router

pszName [OUT]
Pointer to get router name

nMaxLen [IN]
Maximum buffer length of pszName

Result
error code

18.1.18 RouterGetMaxMessageSize

*RTS_RESULT RouterGetMaxMessageSize (RTS_HANDLE hRouter, PEERADDRESS addrPeer, RTS_UI16 *usMaxSize)*

Get the max. size of messages, which can be sent by higher layers to this peer address

hRouter [IN]
Handle to router. If set to RTS_INVALID_HANDLE, the first router instance is used.

addrPeer [IN]
Address to check

byMaxSize [OUT]
Max payload of the router message.

Result
error code

18.1.19 RouterGetMaxMessageSizeByAddressLength

*RTS_RESULT RouterGetMaxMessageSizeByAddressLength (RTS_HANDLE hRouter, RTS_UI16 usSumAddrLen, RTS_UI16 *usMaxSize)*

Get the max. size of messages, which can be sent by higher layers to this peer address

hRouter [IN]
Handle to router

usSumAddrLen [IN]
Sum of sender and receiver address length

byMaxSize [OUT]
Max payload of the router message.

Result
error code

18.1.20 RouterGetMaxAddressSize

*RTS_RESULT RouterGetMaxAddressSize (RTS_HANDLE hRouter, PEERADDRESS addrPeer, RTS_UI8 *byMaxSize)*

Get the sum of the address lengths of the current router and the addrPeer.

hRouter [IN]
Handle to router. If set to RTS_INVALID_HANDLE, the first router instance is used.

addrPeer [IN]
Address to check

byMaxSize [OUT]
Sum of router address and addrPeer length

Result
error code

18.1.21 RouterSend2

RTS_RESULT RouterSend2 (RTS_HANDLE hRouter, PEERADDRESS addrReceiver, int nProtocolId, RTS_UI8 byMessageId, ROUTERPRIORITY prio, PROTOCOL_DATA_UNIT pduData, RTS_BOOL bUseQueue)

Protocol handlers call this function to send a data package.

hRouter [IN]
Handle to the router
addrReceiver [IN]
Address of the receiver. Relative addresses are allowed as well as absolute ones.
nProtocolId [IN]
Identifies the protocol handler on the receiving host.
byMessageId [IN]
typically 0
prio [IN]
Priority of the message.
pduData [IN]
The data to be sent.
bUseQueue [IN]
Defines if the message should be queued, if it can not be send at once.
Result
error code

18.1.22 RouterSend

RTS_RESULT RouterSend (RTS_HANDLE hRouter, PEERADDRESS addrReceiver, int nProtocolId, RTS_UI8 byMessageId, ROUTERPRIORITY prio, PROTOCOL_DATA_UNIT pduData)
Protocol handlers call this function to send a data package.
hRouter [IN]
Handle to the router
addrReceiver [IN]
Address of the receiver. Relative addresses are allowed as well as absolute ones.
nProtocolId [IN]
Identifies the protocol handler on the receiving host.
byMessageId [IN]
typically 0
prio [IN]
Priority of the message.
pduData [IN]
The data to be sent.
Result
error code

18.1.23 RouterGetHostAddress

*RTS_RESULT RouterGetHostAddress (RTS_HANDLE hRouter, NODEADDRESS *pAddrRouter)*
Get the routers nodeaddress.
hRouter []
Handle to router. If set to RTS_INVALID_HANDLE, the first router instance is used.
pAddrRouter [OUT]
Is set to the nodeaddress of the router
Result
error code

18.1.24 RouterGetParentAddress

*RTS_RESULT RouterGetParentAddress (RTS_HANDLE hRouter, NODEADDRESS *pAddrParent)*
Get the nodeaddress of the parent node.
hRouter []
Handle to router. If set to RTS_INVALID_HANDLE, the first router instance is used.
pAddrParent [OUT]
Is set to the nodeaddress of the router
Result
error code

19 CmpScheduleTimer

The so called "Timer Scheduler" is in fact a mixture between the "Embedded Scheduler" and the full "Multitasking Scheduler" of CoDeSys Control. It supports the following task types: Freewheeling Tasks Cyclic Tasks Event Tasks External Events are currently not supported. Freewheeling- and Event Tasks are scheduled in the same way as with the "Embedded Scheduler" - they are called within the context of the Comm-Cycle in the Background loop. Contrary to this, the Cyclic Tasks are set up on hardware Timers, using the Interface of the component "SysTimer". The "SysTimer" component just needs to provide some kind of cyclic callback. It don't essentially needs to be a real hardware timer, but can also be emulated by software. The only prerequisit that the cyclic callback from SysTimer needs to fullfill is that the callbacks can only be preempted in a priorized way and that they don't block in between a cycle. This is necessary, because the locking scheme of the "Timer Scheduler" relys on this behavior. The simplest solution to implement those cyclic tasks, would be, to call the task code directly out of the ISR of the hardware timer, and that's it. But this solution reaches it's limits, when we want to do debugging in one task. To do this, we need to jump actively out of the task code and resume later at the same position. . prio . ^ . | . ISR | XXXXXXXXX . | | | . | | | . Comm Cycle |XXXXXX:::::::::XXXXXX . | . '----------------------> t This debugging scenario can be solved easily with the standard calls setjump and longjump. This is the first IEC cycle, where we set up the Task context: . _,._ . / '-. . / '-. . | ' . | [2] (1) . IEC Task | | XXXXXXXXXXXXX . | v | | . | [1]| | . ISR | XXX::::::::::::XXX . | | | . | | | . Comm Cycle |XXXXXXXXXXXXX:::::::::::::::::::XXXXXXXXXXXXXXXX . | . '---> t . . Index: . [1] setjump ContextFrame . [2] setjump ContextIecTask . (1) longjump ContextFrame On the first switch between the ISR context and the IEC task context, we have to switch the task stack. We do this with a call to the macro RTS_CPU_CALL_ON_STACK(), which needs to be implemented platform specifically. On every next call of the ISR, we don't switch the stack anymore, but simply make a longjump into the saved context number two (ContextIecTask). . _,._ . ,' '-. . / '. . / -. '. . / / \ \ . | | v '. . | | [2] (1) . IEC Task | . | XXXXXXXXXXXXX . | v | | | . | [1] (2) | | . ISR | XXXXXXXXX::::::::::::XXX . | | | . | | | . Comm Cycle |XXXXXXX:::::::::::::::::::::::::XXXXXXXXXXXXXXXXX . | . '---> t . . Index: . [1] setjump ContextFrame . [2] setjump ContextIecTask . (1) longjump ContextFrame . (2) longjump ContextIecTask The reason, why we have to jump in and out of the IEC task context every time is, that later in the debug case, our calling context may have changed when we leave the task. When we are running on a breakpoint, our IEC task saves the breakpoint context and jumps out of the task. . __ . ,' '-. . / -. '. . | / \ \ . | | v '. . | | [2][3](1) . IEC Task | | | XXXXXXXXXXXXX . | v | | | . | [1] (2) | | . ISR | XXXXXXXXX::::::::::::XXX . | | | . | | | . Comm Cycle |XXXXXXXX:::::::::::::::::::::::::XXXXXXXXXXXXXXXXX . | . '---> t . . Index: . [1] setjump ContextFrame . [2] setjump ContextIecTask . [3] setjump ContextIecTaskBP . (1) longjump ContextFrame . (2) longjump ContextIecTask As long as the task is waiting on the breakpoint, we are jumping into the stored breakpoint context. The breakpoint handling code there is checking by it's own, if it has to continue the task execution or if it has to jump back to the ContextFrame again. . __ . ,' '-. . / ,.. '. . | / '. \ . | / v '. . | | [2][3](1) . IEC Task | | | XXXXXXXXXXXXX . | v | | | . | [1] (3) | | . ISR | XXXXXXXXX::::::::::::XXX . | | | . | | | . Comm Cycle |XXXXXXXX:::::::::::::::::::::::::XXXXXXXXXXXXXXXXX . | . '---> t . . Index: . [1] setjump ContextFrame . [2] setjump ContextIecTask . [3] setjump ContextIecTaskBP . (1) longjump ContextFrame . (2) longjump ContextIecTask . (3) longjump ContextIecTaskBP The command CMD_TICK checks itself if the period in which it was called matches the configured period, which is set by the define SCHEDULEKEY_INT_SCHEDULER_INTERVAL_DEFAULT. If it missed one tick, it imediately generates an exception. This implies, that SysTimer and SysTime are synchronized or are based on the same timer source. This is a design pattern for the synchronization in pseudocode: static variable s_Time TIMER_ISR() IF s_LastTime = 0 s_LastTime = current time WHILE (current time - s_LastTime) > Period CALL Schedule(CMD_TICK) s_LastTime += Period This design pattern will base the call on the frequency of the hardware timer, but synchronizes it with another timer. This might lead to a jitter when the algorithm regulates a timer drift, but it takes care that no tick is missed. What is important when doing a watchdog check. If the both components SysTime and SysTimer are based on the same hardware frequency, it is not necessary to synchronize.

19.1 Define: CMPSCHEDULE_IECTASKS_HIGH_PRIO

Condition: #ifndef CMPSCHEDULE_IECTASKS_HIGH_PRIO
Category: Static defines
Type:
Define: CMPSCHEDULE_IECTASKS_HIGH_PRIO
Key:

This is a soft barrier and is not mandatory for the scheduler. But the Timer Scheduler is using this to place the task stacks of tasks with a higher priority (<= CMPSCHEDULE_IECTASKS_HIGH_PRIO) into a different memory area than the others (e.g. internal RAM).

The physical location of the stacks can be influenced with the two macros:

- CMPSCHEDULE_IECTASK_STACK_STD_ATTRIBUTE
- CMPSCHEDULE_IECTASK_STACK_HIGH_ATTRIBUTE

19.2 Define: CMPSCHEDULE_IECTASKS_STACK_STD_ATTRIBUTE

Condition: #ifndef CMPSCHEDULE_IECTASKS_STACK_STD_ATTRIBUTE
Category: Static defines
Type:
Define: CMPSCHEDULE_IECTASKS_STACK_STD_ATTRIBUTE
Key:
Attribute to place the stack of the IEC tasks in a different location.

19.3 Define: CMPSCHEDULE_IECTASKS_STACK_HIGH_ATTRIBUTE

Condition: #ifndef CMPSCHEDULE_IECTASKS_STACK_HIGH_ATTRIBUTE
Category: Static defines
Type:
Define: CMPSCHEDULE_IECTASKS_STACK_HIGH_ATTRIBUTE
Key:
Attribute to place the stack of high priority tasks in a different location (e.g. internal RAM).

Compiler Switch

- #define CMPSCHEDULE_DISABLE_TASK_WAKEUP_ON_CYCLE_END
  Switch to disable calling the scheduler at each IEC task cycle end once more to reduce processor load

19.4 CmpScheduleItf

Interface of the scheduler.

19.4.1 Define: CMPSCHEDULE_IECTASK_STACK_SIZE

Condition: #ifndef CMPSCHEDULE_IECTASK_STACK_SIZE
Category: Stack size
Type:
Define: CMPSCHEDULE_IECTASK_STACK_SIZE
Key: 0
Specifies the stack size of an IEC task of the timer scheduler. 0 is the default size of the operating system or environment.

19.4.2 Define: SCHEDULEKEY_INT_SCHEDULER_INTERVAL_US

Category:
Type:
Define: SCHEDULEKEY_INT_SCHEDULER_INTERVAL_US
Key: SchedulerInterval
Set the interval in microseconds of the scheduler. INT type.

19.4.3 Define: SCHED_DEBUG_LOOP_CYCLE_TIME

Condition: #ifndef SCHED_DEBUG_LOOP_CYCLE_TIME
Category: Static defines
Type:
Define: SCHED_DEBUG_LOOP_CYCLE_TIME
Key: 100
Sleep time in breakpoint loop in milliseconds

19.4.4 Define: SCHEDULE_FEATURE_RESET_ON_BREAKPOINT

Category: Features
Type: Int
Define: SCHEDULE_FEATURE_RESET_ON_BREAKPOINT
Key: 0x00000001
Supported features of the scheduler

19.4.5 Define: EVT_TaskCreateDone

Category: Events
Type:
Define: EVT_TaskCreateDone
Key: MAKE_EVENTID
Event is sent, after an IEC-task was created at application download

19.4.6 Define: EVT_PrepareTaskDelete

Category: Events
Type:
Define: EVT_PrepareTaskDelete
Key: MAKE_EVENTID
Event is sent, before an IEC-task will be deleted at application download

19.4.7 Define: EVT_ExternalEventTaskCreateDone

Category: Events
Type:
Define: EVT_ExternalEventTaskCreateDone
Key: MAKE_EVENTID
Event is sent, after an IEC-task, that is external event triggered was created

19.4.8 Define: EVT_PrepareExternalEventTaskDelete

Category: Events
Type:
Define: EVT_PrepareExternalEventTaskDelete
Key: MAKE_EVENTID
Event is sent, before an IEC-task, that is external event triggered will be deleted

19.4.9 Define: EVT_ScheduleTick

Category: Events
Type:
Define: EVT_ScheduleTick
Key: MAKE_EVENTID
Event is sent always in the schedule tick

19.4.10 Define: EVT_ScheduleTaskGap

Category: Events
Type:
Define: EVT_ScheduleTaskGap
Key: MAKE_EVENTID
Event is sent always, if no IEC task is active (task gap)

19.4.11 Typedef: EVTPARAM_CmpSchedule

Stuctname: EVTPARAM_CmpSchedule
Category: Event parameter
Typedef: typedef struct { Task_Desc* pTaskDesc; RTS_HANDLE hEvent; } EVTPARAM_CmpSchedule;

19.4.12 Typedef: EVTPARAM_CmpScheduleTick

Stuctname: EVTPARAM_CmpScheduleTick
Category: Event parameter
Typedef: typedef struct { RTS_SYSTIME *pSchedTime; int nScheduleIntervalUs; } EVTPARAM_CmpScheduleTick;

19.4.13 Typedef: schedwaitsleep_struct

Stuctname: schedwaitsleep_struct
Function to sleep a specified time interval in microseconds _without_ consuming processor load!
Typedef: typedef struct tagschedwaitsleep_struct { RTS_IEC_ULINT *ptSleepUs; VAR_IN_OUT RTS_IEC_UDINT SchedWaitSleep; VAR_OUTPUT } schedwaitsleep_struct;

19.4.14 Typedef: schedsettaskinterval_struct

Stuctname: schedsettaskinterval_struct
Set the actual interval of a cyclic task. If the specified task is no cyclic task, the function return an error. This interface can be used to synchronize a task to another task or to events.
Typedef: typedef struct tagschedsettaskinterval_struct { RTS_IEC_BYTE *hSchedTask; VAR_INPUT RTS_IEC_UDINT ulInterval; VAR_INPUT RTS_IEC_UDINT SchedSetTaskInterval; VAR_OUTPUT } schedsettaskinterval_struct;

19.4.15 Typedef: schedgetcurrenttask_struct

Stuctname: schedgetcurrenttask_struct

Is called to get the schedule handle of the current running task

Typedef: typedef struct tagschedgetcurrenttask_struct { RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SchedGetCurrentTask; VAR_OUTPUT } schedgetcurrenttask_struct;

19.4.16 Typedef: schedwaitbusy_struct

Stuctname: schedwaitbusy_struct

Function to wait busy during a specified time interval. This consumes maximum of processor load!

Typedef: typedef struct tagschedwaitbusy_struct { RTS_IEC_ULINT *ptSleepUs; VAR_IN_OUT RTS_IEC_UDINT SchedWaitBusy; VAR_OUTPUT } schedwaitbusy_struct;

19.4.17 Typedef: schedgettaskinterval_struct

Stuctname: schedgettaskinterval_struct

Get the actual interval of a cyclic task. If the specified task is no cyclic task, the function return an error.

Typedef: typedef struct tagschedgettaskinterval_struct { RTS_IEC_BYTE *hSchedTask; VAR_INPUT RTS_IEC_UDINT *pulInterval; VAR_IN_OUT RTS_IEC_UDINT SchedGetTaskInterval; VAR_OUTPUT } schedgettaskinterval_struct;

19.4.18 Typedef: schedgettaskeventbyhandle_struct

Stuctname: schedgettaskeventbyhandle_struct

Function returns the handle to the task event. With this event a task can be activaed externally, e.g. for external triggered event tasks. The event can be sent by SysEventSet(EventHandle);

Typedef: typedef struct tagschedgettaskeventbyhandle_struct { RTS_IEC_BYTE *hSchedTask; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SchedGetTaskEventByHandle; VAR_OUTPUT } schedgettaskeventbyhandle_struct;

19.4.19 Typedef: schedgettaskhandlebyname_struct

Stuctname: schedgettaskhandlebyname_struct

Function returns the handle to the task specified by name.

Typedef: typedef struct tagschedgettaskhandlebyname_struct { RTS_IEC_STRING *pszTaskName; VAR_IN_OUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SchedGetTaskHandleByName; VAR_OUTPUT } schedgettaskhandlebyname_struct;

19.4.20 Typedef: schedgetnumoftasks_struct

Stuctname: schedgetnumoftasks_struct

Is called to get the number of all registerd IEC tasks in the scheduler.

Typedef: typedef struct tagschedgetnumoftasks_struct { APPLICATION *pApp; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SchedGetNumOfTasks; VAR_OUTPUT } schedgetnumoftasks_struct;

19.4.21 Typedef: schedgetprocessorload_struct

Stuctname: schedgetprocessorload_struct

Returns the processor load of all IEC tasks

Typedef: typedef struct tagschedgetprocessorload_struct { RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_UDINT SchedGetProcessorLoad; VAR_OUTPUT } schedgetprocessorload_struct;

19.4.22 Typedef: schedgettaskhandlebyindex_struct

Stuctname: schedgettaskhandlebyindex_struct

Function returns the task handle of a task specified by an index.

Typedef: typedef struct tagschedgettaskhandlebyindex_struct { APPLICATION *pApp; VAR_INPUT RTS_IEC_DINT nIndex; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SchedGetTaskHandleByIndex; VAR_OUTPUT } schedgettaskhandlebyindex_struct;

19.4.23 SchedHasFeature

RTS_RESULT SchedHasFeature (unsigned long ulFeatures)

Routine to check, if a scheduler has the specified feature.

ulFeatures [IN]

Feature flags, See corresponding category "Features".

Result

error code

19.4.24 SchedAddTask

*RTS_HANDLE SchedAddTask (Task_Desc *pTask, RTS_RESULT *pResult)*

Create a new IEC task, that is under control of the CoDeSys Control Scheduler

Depending on the type of Scheduler which is used, the tasks may be created in different ways. Here is a small list of the most common Scheduling schemes:

- **CmpSchedule:** A full multitasking scheduler, where every task (independent of its type) has a corresponding OS task.
- **CmpScheduleEmbedded:** A very simple scheduler, where every task runs in the context of a super loop in the background (the comm cycle). The tasks are scheduled in polling mode by the scheduler.
- **CmpScheduleTimer:** Freewheeling and Event tasks are scheduled similar to CmpScheduleEmbedded, but cyclic tasks are placed on preemptive timers.

The tasks are set up, but not started, yet. To start the tasks, one should call SchedStart().

pTask [IN]
Task description

pResult [OUT]
Result

Result
Handle to task

19.4.25 SchedRemoveTask

RTS_RESULT SchedRemoveTask (RTS_HANDLE hSchedTask)

Removes a task from scheduler

hSchedTask [IN]
Handle to taks

Result
ERR_OK

19.4.26 SchedRemoveTask2

RTS_RESULT SchedRemoveTask2 (RTS_HANDLE hSchedTask, RTS_UI32 ulTimeoutMs)

Removes a task from scheduler with timeout

hSchedTask [IN]
Handle to taks

ulTimeoutMs [IN]
Timeout in milliseconds to wait for removing the task Some timeouts are predefined (see CmpStd.h):
- RTS_TIMEOUT_DEFAULT: Use default wait time
- RTS_TIMEOUT_NO_WAIT: No wait

Result
error code

19.4.27 Schedule

RTS_RESULT Schedule (RTS_HANDLE hSchedTask, int iCmd)

Execute a scheduler command.

Depending on the kind of scheduler, it may support different scheduler specific commands. Those commands may be described in the documentation of the scheduler component itself.

The following are generic commands which every scheduler should support:

- CMD_TICK: This command can be used by the scheduler to schedule it's cyclic- or event tasks. But most essentially this tick has to check the tasks watchdogs.
- CMD_DEBUG_LOOP: This command is used by the IEC tasks to halt on a breakpoint. It should only be called by an IEC task, which ran on a breakpoint. The implementation is very dependent on the kind of scheduler, but it will halt the execution of the IEC task until the breakpoint is left.

hSchedTask [IN]
Handle to taks

iCmd [IN]

Type of schedule command

Result
error code

19.4.28 SchedDebugEnter

RTS_RESULT SchedDebugEnter (RTS_HANDLE hSchedTask)
Enter task for debugging

hSchedTask [IN]
Handle to taks

Result
ERR_OK

19.4.29 SchedDebugLeave

RTS_RESULT SchedDebugLeave (RTS_HANDLE hSchedTask)
Leave from debugging

hSchedTask [IN]
Handle to taks

Result
ERR_OK

19.4.30 SchedPrepareReset

*RTS_RESULT SchedPrepareReset (APPLICATION *pApp, int bResetOrigin)*
Prepare reset, delete all IEC tasks in error state

Result
ERR_OK

19.4.31 SchedResetDone

*RTS_RESULT SchedResetDone (APPLICATION *pApp, int bResetOrigin)*
Restart all IEC tasks in error state

Result
ERR_OK

19.4.32 SchedStart

*RTS_RESULT SchedStart (APPLICATION *pApp)*

Start scheduling of all tasks of an application

pApp [IN]
APPLICATION object

Result
error code

19.4.33 SchedStop

*RTS_RESULT SchedStop (APPLICATION *pApp, RTS_HANDLE hTaskToExclude)*
Stop scheduling all tasks specified by application

pApp [IN]
APPLICATION object

hTaskToExclude [IN]
Handle of task to exclude from scheduling. hTask=RTS_INVALID_HANDLE, all tasks are disabled

Result
error code

19.4.34 SchedGetCurrentTask

*RTS_HANDLE SchedGetCurrentTask (Task_Desc **ppTask, RTS_RESULT *pResult)*
Is called to get the schedule handle of the current running task

ppTask [OUT]
Task description for the IecTask component. Can be NULL.

pResult [OUT]

ERR_OK or Error code

Result

Handle to the current running task or RTS_INVALID_HANDLE if failed

19.4.35 SchedGetNumOfTasks

*int SchedGetNumOfTasks (APPLICATION *pApp, RTS_RESULT *pResult)*

Is called to get the number of all registerd IEC tasks in the scheduler.

pApp [IN]

If an application is specified, only the tasks of this application is returned. If NULL, number of all tasks is returned.

pResult [OUT]

ERR_OK or Error code

Result

Number of tasks

19.4.36 SchedGetTaskDescByIndex

*Task_Desc * SchedGetTaskDescByIndex (APPLICATION *pApp, int iIndex, RTS_RESULT *pResult)*

Function returns the task description of a task specified by an index.

pApp [IN]

If an application is specified, only the task of this application is returned. If NULL, the task with the index in all tasks i returned.

pResult [OUT]

ERR_OK or Error code

Result

Task description

19.4.37 SchedGetTaskDescByHandle

*Task_Desc * SchedGetTaskDescByHandle (RTS_HANDLE hSchedTask, RTS_RESULT *pResult)*

Function returns the task description of a task specified by handle.

hSchedTask [IN]

Handle of the task

pResult [OUT]

Pointer to error code

Result

Task description

19.4.38 SchedGetTaskHandleByIndex

*RTS_HANDLE SchedGetTaskHandleByIndex (APPLICATION *pApp, int iIndex, RTS_RESULT *pResult)*

Function returns the task handle of a task specified by an index.

pApp [IN]

If an application is specified, only the task of this application is returned. If NULL, the task with the index in all tasks i returned.

pResult [OUT]

ERR_OK or Error code

Result

Handle to the task

19.4.39 SchedGetTaskHandleByName

*RTS_HANDLE SchedGetTaskHandleByName (char *pszTaskName, RTS_RESULT *pResult)*

Function returns the handle to the task specified by name.

pszTaskName [IN]

Task name

pResult [OUT]

ERR_OK or Error code

Result

Scheduler task handle

19.4.40 SchedGetTaskEventByHandle

*RTS_HANDLE SchedGetTaskEventByHandle (RTS_HANDLE hSchedTask, RTS_RESULT *pResult)*

Function returns the handle to the task event. With this event a task can be activaed externally, e.g. for external triggered event tasks. The event can be sent by SysEventSet(EventHandle);

hSchedTask [IN]
Scheduler task handle

pResult [OUT]
ERR_OK or Error code

Result
Event handle

19.4.41 SchedTimeslicePlcBegin

RTS_RESULT SchedTimeslicePlcBegin (void)

Begin of the plc timeslice. Can be called by an external component, if external timselicing is enabled.

Result
error code

19.4.42 SchedTimeslicePlcEnd

RTS_RESULT SchedTimeslicePlcEnd (void)

End of the plc timeslice. Can be called by an external component, if external timselicing is enabled.

Result
error code

19.4.43 SchedGetProcessorLoad

*unsigned long SchedGetProcessorLoad (RTS_RESULT *pResult)*

Returns the processor load, that is consumed by all IEC tasks in %. Can be in the range of 0..100 [%]. This feature must be enabled with the setting "ProcessorLoad.Maximum" > 0.

pResult [OUT]
Pointer to error code

Result
Processor load in percent

19.4.44 SchedGetTaskInterval

*RTS_RESULT SchedGetTaskInterval (RTS_HANDLE hSchedTask, RTS_UI32 *pulInterval)*

Get the actual interval of a cyclic task. If the specified task is no cyclic task, the function return an error.

hSchedTask [IN]
Handle to the task

ptInterval [OUT]
Pointer to the interval

Result
error code

19.4.45 SchedSetTaskInterval

RTS_RESULT SchedSetTaskInterval (RTS_HANDLE hSchedTask, RTS_UI32 ulInterval)

Set the actual interval of a cyclic task. If the specified task is no cyclic task, the function return an error. This interface can be used to synchronize a task to another task or to events.

hSchedTask [IN]
Handle to the task

tInterval [IN]
New interval

Result
error code

19.4.46 SchedWaitSleep

*RTS_RESULT SchedWaitSleep (RTS_SYSTIME *ptSleepUs)*

Function to sleep a specified time interval in microseconds _without_ consuming processor load!

ptSleepUs [IN]
Time to sleep in microseconds
Result
error code

19.4.47 SchedWaitBusy

*RTS_RESULT SchedWaitBusy (RTS_SYSTIME *ptSleepUs)*
Function to wait busy during a specified time interval. This consumes maximum of processor load!
ptSleepUs [IN]
Time to sleep in microseconds
Result
error code

19.4.48 SchedWatchdogExceptionHandler

RTS_RESULT SchedWatchdogExceptionHandler (RTS_HANDLE hSchedTask)
Handler is called, if a watchdog exception occurred.
hSchedTask [IN]
Handle to the task
Result
error code

19.4.49 SchedGetScheduleIntervalUs

*int SchedGetScheduleIntervalUs (RTS_RESULT *pResult)*
Get the actual schedule interval in microseconds.
pResult [OUT]
Pointer to error code
Result
Schedule interval in microseconds

19.4.50 SchedSetScheduleIntervalUs

RTS_RESULT SchedSetScheduleIntervalUs (int iScheduleIntervalUsNew)
Get the actual schedule interval in microseconds. The actual schedule interval (system base tick) has to be adapted accordingly.
iScheduleIntervalUsNew [IN]
Schedule interval in microseconds to set
Result
error code

19.4.51 SchedSetTaskIntervalByTaskHandle

RTS_RESULT SchedSetTaskIntervalByTaskHandle (RTS_HANDLE hSysTask, RTS_UI32 ulInterval)
Set the actual interval of a cyclic task. If the specified task is no cyclic task, the function returns an error. This interface can be used to synchronize a task to another task or to events. NOTE: The provided handle is a SysTask handle, no schedule handle!
hSysTask [IN]
Handle to the task provided by SysTask component
ulInterval [IN]
New interval
Result
error code

20 CmpSettingsEmbedded

This component enables the access to all runtime settings. Each component can have separate settings. NODE: SysSockCreate can be used to check, if SysSocket component is available. Is used actually e.g. for the embedded settings component to evaluate for the router component.

20.1 CmpSettingsItf

Interface for the settings component. The settings component can have different backend components, to realise different sources for the settings (e.g. ini-File, hardcoded, XML, etc.).

20.1.1 Define: STD_SETTINGS_DATABASE

Condition: #ifndef STD_SETTINGS_DATABASE
Category: Static defines
Type:
Define: STD_SETTINGS_DATABASE
Key: CODESYSControl
This define can be used to call always the standard settings database.

20.1.2 Define: STD_SETTINGS_DATABASE_OLD

Condition: #ifndef STD_SETTINGS_DATABASE_OLD
Category: Static defines
Type:
Define: STD_SETTINGS_DATABASE_OLD
Key: CoDeSysSP
This define can be used to call always the standard settings database.

20.1.3 Define: FILE_END_OF_LINE_DELIMITER

Condition: #ifndef FILE_END_OF_LINE_DELIMITER
Category: Static defines
Type:
Define: FILE_END_OF_LINE_DELIMITER
Key: \r\n
Delimiter for the end of line, if a settings file is used as backend.

20.1.4 Define: CMPSETTINGS_NUM_OF_STATIC_FILES

Condition: #ifndef CMPSETTINGS_NUM_OF_STATIC_FILES
Category: Static defines
Type:
Define: CMPSETTINGS_NUM_OF_STATIC_FILES
Key: 1
Number of static setting files. Only used by CmpSettings component to handle configuration files in static data.

20.1.5 Define: CMPSETTINGS_MASTER_CONFIG

Condition: #ifndef CMPSETTINGS_MASTER_CONFIG
Category: Static defines
Type:
Define: CMPSETTINGS_MASTER_CONFIG
Key: Master.cfg
Name of the master configuration file, if standard config file could not be opened

20.1.6 Define: CMPSETTINGS_STD_CONFIG

Condition: #ifndef CMPSETTINGS_STD_CONFIG
Category: Static defines
Type:
Define: CMPSETTINGS_STD_CONFIG
Key: STD_SETTINGS_DATABASE
Name of the standard configuration file

20.1.7 Define: USERDB_OBJECT_SETTINGS

Category: Static defines
Type:
Define: USERDB_OBJECT_SETTINGS
Key: Device.Settings
Predefined objects in the runtime

20.1.8 SettgSetDatabaseName

*RTS_RESULT SettgSetDatabaseName (char *pszName)*

Set database name for all settings

pszName [IN]
Name of settings file. Can be *.cfg or *.ini

Result
ERR_OK

20.1.9 SettgGetDatabaseName

*RTS_RESULT SettgGetDatabaseName (char *pszName, RTS_SIZE nNameLen)*

Get database name for all settings

pszName [OUT]
Name of settings file. Can be *.cfg or *.ini

nNameLen [IN]
Lenght of the name buffer in bytes

Result
Error code

20.1.10 SettgSetOptions

RTS_RESULT SettgSetOptions (int bSplitDatabases)

Set optional settings

bSeparateDatabases [IN]
If bSeparateDatabases=1, split settings file into each file per component

Result
ERR_OK

20.1.11 SettgIsOptionSplitDatabases

RTS_RESULT SettgIsOptionSplitDatabases (void)

Is option set to split all settings databases into each file per component

Result
ERR_OK

20.1.12 SettgGetIntValue

*RTS_RESULT SettgGetIntValue (const char *pszComponent, const char *pszKey, RTS_I32 *piValue, int iDefault, int bCached)*

Get an interger value from settings

pszComponent [IN]
Name of component

pszKey [IN]
Name of key

piValue [INOUT]
Pointer to value fro result

iDefault [IN]
Default value to set, if key not found

bCached [IN]
Flag, if value should be read cached or always direkt from file

Result
ERR_OK

Result
ERR_FAILED: Key not found

20.1.13 SettgSetIntValue

*RTS_RESULT SettgSetIntValue (const char *pszComponent, const char *pszKey, RTS_I32 iValue, int iBase)*

Get an interger value from settings

pszComponent [IN]
Name of component

pszKey [IN]
Name of key
iValue [IN]
Value to write
iBase [IN]
2=Base 2, 10=Decimal values, 16=Hex values
Result
ERR_OK

20.1.14 SettgGetStringValue

*RTS_RESULT SettgGetStringValue (const char *pszComponent, const char *pszKey, char *pszValue, int *piLen, char *pszDefault, int bCached)*
Get a string value from settings
pszComponent [IN]
Name of component
pszKey [IN]
Name of key
pszValue [INOUT]
Pointer to value for result
piLen [INOUT]
Max length of string value as IN and length of copied values excluding the NUL ending as OUT!
pszDefault [IN]
Default value to set, if key not found
bCached [IN]
Flag, if value should be read cached or always direkt from file
Result
ERR_OK
Result
ERR_FAILED: Key not found

20.1.15 SettgSetStringValue

*RTS_RESULT SettgSetStringValue (const char *pszComponent, const char *pszKey, char *pszValue, RTS_SIZE iLen)*
Get a string value from settings
pszComponent [IN]
Name of component
pszKey [IN]
Name of key
pszValue [IN]
Pointer to write value
iLen [IN]
Length of string to write excluding the NUL ending
Result
ERR_OK

20.1.16 SettgGetWStringValue

*RTS_RESULT SettgGetWStringValue (const char *pszComponent, const char *pszKey, RTS_WCHAR *pwszValue, int *piLen, RTS_WCHAR *pwszDefault, int bCached)*
Get a unicode string value from settings
pszComponent [IN]
Name of component
pszKey [IN]
Name of key
pwszValue [INOUT]
Pointer to value for result
piLen [INOUT]
Max length of string in unicode characters (not bytes!) as IN and length of copied unicode characters excluding the NUL ending as OUT

pwszDefault [IN]
Default value to set, if key not found
bCached [IN]
Flag, if value should be read cached or always direkt from file
Result
ERR_OK
Result
ERR_FAILED: Key not found

20.1.17 SettgSetWStringValue

*RTS_RESULT SettgSetWStringValue (const char *pszComponent, const char *pszKey, RTS_WCHAR *pwszValue, RTS_SIZE iLen)*
Get a unicode string value from settings
pszComponent [IN]
Name of component
pszKey [IN]
Name of key
pszValue [IN]
Pointer to write value
iLen [IN]
Length of string in unicode characters (not bytes!) to write without terminating NUL
Result
ERR_OK

20.1.18 SettgRemoveKey

*RTS_RESULT SettgRemoveKey (const char *pszComponent, const char *pszKey)*
Remove the specified key
pszComponent [IN]
Name of component
pszKey [IN]
Name of key
Result
ERR_OK

21 CmpSIL2

This component is seperated into two main parts: Generic Part (CmpSil2.c) Customer specific part (CmpSIL2OEM.c) The latter should be provided in form of a template, and be implemented by the OEM customer.

21.1 CmpSIL2Itf

This is the interface of the SIL2 component. The CmpSIL2 is used to implement several savety related feates, which are (in this form) only necessary in safety PLCs, following the Safety SIL2 concept of 3S.

These features include the following:

- Safe loading of a bootproject
- Selftest Hooks
- Switch between safe- and unsafe contexts
- Safe logger interface
- Check functions, to determine the current context and operation mode
- Control Flow

1) Bootproject

In CoDeSys Control SIL2, the bootproject is always loaded from flash, and from there, only in the "compact download format", as it is used by CmpAppEmbedded. In a standard CoDeSys Control Runtime, the bootproject is implicitely loaded, by CmpApp/CmpAppEmbedded within an init hook, that is executed during the call of CMInit() at startup.

In CoDeSys Control SIL2, this implicite loading is deactivated. Instead, the OEM customer has to call SIL2AppBoot() after CMInit().

The reason for this constraint is, that the function SIL2AppBoot() can make sure that the application is loaded with the constraints that are defined for a CoDeSys Control SIL2 system. Additionally, it is more safe to load the application after system startup is completed, because no unsafe init code is executed anymore after the application is loaded.

2) Selftest Hooks

Virtually every safety system needs some kind of self tests, which are either executed in every Safety Process Cycle, or within a specific time frame in the background.

While the selftests in the background need to be seperated into small chunks, which are executed in the COMM-Cycle (= Super Loop), the selftests at startup need to be executed at once, to prohibit the execution of the application when there is an error in the hardware or firmware.

3) Switches between safe- and unsafe contexts

The current execution context is managed by the OEM customer. But, to have a clear interface to switch between the contexts in a higher level, and to be able to execute unsafe code when coming from a safe context.

4) Safe logger interface

The standard monitoring of CoDeSys as well as the standard logger are not safe. Therefore, they may not be usable for the Endcustomer in some circumstances.

To have a safe interface to transport messages and data from the CoDeSys Control Runtiem to the programming and debugging system, the component CmpSIL2 provides it's own logger interface, which secures the data from unintended modifications with a checksum.

4) Check functions

The CmpSIL2 interface specifies a few functions, which need to be implemented by the OEM customer to determine the current context or operation mode. This is mainly necessary to restrict the execution of several functions, but there might also be other use-cases in which those functions migth be helpful for the OEM himself.

5) Control Flow

The component CmpSIL2, provides a control flow logging mechanism, that is used for diagnostic purposes, to check the control flow of the IEC tasks, before they are passing their outputs to the I/O drivers.

This interface is generically designed, and may therefore also be used by the OEM customer, to log and check his own control flow of his subsystems.

Example usage: SIL2_CONTROLFLOW s_IoMgrControlFlow[3]; SIL2ControlFlowLog(s_IoMgrControlFlow, 0); SIL2C

21.1.1 Define: CMPSIL2_LOGADD_MAX_STRLEN

Condition: #ifndef CMPSIL2_LOGADD_MAX_STRLEN
Category: Buffer sizes
Type:
Define: CMPSIL2_LOGADD_MAX_STRLEN
Key: 36
Specifies the maximum logable string size.

21.1.2 Define: CMPSIL2_LOGADD_MAX_ENTRIES

Condition: #ifndef CMPSIL2_LOGADD_MAX_ENTRIES
Category: Buffer sizes
Type:
Define: CMPSIL2_LOGADD_MAX_ENTRIES
Key: 50
Specifies the maximum number of strings.

21.1.3 Define: RTS_SIL2_FAIL_GENERAL

Category: Exceptions
Type:
Define: RTS_SIL2_FAIL_GENERAL
Key: 0x01
Exceptions: Runtime has encountered an error in safety mode and passes the info to an OEM Interface function

21.1.4 Typedef: SIL2_CONTROLFLOW

Stuctname: SIL2_CONTROLFLOW
Category: Typedef

Buffertype for Control Flow mechanism

Typedef: typedef struct SIL2_CONTROLFLOW { RTS_UI32 uiPattern; RTS_UI32 uiCRC; } SIL2_CONTROLFLOW;

21.1.5 Typedef: sil2checkcallercontext_struct

Stuctname: sil2checkcallercontext_struct

sil2checkcallercontext

Typedef: typedef struct tagsil2checkcallercontext_struct { RTS_IEC_UDINT udiCallerContextExpected; VAR_INPUT RTS_IEC_RESULT SIL2CheckCallerContext; VAR_OUTPUT } sil2checkcallercontext_struct;

21.1.6 Typedef: sil2oemexception_struct

Stuctname: sil2oemexception_struct

sil2oemexception

Typedef: typedef struct tagsil2oemexception_struct { RTS_IEC_UDINT udiException; VAR_INPUT RTS_IEC_RESULT SIL2OEMException; VAR_OUTPUT } sil2oemexception_struct;

21.1.7 Typedef: sil2addlog_struct

Stuctname: sil2addlog_struct

sil2addlog

Typedef: typedef struct tagsil2addlog_struct { RTS_IEC_STRING sMessage[81]; VAR_INPUT RTS_IEC_UDINT udiLogId; VAR_INPUT RTS_IEC_RESULT SIL2AddLog; VAR_OUTPUT } sil2addlog_struct;

21.1.8 Typedef: sil2oemgetoperationmode_struct

Stuctname: sil2oemgetoperationmode_struct

sil2oemgetoperationmode

Typedef: typedef struct tagsil2oemgetoperationmode_struct { RTS_IEC_UDINT SIL2OEMGetOperationMode; VAR_OUTPUT } sil2oemgetoperationmode_struct;

21.1.9 Typedef: sil2oemgetcallercontext_struct

Stuctname: sil2oemgetcallercontext_struct

sil2oemgetcallercontext

Typedef: typedef struct tagsil2oemgetcallercontext_struct { RTS_IEC_UDINT SIL2OEMGetCallerContext; VAR_OUTPUT } sil2oemgetcallercontext_struct;

21.1.10 Typedef: sil2oemgetmemorystate_struct

Stuctname: sil2oemgetmemorystate_struct

sil2oemgetmemorystate

Typedef: typedef struct tagsil2oemgetmemorystate_struct { RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_UDINT udiLength; VAR_INPUT RTS_IEC_UDINT SIL2OEMGetMemoryState; VAR_OUTPUT } sil2oemgetmemorystate_struct;

21.1.11 Typedef: sil2oemstackisvalid_struct

Stuctname: sil2oemstackisvalid_struct

sil2oemstackisvalid

Typedef: typedef struct tagsil2oemstackisvalid_struct { RTS_IEC_RESULT SIL2OEMStackIsValid; VAR_OUTPUT } sil2oemstackisvalid_struct;

21.1.12 SIL2AppBoot

RTS_RESULT SIL2AppBoot (void)

Function to load and start a Bootproject from Flash

Application Note: The caller must ensure that this function is called only within a safe context!

The SIL2 Component only loads bootprojects, which are in the "compact download format" format, and those only from flash. This format includes an checksum to check the consistency of that bootproject. Before loading the bootproject the CmpSIL2 checks this CRC to make sure that the bootproject was not corrupted after CoDeSys Programming System created it. As the data-initialization is done from bootproject-code, the CRC also covers the data in the bootproject. The main steps of loading the bootproject are:

- Create an application with the given name
- Get the Startaddress of the Bootproject in Flash
- Allocate Application Areas (Code - Flash, Data - SysMemAllocArea())
- Check the Bootproject (Bootproject Type, Target ID, Type, Version) and create GUIDs
- Call the Data-Initialization-Code from the bootproject (Relocation, Code Init, Download POU, Global Init, Global Exit)
- Set the Application State to loaded
- If everything is correct, start the Application

Function has no function parameter, but behavior is also dependant on state of bootproject

Result

Result of loading bootproject

21.1.13 SIL2CheckCallerContext

RTS_RESULT SIL2CheckCallerContext (RTS_UI32 ulCallerContextExpected)

Function to check the current Caller Context of the Runtime

Expected Context is checked vs the current context, and an exception is thrown if they are different.

ulCallerContextExpected [IN]

Expected Context, that is checked against current context.

Result

Returns if comparision was ok or not, or if problem occured.

21.1.14 SIL2OEMEnterDebugMode

RTS_RESULT SIL2OEMEnterDebugMode (void)

Function to set Runtime to DebugMode

This function is called from save context after receiving an "EnterDebugMode" online service to set runtime to debug mode

As this function is only called from within save context no further check is required

Result

Result of Setting Mode

21.1.15 SIL2OEMException

void SIL2OEMException (RTS_SIL2_EXCEPTION Exception)

Function to set Runtime into Exception-Mode

Whenever the runtime detects invalid behaviour, values or states, it calls this function with a specific Exception Code.

Depending on the implementation this function may not return!

Exception [IN]
Exception Code

21.1.16 SIL2OEMRuntimeCheckCyclic

RTS_RESULT SIL2OEMRuntimeCheckCyclic (RTS_UI32 ulCRCExpected)

Function to start/continue cyclic Runtime CRC Check

If the check fails, the function SIL2OEMException is called with exception code RTS_SIL2_EXCEPTION_CRC_CYCLIC

The returnvalue ERR_PENDING could be used to control if a cyclic check takes too long

Function does not return if CRC is wrong, as SIL2OEMException is called!

ulCRCExpected [IN]
Expected CRC, that is compared to the one that is calculated

Result
Result of Cyclic Check

21.1.17 SIL2OEMRuntimeCheckComplete

RTS_RESULT SIL2OEMRuntimeCheckComplete (RTS_UI32 ulCRCExpected)

Function to start complete Runtime CRC Check

If the check fails, the function SIL2OEMException is called with exception code RTS_SIL2_EXCEPTION_CRC_COMPLETE

Function does not return if CRC is wrong, as SIL2OEMException is called!

uiCRCExpected [IN]
Expected CRC, that is compared to the one that is calculated

Result
Result of Complete Check

21.1.18 SIL2AddLog

RTS_RESULT SIL2AddLog (char pszMessage,RTS_UI32 ulLogId)*

Function to add a secure Logentry

These secure logentries don't use the standard Logging mechanism! With this function it is possible to add secure messages to a Messagequeue within CmpSIL2. This Messagequeue can be fetched by an Onlineservice from the SIL2 Programmingsystemplugin. The messages are stored and transmitted with a CRC to provide a secure logging. The maximal length of the string is restricted (CMPSIL2_LOGADD_MAX_STRLEN).

pszMessage [IN]
String containing the message, must not be longer than CMPSIL2_LOGADD_MAX_STRLEN

ulLogId [IN]
Id for own Identification purposes

Result
Result of adding secure Logentry

21.1.19 SIL2OEMGetOperationMode

RTS_SIL2_OPMODE SIL2OEMGetOperationMode (void)

Function to get the Operationmode of the Runtime

Returns RTS_SIL2_OPMODE_DEBUG or RTS_SIL2_OPMODE_SAFE depending on Operationmode, returns RTS_SIL2_OPMODE_ERROR if error occured or if in unknown state

Result
Returns SIL2 Operation Mode: RTS_SIL2_OPMODE_DEBUG or RTS_SIL2_OPMODE_SAFE if operation was successful, RTS_SIL2_OPMODE_ERROR if error occured or if in unknown state!

21.1.20 SIL2OEMGetCallerContext

RTS_UI32 SIL2OEMGetCallerContext (void)

Function to get the current Caller Context of the Runtime

Returns RTS_SIL2_CALLERCTX_SAFE or RTS_SIL2_CALLERCTX_UNSAFE depending on Caller context, returns RTS_SIL2_CALLERCTX_ERROR if error occured or if in unknown state

Result
Returns SIL2 Caller Context: RTS_SIL2_CALLERCTX_SAFE or RTS_SIL2_CALLERCTX_UNSAFE if operation was successful, RTS_SIL2_CALLERCTX_ERROR if error occured or if in unknown state!

21.1.21 SIL2OEMGetMemoryState

*RTS_SIL2_ADDRESSSTATE SIL2OEMGetMemoryState (RTS_UI8 *pAddress, RTS_UI32 ulLength)*

Function to get the MemoryState (safe/unsafe) for a specific Memoryrange

The Addressrange where pAddress points to with the length of ulLength is checked and the corresponding RTS_SIL2_ADDRESSSTATE is returned: RTS_SIL2_ADDRESS_SAFE or RTS_SIL2_ADDRESS_UNSAFE

pAddress [IN]
Pointer to Addressrange to check for Addressstate
ulLength [IN]
Length of Addressrange to check for Addressstate
Result
Result for Memoryaddress check

21.1.22 SIL2OEMStackIsValid

RTS_RESULT SIL2OEMStackIsValid (void)

Function to check if Stack is Valid

This function is called before entering the Safemode, it returns ERR_OK if the stack is valid, and ERR_FAILED if an error occured or was detected!

Result
Result of Stackcheck

21.1.23 SIL2OEMExecuteNonSafetyJob

*RTS_RESULT SIL2OEMExecuteNonSafetyJob (void (*pfNonSafetyJob)(void * pParam),void * pParam)*

Function to delegate a Non-Safety Job

This function can be used to delegate a non-safety job from within the safe-context to be executed from the Unsafe context

The function pfNonSafetyJob has to be called from Unsafe Context with pParam as argument

pfNonSafetyJob [IN]
Function Pointer to NonSafety Job
pParam [IN]
Pointer to Parameter for NonSafety Job
Result
Result of delegating a Non-Safety Job

21.1.24 SIL2ControlFlowLog

RTS_RESULT SIL2ControlFlowLog (SIL2_CONTROLFLOW pControlFlow, RTS_UI8 uiCurrID)*

Function to log the program flow

The caller is responsible for providing buffer for storing the control flow data. After having logged, the function SIL2ControlFlowCheck can be used to check if all control positions have been logged in the correct order.

The Log IDs may not be out of range of the specified log buffer.

pControlFlow [IN]
Buffer to store control flow data in

uiCurrID [IN]
Current Log Nr

Result
Result of Log to control-flow

21.1.25 SIL2ControlFlowCheck

RTS_RESULT SIL2ControlFlowCheck (SIL2_CONTROLFLOW pControlFlow, RTS_UI8 uiTotalNr)*

Function to check the previously logged program flow

The caller is responsible for providing buffer for storing the control flow data. After having logged, the function SIL2ControlFlowCheck can be used to check if all control positions have been logged in the correct order.

The number of IDs may not be out of range of the specified log buffer.

pControlFlow [IN]
Buffer to store control flow data in

uiTotalNr [IN]
Nr of Logs expected or to check

Result
Result of control-flow check

22 CmpSrv

Level 7 communication component of the runtime system. Realizes the server part in the communication system. Other components can register at this component, if they intend to use communication services. The communication services are organized in service groups, so one component can only register one handler to one single service group. This handler will be called, if a service of the assigned service group is sent.

22.1 CmpSrvItf

Interface of the level 7 server.

22.1.1 Define: HEADERTAG_3S

Category:
Type:
Define: HEADERTAG_3S
Key: 0xCD55
Defines the default CoDeSys layer 7 protocol.

22.1.2 Define: HEADERTAG_SAFETY

Category:
Type:
Define: HEADERTAG_SAFETY
Key: 0x5AF4
Defines the default safety communication layer 7 protocol.

22.1.3 Define: SG_REPLY_PREFIX

Category: Service Groups
Type:
Define: SG_REPLY_PREFIX
Key: 0x0080
Service groups for the layer 7 communication

22.1.4 Define: SRV_NUM_OF_STATIC_GROUPS

Condition: #ifndef SRV_NUM_OF_STATIC_GROUPS
Category: Static defines
Type:
Define: SRV_NUM_OF_STATIC_GROUPS
Key: SG_MAX_DEFINED
Number of static groups

22.1.5 Define: SRV_NUM_OF_SYNC_SERVICES

Condition: #ifndef SRV_NUM_OF_SYNC_SERVICES
Category: Static defines
Type:
Define: SRV_NUM_OF_SYNC_SERVICES
Key: 10
Max number of services that can be defined for synchronous execution

22.1.6 Define: TAG_ERROR

Category: Service reply tags
Type:
Define: TAG_ERROR
Key: 0xFF7F
Global service reply tags

22.1.7 Define: EVT_ExecuteOnlineService

Category: Events
Type:
Define: EVT_ExecuteOnlineService
Key: MAKE_EVENTID
NOTE: Every service group can be opened, as it is still provided by the server component! The corresponding event will created implicitly by the server component, if EventOpen() is called.
Example: Client calls: hEvent = CAL_EventOpen([Service group], CMPID_CmpSrv, NULL); CmpSrv created the event: hEvent = CAL_EventCreate2([Service group], CMPID_CmpSrv, 1, NULL);
ATTENTION: This feature is only available right after CH_INIT3 step!

22.1.8 Typedef: EVTPARAM_CmpSrv

Stuctname: EVTPARAM_CmpSrv
Category: Event parameter
Typedef: typedef struct { SERVICEHANDLER_PARAMETER *pServceHandlerParameter; } EVTPARAM_CmpSrv;

22.1.9 ServerAppHandleRequest

RTS_RESULT ServerAppHandleRequest (RTS_UI32 ulChannelId, PROTOCOL_DATA_UNIT pduRequest, PROTOCOL_DATA_UNIT pduReply)

Handle one request from the communication layer below (router)

ulChannelId [IN]
Id of the channel on which the request arrived

pduRequest [IN]
Pointer to the request

pduReply [OUT]
Pointer to the request reply

Result
error code

22.1.10 ServerAppHandleRequest2

RTS_RESULT ServerAppHandleRequest2 (RTS_HANDLE hRouter, RTS_UI32 ulChannelId, PROTOCOL_DATA_UNIT pduRequest, PROTOCOL_DATA_UNIT pduReply)

Obsolete: Use ServerAppHandleRequest instead. Will be removed in future versions!

Handle one request from the communication layer below (router)

hRouter [IN]
Obsolete parameter, should be set to RTS_INVALID_HANDLE.

ulChannelId [IN]
Id of the channel on which the request arrived

pduRequest [IN]
Pointer to the request

pduReply [OUT]
Pointer to the request reply

Result
error code

22.1.11 ServerRegisterServiceHandler

RTS_RESULT ServerRegisterServiceHandler (RTS_UI32 ulServiceGroup, PFServiceHandler pfServiceHandler)

Register a handler for requests to a specific service group

ulServiceGroup [IN]
The service group which is handled by the provided function

pfServiceHandler [IN]
A handler function.

22.1.12 ServerRegisterServiceHandler2

*RTS_RESULT ServerRegisterServiceHandler2 (RTS_UI32 ulServiceGroup, PFServiceHandler pfServiceHandler, char *pszRouter)*

Obsolete: Use ServerRegisterServiceHandler instead. Will be removed in future versions!

Register a handler for requests to a specific service group

ulServiceGroup [IN]
The service group which is handled by the provided function

pfServiceHandler [IN]
A handler function.

pszRouter [IN]
Obsolete parameter, should be set to NULL.

Result
error code

22.1.13 ServerRegisterServiceHandler3

RTS_RESULT ServerRegisterServiceHandler3 (RTS_UI32 ulServiceGroup, PFServiceHandler2 pfServiceHandler2)

Obsolete: Use ServerRegisterServiceHandler instead. Will be removed in future versions!

Register a handler for requests to a specific service group

ulServiceGroup [IN]
The service group which is handled by the provided function
pfServiceHandler2 [IN]
A handler function.
Result
error code

22.1.14 ServerUnRegisterServiceHandler

RTS_RESULT ServerUnRegisterServiceHandler (RTS_UI32 ulServiceGroup, PFServiceHandler pfServiceHandler)
Unregister a handler for requests to a specific service group
ulServiceGroup [IN]
The service group which is handled by the provided function
pfServiceHandler [IN]
A handler function.

22.1.15 ServerRegisterProtocolHandler

RTS_RESULT ServerRegisterProtocolHandler (RTS_UI16 usProtocolId, PFServiceHandler pfServiceHandler)
Register a handler for requests of a specific protocol other then HEADERTAG_3S. All requests with that protocol will be sent to this handler.
usProtocolId [IN]
The protocol id which is handled by the provided function
pfServiceHandler [IN]
A handler function.

22.1.16 ServerRegisterProtocolHandler2

*RTS_RESULT ServerRegisterProtocolHandler2 (RTS_UI16 usProtocolId, PFServiceHandler pfServiceHandler, char *pszRouter)*

Obsolete: Use ServerRegisterProtocolHandler instead. Will be removed in future versions!

Register a handler for requests of a specific protocol other then HEADERTAG_3S. All requests with that protocol will be sent to this handler.

usProtocolId [IN]
The protocol id which is handled by the provided function
pfServiceHandler [IN]
A handler function.
pszRouter [IN]
Obsolete parameter, should be set to NULL.
Result
error code

22.1.17 ServerUnRegisterProtocolHandler

RTS_RESULT ServerUnRegisterProtocolHandler (RTS_UI16 usProtocolId, PFServiceHandler pfServiceHandler)
Unregister a handler for requests of a specific protocol.
usProtocolId [IN]
The protocol id which is handled by the provided function
pfServiceHandler [IN]

A handler function.

22.1.18 ServerFinishRequest

RTS_RESULT ServerFinishRequest (RTS_UI32 ulChannelId, PROTOCOL_DATA_UNIT pduData)

Send a reply to a request previously received by a service handler

ulChannelId [IN]
Id of the channel on which to answer a reply. Must be the same that was passed in to the service handler function.

pduData [IN]
Contains the buffer and the length of the reply data. Buffer must be the same as the one passed in to the service handler function, the length must not be greater then the maximum reply buffer length.

22.1.19 ServerGenerateSessionId

*RTS_RESULT ServerGenerateSessionId (RTS_UI32 *pulSessionId)*

Generates a unique session Id

pulSessionId [OUT]
Pointer to get back the generated session Id

Result
error code

22.1.20 ServerSetRedundancyMode

RTS_RESULT ServerSetRedundancyMode (RTS_BOOL bRedundant)

Set redundancy mode to enable synchronous execution of services.

bRedundant [IN]

Result
error code

22.1.21 ServerExecuteOnlineService

RTS_RESULT ServerExecuteOnlineService (void)

Set flag to execute online service

Result
error code

22.1.22 ServerHandleRequest

RTS_RESULT ServerHandleRequest (RTS_UI32 ulChannelId, PROTOCOL_DATA_UNIT pduRequest, PROTOCOL_DATA_UNIT pduReply)

Handle one request (called by CmpRedundancy)

ulChannelId [IN]
Id of the channel on which the request arrived

pduRequest [IN]
Pointer to the request

pduReply [OUT]
Pointer to the request reply

Result
error code

22.1.23 SrvGetUserNotificationService

*RTS_RESULT SrvGetUserNotificationService (BINTAGWRITER *pWriter, PROTOCOL_DATA_UNIT *pduSendBuffer)*

Get the last log entry of class LOG_USER_NOTIFY as a toplevel online service tag

pWriter [IN]
Pointer to the bintag writer to get the service tag

pduSendBuffer [IN]
Pointer to the send buffer to reset the content of the bintag writer

Result
Error code:
- ERR_OK: There is still an unread log entry of the type LOG_USER_NOTIFY

- ERR_NO_OBJECT: No pending log entry of the type LOG_USER_NOTIFY
- ERR_NOT_SUPPORTED: Service not supported

22.1.24 SrvGetUserNotificationService2

*RTS_RESULT SrvGetUserNotificationService2 (BINTAGWRITER *pWriter, PROTOCOL_DATA_UNIT *pduSendBuffer, unsigned long ulTagId)*

Get the last log entry of class LOG_USER_NOTIFY as a toplevel online service tag

pWriter [IN]

Pointer to the bintag writer to get the service tag

pduSendBuffer [IN]

Pointer to the send buffer to reset the content of the bintag writer

ulTagId [IN]

TagId to send user notify info

Result

Error code:

- ERR_OK: There is still an unread log entry of the type LOG_USER_NOTIFY
- ERR_NO_OBJECT: No pending log entry of the type LOG_USER_NOTIFY
- ERR_NOT_SUPPORTED: Service not supported

23 IoDrvUnsafeBridge

Interface of IoDrvBridge, which links unsafe IO drivers to safe context. The IoDrvBridge is used to handle unsafe IO drivers. This is done by copying the whole IO configuration of the downloaded application to unsafe memory and to call the unsafe IO drivers with the CmpSIL2 interface function SIL2OEMExecuteNonSafetyJob() The following drawing illustrates how an IO configuration with IoDrvBridge looks like in CoDeSys. Safe IO drivers are placed directly underneath the device and unsafe IO drivers are placed underneath the IoDrvBridge. Both the safe IO drivers under the device and the unsafe IO drivers under IoDrvBridge may be C or IEC IO drivers. . +--------+ . | Device | . +--------+ . | . | +----------------+ . +--| Safe IO driver | . | +----------------+ . | +----------------+ . +--| IoDrvBridge | . +----------------+ . | . | +--------------------+ . +--| Unsafe IO driver 1 | . | +--------------------+ . | +--------------------+ . +--| Unsafe IO driver n | . +--------------------+ To handle the copied IO configuration and the supported IO drivers, the following mem pools and memory is used: Copied IO configuration: The copied IO configuration is stored in a static buffer, which is handled by the following local functions: AllocatorBufferInit() AllocatorAdd() AllocatorSpaceLeft() These functions take care about the alignment, the current position and the available space left of the buffer. It is only possible to allocate and not to free memory blocks. The memory blocks to allocate may be of any size as long as it fits to the memory. The size of the static buffer is defined by COPIED_IO_CONFIG_SIZE. The alignment of an element depends on the size of the element as follows: (size >= 8) --> align 8 (size < 8 && >= 4) --> align 4 (size < 4 && >= 2) --> align 2 (size > 2) --> align 1 Example: . s_byCopiedIoConfig . +-------------------------+ . | ElementA_1 | . | +-------------+ . | | ALIGNMENT | . +-------------------------+ . | ElementA_n | . | +-------------+ . | | ALIGNMENT | . +-------------+-------------+ . | ElementB_1 | ElementB_2 | . +-------------+-------------+ . | ElementB_3 | ALIGNMENT | . +-------------+-------------+ . | ElementC_1 | . +-------------------------+ . | ElementC_n | . +-------------------------+ . | ... | . +-------------------------+ Copied IO configuration elements: To be able to link original and copied IO configuration elements, this local mem pool stores a pointer to the original element list, a pointer to the copied element list and the number of elements in the list. There is no need to lock this mem pool for every access, because elements are only added/deleted in IoDrvUpdateConfiguration() and IoDrvUpdateMapping, which are called from communication context. This mem pool is handled by the following local functions: MappingListAdd() - to add an element MappingListGetCopiedElement() - to get the corresponding copied element to an original element MappingListGetOriginalElement() - to get the corresponding original element to a copied element MappingListDeleteAllElements() - to remove all elements from the pool The following drawing illustrates an example IO configuration . Original IO config ElementMappingPool Copied IO config . . +--------------+ +-----------------+ +--------------+ . | Connector_1 |<-----| pOriginalElement | +->| Connector_1 | . +--------------+ +-----------------+ | +--------------+ . | Connector_2 | | pCopiedElement |---+ | Connector_2 | . +--------------+ +-----------------+ +--------------+ . | Connector_n | | NumOfElement = n | | Connector_n | . +--------------+ +-----------------+ +--------------+ . +--------------+ +-----------------+ +--------------+ . | Parameter_1 |<-----| pOriginalElement | +->| Parameter_1 | . +--------------+ +-----------------+ | +--------------+ . | Parameter_2 | | pCopiedElement |---+ | Parameter_2 | . +--------------+ +-----------------+ +--------------+ . | Parameter_m | | NumOfElement = m | | Parameter_m | . +--------------+ +-----------------+ +--------------+ Handled IO drivers: s_DriverMappingPool stores elements of DriverMapEntry, which contain the instance information of the IO driver. There is no need to lock this mem pool for every access, because elements are only added/deleted in IoDrvUpdateConfiguration() and IoDrvUpdateMapping, which are called from comm cycle. The IoDrvBridge is implementing the IoDrv interface and is using the IoMgr interface as all other IO drivers. The following points describe the handling of unsafe IO drivers in IoDrv interface function implementations of IoDrvBridge. IoDrvCreate() To be able to handle all unsafe IO drivers underneath this IO driver, the IoDrvBridge must be the first IO driver registered in the IoMgr. This is checked in the IoDrvCreate function of IoDrvBridge by calling the IoMgr interface function IoMgrGetFirstDriverInstance. If this function returns an instance of an IO driver, an exception is thrown. If no driver is registered yet, an instance of the IoDrvBridge is created. After that all other IO drivers may register in IoMgr as usual. IoDrvUpdateConfiguration() This is the first function, which is called by the IoMgr, after an application, containing an IO configuration was downloaded to the device. The complete connector list and the number of connectors are passed as parameters. As the IoDrvBridge is the first IO driver, which was registered in IoMgr, it is also the first IO driver, which is called by the IoMgr. This allows the IoDrvBridge to take care about the unsafe IO drivers before they are called by the IoMgr from safe context. First of all the whole connector list is copied to the copied IO configuration memory by using the AllocatorAdd() function and an element is added to the CopiedElementListMappingPool with the MappingListAdd() function. After that all parameter lists of all connectors are copied and mapping list elements are added similarly to the connector list. To make the copied connector list consistent, each parameter list pointer is updated to the copied parameter list. To get the IO driver instances the IoMgr functions IoMgrGetFirstDriverInstance() and IoMgrGetNextDriverInstance() are used. Those IO drivers, whose module ID is matching one of the module IDs SUPPORTED_UNSAFE_DRIVERS_LIST, are added to the DriverMappingPool. To avoid calls from IoMgr to IO drivers, which are handled by the IoDrvBridge, they are unregisters from IoMgr. The remaining registered IO drivers in IoMgr must be safe IO drivers. After that, IoDrvUpdateConfiguration() of every supported IO driver is called with the copied connector list. As every IO driver is registering to connectors in the copied connector list, the IoDrvBridge must register to all corresponding connectors in the original connector list and also copy the flags of the copied connectors to the original connectors. The

following example shows an original and a copied connector list. As IoDrvTest is registered for Connector_1 in the copied connector list, IoDrvBridge must register for the same connector in the original connector list and copy the flags. When IoDrvBridge is called with an specific, connector for example to read inputs, it is possible to get the corresponding IO driver, which is registered for this connector. . . Original connector list ElementMappingPool Copied connector list . . +------------------+ +------------------+ +------------------+ . | Connector_1 |<----| pOriginalElement | +->| Connector_1 | . | | +------------------+ | | | . |hIoDrv=IoDrvBridge| | pCopiedElement |---+ |hIoDrv=IoDrvTest_1| . | dwFlags = 0x1234 | +------------------+ | dwFlags = 0x1234 | . | | | NumOfElement = n | | | . | | +------------------+ | | . +------------------+ +------------------+ . | Connector_2 | | Connector_2 | . | | | | . |hIoDrv=IoDrvBridge| |hIoDrv=IoDrvTest_2| . | dwFlags = 0x4321 | | dwFlags = 0x4321 | . | | | | . | | | . +------------------+ +------------------+ . | Connector_n | | Connector_n | . | | | | . |hIoDrv=IoDrvBridge| |hIoDrv=IoDrvTest_x| . | dwFlags = 0xAA55 | | dwFlags = 0xAA55 | . | | | | . | | | . +------------------+ +------------------+ When a reset is performed, IoDrvUpdateConfiguration() is called without a connector list to allow the IO driver to clean up the environment. In case of the IoDrvBridge the copied IO configuration buffer and all mem pools are cleared. Before the supported IO driver mem pool is cleared, the C IO drivers are registered in IoMgr again and the IEC driver instances are deleted. IoDrvUpdateMapping() IoDrvUpdateMapping() copies the IoConfigTaskMap, IoConfigConnectorMap and IoConfigChannelMap of the original IO configuration as described in IoDrvUpdateConfiguration. The connector pointers of the copied IoDrvConnectorMap list are updated to the copied connectors and the parameter pointers of the copied IoDrvChannelMap list are updated to the copied parameters. Additionally the memory of every IO channel is duplicated. This means that memory is allocated in the CopiedIoConfig buffer and the pointer to IEC address is set to this memory in the copied IoDrvChannelMap. The same thing is done for every parameter of IoConfigParameter, as the parameters may change. Therefore the pointer dwValue is set to the new memory in every copied IoConfigParameter. At the end all supported IO drivers are called with the copied IoConfigTaskMap. IoDrvReadInputs() The IoDrvReadInputs() function of IoDrvBridge gets the corresponding copied connector map, gets the registered supported IO driver as described in the example in IoDrvUpdateConfiguration() and calls the IO driver with the copied connector map. After the IO driver wrote the inputs to the copied memory, the IoDrvBridge copies this data to the real IEC memory. IoDrvWriteOutputs() This function of the IoDrvBridge works similar to IoDrvReadInputs, but it copies the outputs to the copied IO memory before the IO driver is called. IoDrvStartBusCycle() This function gets the corresponding copied connector to the original connector, which is passed as parameter. Then the unsafe IO driver, which is registered to the copied connector is called with the copied connector list. IoDrvGetModuleDiagnosis() This function may be called from IoMgr and from IO driver context. Usually from IoDrvStartBusCycle. It sets the connector flags corresponding to the state of the IO driver. The IoDrvGetModuleDiagnosis() implementation is never called by the supported IO drivers, because they always call their own implementation of this function and it is not possible to get the changed flags in IoDrvBridge. Because of this, all flags of the copied connector list are copied in the safe COMM_CYCLE_HOOK of IoDrvBridge. The period is defined by IODRVBRIDGE_CONNECTOR_FLAGS_UPDATE_PERIOD. When this function is called from IoMgr(the passed parameter p_IBase handle is equal to s_pIBase of IoDrvBridge) the corresponding IO driver implementation of IoDrvGetModuleDiagnosis() is called with the corresponding copied connector.

23.1 IoDrvBridgeItf

IoDrvBridge interface.

23.1.1 Define: NUM_OF_MAPPED_COPIED_ELEMENTS

Condition: #ifndef NUM_OF_MAPPED_COPIED_ELEMENTS
Category: Static defines
Type:
Define: NUM_OF_MAPPED_COPIED_ELEMENTS
Key: 40
Maximum mapped elements

23.1.2 Define: NUM_OF_DRIVER_ELEMENTS

Condition: #ifndef NUM_OF_DRIVER_ELEMENTS
Category: Static defines
Type:
Define: NUM_OF_DRIVER_ELEMENTS
Key: 10
Maximum num of drivers

23.1.3 Define: COPIED_IO_CONFIG_SIZE

Condition: #ifndef COPIED_IO_CONFIG_SIZE
Category: Static defines
Type:

Define: COPIED_IO_CONFIG_SIZE
Key: 0x4000
Maximum size of copied IO configuration buffer

23.1.4 Define: COPIED_IO_CONFIG_ALIGNMENT

Condition: #ifndef COPIED_IO_CONFIG_ALIGNMENT
Category: Static defines
Type:
Define: COPIED_IO_CONFIG_ALIGNMENT
Key: 4
Alignment of copied IO configuration buffer

23.1.5 Define: IODRVBRIDGE_CONNECTOR_FLAGS_UPDATE_PERIOD

Condition: #ifndef IODRVBRIDGE_CONNECTOR_FLAGS_UPDATE_PERIOD
Category: Static defines
Type:
Define: IODRVBRIDGE_CONNECTOR_FLAGS_UPDATE_PERIOD
Key: 2000
Period of connector flags update in ms

23.1.6 Define: SUPPORTED_UNSAFE_DRIVERS_LIST

Category: Static defines
Type:
Define: SUPPORTED_UNSAFE_DRIVERS_LIST
Key: SUPPORTED_UNSAFE_DRIVERS_LIST
List of supported unsafe drivers CANOpenManager: wModuleType = 0x0F CANOpenMaster: wModuleType = 0x10 CANOpenSlave: wModuleType = 0x11 CANOpenDevice: wModuleType = 0x12

23.2 CmpIoDrvItf

Standard IO-driver interface.

THis interface is used for I/O drivers written in C as well as I/O drivers written in IEC. IEC I/O drivers are typically written as a function block, which implementes the IIoDrv Interface.

To be able to access this IEC interface from C (which is for example necessary for the IoMgr), the component CmpIoDrvIec acts as a wrapper between C and IEC and implements exactly this interface.

23.2.1 Define: CT_PROGRAMMABLE

Category: Connector types
Type:
Define: CT_PROGRAMMABLE
Key: 0x1000

These are the connector types which are used most frequently. This list is not complete.

23.2.2 Define: CF_ENABLE

Category: Connector flags
Type:
Define: CF_ENABLE
Key: 0x0001
Flags that specifies diagnostic informations of a connector

23.2.3 Define: CO_NONE

Category: Connector options
Type:
Define: CO_NONE
Key: 0x0000
Options to specify properties of a connector

23.2.4 Define: PVF_FUNCTION

Category: Parameter value flags
Type:
Define: PVF_FUNCTION

Key: 0x0001

Defines the type of the parameter

Actually only PVF_POINTER is currently supported by CoDeSys

23.2.5 Define: FIP_NETX_DEVICE

Category: Fieldbus independent parameters
Type:
Define: FIP_NETX_DEVICE
Key: 0x70000000

23.2.6 Define: TMT_INPUTS

Category: Task map types
Type:
Define: TMT_INPUTS
Key: 0x0001

Types of IO-channels in a task map

23.2.7 Define: DRVPROP_CONSISTENCY

Category: Driver property flags
Type:
Define: DRVPROP_CONSISTENCY
Key: 0x0001

23.2.8 Define: CMLSI_BaseTypeMask

Category: Channel Map List Swapping Information
Type:
Define: CMLSI_BaseTypeMask
Key: 0x00FF

Every Io driver needs information on the base data type of an io channel to perform a correct swapping operation.The basedata type and some additional swapping information can be found in the wDummy Byte of the ChannelMapList struct.

23.2.9 IoDrvCreate

*RTS_HANDLE IoDrvCreate (RTS_HANDLE hIIoDrv, CLASSID ClassId, int iId, RTS_RESULT *pResult)*

Create a new I/O driver instance.

This function is obsolete, because the instance has to be created by the caller before he registers the I/O driver in the I/O Manager.

hIIoDrv [IN]
Handle to the IO-driver interface. Must be 0 and is filled automatically by calling the CAL_IoDrvCreate() macro!

ClassId [IN]
ClassID of the driver. See "Class IDs" section in CmpItf.h or in the Dep.h file of the IO-driver.

iId [IN]
Instance number of the IO-driver

pResult [OUT]
Pointer to error code

Result
Should return RTS_INVALID_HANDLE

23.2.10 IoDrvGetInfo

*RTS_RESULT IoDrvGetInfo (RTS_HANDLE hIoDrv, IoDrvInfo **ppIoDrv)*

Get a driver specific info structure.

This structure contains IDs and names of the driver.

hIoDrv [IN]
Handle to the IO-driver instance

pploDrv [OUT]
Pointer to pointer to the driver info. Pointer must be set by the driver to its internal driver info structure!

Result
Error code

23.2.11 IoDrvGetModuleDiagnosis

*RTS_RESULT IoDrvGetModuleDiagnosis (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Update Connector Flags in the device tree.

The driver should write the current diagnostic information (available, no driver, bus error,...) with the function IoDrvSetModuleDiagnosis() to the I/O connector.

This function can be used by other components or from the IEC application to update the diagnostic flags of the connector. To update the status from the driver, it has to call this function manually.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, that the diagostic information is requested

Result
Error code

23.2.12 IoDrvIdentify

*RTS_RESULT IoDrvIdentify (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Identify plugable I/O card or slave.

If the configurator supports scanning of modules, this function can be used our of a communication service to identify a module on the bus or locally on the PLC. This This might be done by a blinking LED or whatever the hardware supports.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, that should identify itself physically

Result
Error code

23.2.13 IoDrvReadInputs

*RTS_RESULT IoDrvReadInputs (RTS_HANDLE hIoDrv, IoConfigConnectorMap *pConnectorMapList, int nCount)*

Read inputs for one task

This function is called cyclically from every task that is using inputs. A part of the task map list, which contains only the data of one connector are passed to the driver (called connector map list).

If a driver has registered one instance to more than one connector, it might get more than one call with a different subset of the task map list.

The I/O driver should read the data from the local hardware or a buffer and write them to the corresponding IEC variables.

hIoDrv [IN]
Handle to the IO-driver instance

pConnectorMapList [IN]
Pointer to the connector map list

nCount [IN]
Number of entries in the connector map list

Result
Error code

23.2.14 IoDrvScanModules

*RTS_RESULT IoDrvScanModules (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector, IoConfigConnector **ppConnectorList, int *pnCount)*

Scan for submodules of a connector.

This function is executed when the driver is downloaded. It is called over a communication service.

The I/O driver should search for connected subumodules and return them via ppConnectorList.

NOTE: This interface is called synchronously and the buffer for the connector list has to be allocated by the driver.

The buffer might be freed at the next scan or at the next UpdateConfiguration.

hIoDrv [IN]
Handle to the IO-driver instance
pConnector [IN]
Pointer to the connector, which layout should be scanned
ppConnectorList [IN]
Pointer to the scanned connectors (devices) to return
pnCount [IN]
Pointer to the number of entries in the connector list to return
Result
Error code

23.2.15 IoDrvStartBusCycle

*RTS_RESULT IoDrvStartBusCycle (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Start bus cycle for a specific connector.

The bus cycle task is defined globally for the whole PLC or locally for a specific I/O connector in the CoDeSys project. This call can be used by the I/O driver to flush the I/O data if it was cached before.

This way we can get a better and consistent timing on the bus.

d

Note: This function is called for every connector which has a registered I/O driver and "needsbuscycle" set in the device description (this means that it might also be called for children of the connector).

Depending on the device description, this function might be executed at the beginning or at the end of the task cycle.

hIoDrv [IN]
Handle to the IO-driver instance
pConnector [IN]
Pointer to the connector, on which the buscycle must be triggered
Result
Error code

23.2.16 IoDrvUpdateConfiguration

*RTS_RESULT IoDrvUpdateConfiguration (RTS_HANDLE hIoDrv, IoConfigConnector *pConnectorList, int nCount)*

Propagate I/O configuration to the drivers.

This call passes the I/O configuration (based on the configuration tree in the CoDeSys programming system) to all registered I/O drivers. Every driver has the chance to pass this tree and to register itself for a specific connector.

The driver can use the I/O Manager Interface to iterate over the I/O Connectors and to read the I/O Parameters. If it decides to handle the I/Os of one of those connectors, it can register it's driver handle (IBase) to the connector in the member hIoDrv.

This function is called when the application is initialized as well as when it is de- or reinitialized. In this case it is called with pConnectorList = NULL.

hIoDrv [IN]
Handle to the IO-driver instance
pConnectorList [IN]

Pointer to the complete connector list

nCount [IN]
Number of entries in the connector list

Result
Error code

23.2.17 IoDrvUpdateMapping

*RTS_RESULT IoDrvUpdateMapping (RTS_HANDLE hIoDrv, IoConfigTaskMap *pTaskMapList, int nCount)*

Propagate the task map lists to the drivers.

This functions gives the drivers a chance to optimize their internal data structures based on the real task map lists. The function is called on every initialization of the application (download, bootproject,...).

hIoDrv [IN]
Handle to the IO-driver instance

pTaskMapList [IN]
Pointer to the task map list of one task

nCount [IN]
Number of entries in the map list

Result
Error code

23.2.18 IoDrvWatchdogTrigger

*RTS_RESULT IoDrvWatchdogTrigger (RTS_HANDLE hIoDrv, IoConfigConnector *pConnector)*

Trigger the hardware watchdog of a driver.

This function is depricated and not used anymore.

hIoDrv [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector, on which the watchdog should be retriggered

Result
Should return ERR_NOTIMPLEMENTED

23.2.19 IoDrvWriteOutputs

*RTS_RESULT IoDrvWriteOutputs (RTS_HANDLE hIoDrv, IoConfigConnectorMap *pConnectorMapList, int nCount)*

Write outputs for one task

This function is called cyclically from every task that is using outputs. A part of the task map list, which contains only the data of one connector are passed to the driver (called connector map list).

If a driver has registered one instance to more than one connector, it might get more than one call with a different subset of the task map list.

The I/O driver should write out the data to the local hardware, a buffer or a fieldbus.

hIoDrv [IN]
Handle to the IO-driver instance

pConnectorMapList [IN]
Pointer to the connector map list

nCount [IN]
Number of entries in the connector map list

Result
Error code

23.2.20 IoDrvDelete

RTS_RESULT IoDrvDelete (RTS_HANDLE hIoDrv, RTS_HANDLE hIIoDrv)

Delete an I/O driver instance.

This function is obsolete, because the instance has to be deleted by the caller after he unregisters the I/O driver from the I/O Manager.

Delete an IO-driver instance

hIoDrv [IN]
Handle to the IO-driver instance

hIIoDrv [IN]
Handle of the ITFID_ICmpIoDrv interface

Result
Should return ERR_NOTIMPLEMENTED

23.3 CmpIoDrvParameterItf

Interface to access parameters of an IO-driver.

23.3.1 IoDrvReadParameter

*RTS_RESULT IoDrvReadParameter (RTS_HANDLE hDevice, IoConfigConnector *pConnector, IoConfigParameter *pParameter, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Read a driver specific parameters.

These parameters can be read by the application, as well as, by an online service, which is triggered from the device configurator plugin in the CoDeSys programming system.

Note: If the I/O driver returns an error, the I/O Manager may try to read the parameter himself

Note2: On SIL2 runtimes, this interface is not supported.

hDevice [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector (might be determined with IoMgrConfigGetConnector).

pParameter [IN]
Pointer to the parameter (might be determined with IoMgrConfigGetParameter)

pData [IN]
Buffer where the read data is stored

ulBitSize [IN]
Size of the part of the parameter data, that should be read

ulBitOffset [IN]
Offset of the part of the parameter, that should be read

Result
Error code

23.3.2 IoDrvWriteParameter

*RTS_RESULT IoDrvWriteParameter (RTS_HANDLE hDevice, IoConfigConnector *pConnector, IoConfigParameter *pParameter, void *pData, RTS_SIZE ulBitSize, RTS_SIZE ulBitOffset)*

Write a driver specific parameters.

These parameters can be written by the application, as well as, by an online service, which is triggered from the device configurator plugin in the CoDeSys programming system.

Note: On SIL2 runtimes, this interface is not supported. And if called in safe-mode, on SIL2 Runtimes, an exception is generated.

hDevice [IN]
Handle to the IO-driver instance

pConnector [IN]
Pointer to the connector (might be determined with IoMgrConfigGetConnector).

pParameter [IN]
Pointer to the parameter (might be determined with IoMgrConfigGetParameter)

pData [IN]
Buffer where the read data is stored

ulBitSize [IN]
Size of the part of the parameter data, that should be written

ulBitOffset [IN]
Offset of the part of the parameter, that should be written
Result
Error code

24 SysCom

System component that allows access to the serial interface.

24.1 SysComItf

The SysCom interface is projected to connect to a serial COM port (RS232) and to send and receive data via this port.

IMPLEMENTATION NODE: All routines must be realized asynchronous! You have to use the FIFO of the serial device! Don't block the read and write routines, until the operations are finished.

24.1.1 Define: EVT_SysComOpenBefore

Category: Events
Type:
Define: EVT_SysComOpenBefore
Key: MAKE_EVENTID
Platform dependent event. Sent before the physical driver open.

24.1.2 Define: EVT_SysComOpenAfter

Category: Events
Type:
Define: EVT_SysComOpenAfter
Key: MAKE_EVENTID
Platform dependent event. Sent directly after the physical driver open.

24.1.3 Define: SYS_NOWAIT

Category: Timeouts
Type:
Define: SYS_NOWAIT
Key: 0UL

24.1.4 Define: SYS_BR_4800

Category: Baudrates
Type:
Define: SYS_BR_4800
Key: 4800UL

24.1.5 Typedef: EVTPARAM_SysComOpen

Stuctname: EVTPARAM_SysComOpen
Category: Event parameter
Typedef: typedef struct { unsigned char* name; } EVTPARAM_SysComOpen;

24.1.6 Typedef: COM_SettingsEx

Stuctname: COM_SettingsEx
Category: Com port extended settings
Typedef: typedef struct tagCOM_SettingsEx { RTS_IEC_BYTE byByteSize; RTS_IEC_BOOL bBinary; RTS_IEC_BOOL bOutxCtsFlow; RTS_IEC_BOOL bOutxDsrFlow; RTS_IEC_BOOL bDtrControl; RTS_IEC_BOOL bDsrSensitivity; RTS_IEC_BOOL bRtsControl; RTS_IEC_BOOL bTXContinueOnXoff; RTS_IEC_BOOL bOutX; RTS_IEC_BOOL bInX; RTS_IEC_BYTE byXonChar; RTS_IEC_BYTE byXoffChar; RTS_IEC_WORD wXonLim; RTS_IEC_WORD wXoffLim; } COM_SettingsEx;

24.1.7 Typedef: COM_Settings

Stuctname: COM_Settings
Category: Com port settings
Typedef: typedef struct tagCOM_Settings { RTS_IEC_INT sPort; RTS_IEC_BYTE byStopBits; RTS_IEC_BYTE byParity; RTS_IEC_DWORD ulBaudrate; RTS_IEC_UDINT ulTimeout; RTS_IEC_UDINT ulBufferSize; } COM_Settings;

24.1.8 Typedef: syscompurge_struct

Stuctname: syscompurge_struct
syscompurge
Typedef: typedef struct tagsyscompurge_struct { RTS_IEC_BYTE *hCom; VAR_INPUT RTS_IEC_UDINT SysComPurge; VAR_OUTPUT } syscompurge_struct;

24.1.9 Typedef: syscomread_struct

Stuctname: syscomread_struct

syscomread

Typedef: typedef struct tagsyscomread_struct { RTS_IEC_BYTE *hCom; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_UDINT ulSize; VAR_INPUT RTS_IEC_UDINT ulTimeout; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_UDINT SysComRead; VAR_OUTPUT } syscomread_struct;

24.1.10 Typedef: syscomclose_struct

Stuctname: syscomclose_struct

syscomclose

Typedef: typedef struct tagsyscomclose_struct { RTS_IEC_BYTE *hCom; VAR_INPUT RTS_IEC_UDINT SysComClose; VAR_OUTPUT } syscomclose_struct;

24.1.11 Typedef: syscomwrite_struct

Stuctname: syscomwrite_struct

syscomwrite

Typedef: typedef struct tagsyscomwrite_struct { RTS_IEC_BYTE *hCom; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_UDINT ulSize; VAR_INPUT RTS_IEC_UDINT ulTimeout; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_UDINT SysComWrite; VAR_OUTPUT } syscomwrite_struct;

24.1.12 Typedef: syscomopen_struct

Stuctname: syscomopen_struct

Open a serial communication device

Typedef: typedef struct tagsyscomopen_struct { RTS_IEC_INT sPort; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysComOpen; VAR_OUTPUT } syscomopen_struct;

24.1.13 Typedef: syscomsettimeout_struct

Stuctname: syscomsettimeout_struct

syscomsettimeout

Typedef: typedef struct tagsyscomsettimeout_struct { RTS_IEC_BYTE *hCom; VAR_INPUT RTS_IEC_UDINT ulTimeout; VAR_INPUT RTS_IEC_UDINT SysComSetTimeout; VAR_OUTPUT } syscomsettimeout_struct;

24.1.14 Typedef: syscomgetsettings_struct

Stuctname: syscomgetsettings_struct

syscomgetsettings

Typedef: typedef struct tagsyscomgetsettings_struct { RTS_IEC_BYTE *hCom; VAR_INPUT COM_Settings *pSettings; VAR_INPUT COM_SettingsEx *pSettingsEx; VAR_INPUT RTS_IEC_UDINT SysComGetSettings; VAR_OUTPUT } syscomgetsettings_struct;

24.1.15 Typedef: syscomsetsettings_struct

Stuctname: syscomsetsettings_struct

syscomsetsettings

Typedef: typedef struct tagsyscomsetsettings_struct { RTS_IEC_BYTE *hCom; VAR_INPUT COM_Settings *pSettings; VAR_INPUT COM_SettingsEx *pSettingsEx; VAR_INPUT RTS_IEC_UDINT SysComSetSettings; VAR_OUTPUT } syscomsetsettings_struct;

24.1.16 Typedef: syscomopen2_struct

Stuctname: syscomopen2_struct

syscomopen2

Typedef: typedef struct tagsyscomopen2_struct { COM_Settings *pSettings; VAR_INPUT COM_SettingsEx *pSettingsEx; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysComOpen2; VAR_OUTPUT } syscomopen2_struct;

24.1.17 Typedef: COM_Parity

Category: Parity

Typedef: typedef enum { SYS_NOPARITY = 0, SYS_ODDPARITY = 1, SYS_EVENPARITY = 2 } COM_Parity;

24.1.18 Typedef: COM_StopBits

Category: Stop bits

Typedef: typedef enum { SYS_ONESTOPBIT = 1, SYS_ONE5STOPBITS = 2, SYS_TWOSTOPBITS = 3 } COM_StopBits;

24.1.19 Typedef: COM_Ports

Category: Com ports

Com port numbers

Typedef: typedef enum { SYS_COMPORT_NONE = 0, SYS_COMPORT1 = 1, SYS_COMPORT2 = 2, SYS_COMPORT3 = 3, SYS_COMPORT4 = 4 } COM_Ports;

24.1.20 SysComOpen

*RTS_HANDLE SysComOpen (short sPort, RTS_RESULT *pResult)*

Open a serial communication device

sPort [IN]
Number of the Port to open: 1=COM1, 2=COM2, ...

pResult [OUT]
Pointer to error code

Result
Handle to the interface or RTS_INVALID_HANDLE if failed

24.1.21 SysComOpen2

*RTS_HANDLE SysComOpen2 (COM_Settings *pSettings, COM_SettingsEx *pSettingsEx, RTS_RESULT *pResult)*

Opens a serial communication device specified with settings

pSettings [IN]
Settings for the communication device. See category "Com port settings" for detailed information

pSettingsEx [IN]
Optional extended settings for the serial device. Can be NULL.

pResult [OUT]
Pointer to error code

Result
Handle to the interface or RTS_INVALID_HANDLE if failed

24.1.22 SysComSetSettings

*RTS_RESULT SysComSetSettings (RTS_HANDLE hCom, COM_Settings *pSettings, COM_SettingsEx *pSettingsEx)*

Set the parameter of the interface specified by handle

hCom [IN]
Handle to Com Port

pSettings [IN]
Pointer to new settings

pSettingsEx [IN]
Pointer to extended settings

Result
error code

24.1.23 SysComGetSettings

*RTS_RESULT SysComGetSettings (RTS_HANDLE hCom, COM_Settings *pSettings, COM_SettingsEx *pSettingsEx)*

Get the parameter of the interface specified by handle

hCom [IN]
Handle to Com Port

pSettings [IN]
Pointer to new settings

pSettingsEx [IN]
Pointer to extended settings

Result
error code

24.1.24 SysComRead

*unsigned int SysComRead (RTS_HANDLE hCom, unsigned char *pbyBuffer, unsigned int uiSize, unsigned long ulTimeout, RTS_RESULT *pResult)*

Reads a number bytes from the specifed device to the receive buffer.

IMPLEMENTATION NOTE: If the timeout elapsed until the requested number of bytes are received, the function returns with the bytes still received! This must be considered in the caller and the implementation!

hCom [IN]
Handle to com port

pbyBuffer [IN]
Pointer to data buffer for received data

uiSize [IN]
Requested number of bytes to read. Must be less or equal the size of the receive buffer!

ulTimeout [IN]
Timeout to read data from the device. 0=Immediate return. If the timeout elapsed, the function returns with the still received data (could be less then the requested number of bytes!)

pResult [IN]
Pointer to error code, ERR_TIMEOUT if timeout elapsed

Result
Number of actually read bytes

24.1.25 SysComWrite

*unsigned int SysComWrite (RTS_HANDLE hCom, unsigned char *pbyBuffer, unsigned int uiSize, unsigned long ulTimeout, RTS_RESULT *pResult)*

Writes a number bytes to the specifed device from the sent buffer.

IMPLEMENTATION NOTE: If the timeout elapsed until the requested number of bytes are sent, the function returns with the bytes still sent! This must be considered in the caller and the implementation!

hCom [IN]
Handle to com port

pbyBuffer [IN]
Pointer to data buffer for sent data

uiSize [IN]
Requested number of bytes to sent. Must be less or equal the size of the sent buffer!

ulTimeout [IN]
Timeout to sent data to the device. 0=Immediate return. If the timeout elapsed, the function returns with the still sent data (could be less then the requested number of bytes!)

pResult [IN]
Pointer to error code, ERR_TIMEOUT if timeout elapsed

Result
Number of actually written bytes

24.1.26 SysComPurge

RTS_RESULT SysComPurge (RTS_HANDLE hCom)

Clear the fifo buffer of the serial interface

hCom [IN]
Handle to com port

Result
error code

24.1.27 SysComSetTimeout

RTS_RESULT SysComSetTimeout (RTS_HANDLE hCom, unsigned long ulTimeout)

Set the timeout of the specified serial interface (hardware timeout). The Timeout is the time between two received or sent characters until the read or write operation will return. Typically this value should be 0 (returns immediately)

hCom [IN]
Handle to com port

ulTimeout [IN]
Timeout to set

Result
error code

24.1.28 SysComClose

RTS_RESULT SysComClose (RTS_HANDLE hCom)

Close a serial communication device

hCom [IN]
Handle to com port

Result
error code

25 SysCpuHandling

System component that allows access to Cpu specific features.

25.1 SysCpuHandlingItf

The SysCpuHandling interface contains all cpu specific routines.

To detect, for which platform the component is compiled, there are special defines that must be set in sysdefines.h dependant of the compiler specific options (see category "Processor ID" in SysTargetItf.h)

25.1.1 Typedef: syscpucalliecfuncwithparams_struct

Stuctname: syscpucalliecfuncwithparams_struct

Call an IEC function from plain C code. Since different CPU's/systems use different calling conventions, this function should be used as a wrapper.

IEC functions or methods of function block use all the same calling convention: They have no return value and exactly one parameter, which is a pointer to a struct that contains all required IN and OUT parameters.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsyscpucalliecfuncwithparams_struct { RTS_VOID_FCTPTR pfIECFunc; VAR_INPUT RTS_IEC_BYTE *pParam; VAR_INPUT RTS_IEC_UDINT ulSize; VAR_INPUT RTS_IEC_UDINT SysCpuCallIecFuncWithParams; VAR_OUTPUT } syscpucalliecfuncwithparams_struct;

25.1.2 Typedef: syscputestandset_struct

Stuctname: syscputestandset_struct

Test and set a bit in an ULONG variable in one processor step. This operation is to provide a multitasking save operation.

RESULT: Returns the runtime system error code (see CmpErrors.library). ERR_OK: If bit could be set and was set before, ERR_FAILED: If bit is still set

Typedef: typedef struct tagsyscputestandset_struct { RTS_IEC_UDINT *pulValue; VAR_INPUT RTS_IEC_UDINT ulBit; VAR_INPUT RTS_IEC_UDINT SysCpuTestAndSet; VAR_OUTPUT } syscputestandset_struct;

25.1.3 Typedef: syscputestandreset_struct

Stuctname: syscputestandreset_struct

Test and reset a bit in an ULONG variable in one processor step. This operation is to provide a multitasking save operation.

RESULT: Returns the runtime system error code (see CmpErrors.library). ERR_OK: If bit could be reset and was set before, ERR_FAILED: If bit is still reset

Typedef: typedef struct tagsyscputestandreset_struct { RTS_IEC_UDINT *pulValue; VAR_INPUT RTS_IEC_UDINT ulBit; VAR_INPUT RTS_IEC_UDINT SysCpuTestAndReset; VAR_OUTPUT } syscputestandreset_struct;

25.1.4 Typedef: syscputestandsetbit_struct

Stuctname: syscputestandsetbit_struct

The function test and set or clear a bit in a variable in one processor step. This operation must be atomic to provide a multitasking save operation.

IMPLEMENTATION NOTE: Try to use a processor opcode, that provides this operation. If such an opcode is not available, use SysCpuTestAndSetBitBase in your platform adaptation.

RESULT: Returns the runtime system error code (see CmpErrors.library).

- ERR_OK: bSet=1: If bit could be set and was not set before bSet=0: If bit could be cleared and was set before
- ERR_FAILED: bSet=1: If bit is still set bSet=0: If bit is still cleared

- ERR_PARAMETER: If pAddress=NULL or pAddress is unaligned or iBit is out nSize range
- ERR_NOT_SUPPORTED: If function is not available because of missing components (e.g. SysInt for locking bit access)
- ERR_NOTIMPLEMENTED: If function is not implemented on this platform

Typedef: typedef struct tagsyscputestandsetbit_struct { RTS_IEC_BYTE *pAddress; VAR_INPUT RTS_IEC_UDINT nLen; VAR_INPUT RTS_IEC_DINT iBit; VAR_INPUT RTS_IEC_DINT bSet; VAR_INPUT RTS_IEC_UDINT SysCpuTestAndSetBit; VAR_OUTPUT } syscputestandsetbit_struct;

25.1.5 Typedef: syscpuatomicadd_struct

Stuctname: syscpuatomicadd_struct

Function to increment the content of the given pointer by 1 in one atomic operation (task safe). IMPLEMENTATION NOTE: - Add/Sub the value to the content of the pointer - Return the value after the Add/Sub operation Both things must be done atomic!

RESULT: Returns the value after the increment operation in an atomic way!

Typedef: typedef struct tagsyscpuatomicadd_struct { RTS_IEC_DINT *piValue; VAR_INPUT RTS_IEC_DINT nSum; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysCpuAtomicAdd; VAR_OUTPUT } syscpuatomicadd_struct;

25.1.6 syscpudebughandler

void syscpudebughandler (void)

Routine is the entry of the IEC code debugger. Is called out of the IEC task if breakpoint is reached.

IMPLEMENTATION NOTE: In this routine, the AppDebugHandler() function of CmpApp must be called with the appropriate parameters. See CmpAppItf.h for detailed information. The return value of AppDebugHandler() must be used to set the return address in the IEC code. To this address we will return when leaving syscpudebughandler().

SIL2 IMPLEMENTATION NOTE: This routine may never be reached within safety mode. If it is called in safety mode an Exception must be generated immediately! The Execution may not proceed to further debug mechanism!

Result

no return value

25.1.7 SysCpuFlushInstructionCache

*int SysCpuFlushInstructionCache (void * pBaseAddress, unsigned long ulSize)*

On some processors/operating systems this function must be called after code has been changed, so that the CPU is notified about the change.

pBaseAddress [IN]

Start address of the region to be flushed. Set to NULL to flush the complete instruction cache.

ulSize [IN]

Length of the region to be flushed. This parameter is ignored if pBaseAddress is NULL.

Result

error code

25.1.8 SysCpuCallIecFuncWithParams

RTS_RESULT SysCpuCallIecFuncWithParams (RTS_VOID_FCTPTR pfIECFunc, void pParam, int iSize)*

Call an IEC function from plain C code. Since different CPU's/systems use different calling conventions, this function should be used as a wrapper.

IEC functions or methods of function block use all the same calling convention: They have no return value and exactly one parameter, which is a pointer to a struct that contains all required IN and OUT parameters.

IMPLEMENTATION NOTE: The content of the parameter structure must be copied completely on the stack as an input parameter! Don't copy only the pointer! Because of this, the size of the structure is provided as a separate parameter to this function. Additionally, the structure of the IEC function must be copied back into the give parameter to return result values of the IEC function! For all this operations you have to ensure the stack alignment, but avoid copying more bytes than iSize

IMPLEMENTATION NOTE: Unused parameter pParam can be NULL, if function has no argument and no result (e.g. CodeInit)!

pfIECFunc [IN]
Pointer to the IEC function that should be called

pParam [INOUT]
Pointer to the parameter struct that contains the function parameters. ATTENTION: Can be NULL!

iSize [IN]
Size of the parameter structure to copy the content on stack. ATTENTION: Can be 0!

Result
error code

25.1.9 SysCpuGetBreakpointCode

RTS_RESULT SysCpuGetBreakpointCode (unsigned char pbyAreaStart, unsigned char* pbyBPAddress, unsigned char* pbyOpCode, int *piOpcodeSize)*

Routine retrieves the breakpoint opcode for IEC code debugging. This is cpu dependant and depends additionally on the CoDeSys codegenerator.

pbyAreaStart [IN]
Pointer to start of the area.

pbyBPAddress [IN]
Pointer to breakpoint address (where the breakpoint will be set). This parameter is used for breakpoints that uses absolut jumps.

pbyOpCode [OUT]
Pointer to get opcode

piOpcodeSize [INOUT]
Pointer to maximum size of opcode buffer and return the real size of the opcode

Result
error code

25.1.10 SysCpuGetCallstackEntry

*RTS_RESULT SysCpuGetCallstackEntry (RTS_UINTPTR *pBP, void **ppAddress)*

Routine retrieves an IEC callstack entry, if the IEC code execution is stopped in the IEC code (breakpoint, exception, etc.). The callstack can be investigated out of the stack frames of each entered nested function. The first callstack entry is calculated outside this routine from the instruction pointer address (IP) that is provided for the AppDebugHandler() routine from syscpudebughandler(). All further entries are called from this routine.

IMPLEMENTATION NOTE: The stack frame has typically the following structure:

. STACK CODE . ----- ---- . . BP0 /--------| Fct0: . | ... . Return Fct0 | Call Fct1 . |--\ BP1 ---------| Return Fct0 . | . | Retur

In this example, Fct0 calls Fct1 and Fct1 calls Fct2. This is a nested call with 3 Functions. Inside Fct2 we do a halt (e.g. stop on breakpoint). If we take a look on the stack, we see the return addresses from Fct0 and Fct1 on the stack. At the entry of each function, the base- (or frame-) pointer is pushed on the stack with the previous content. So you can see, the BP entries on the stack are organized as a chained list from stackframe to stack frame.

The first stack entry is the current position. This is calculated out of the IP address and cannot be investigated from stack.

The pulBP Parameter is a pointer, thats content is the address of the stack, where the BP entry resides.

pBP [INOUT]
Pointer to last base pointer entry (or frame pointer) and returns the pointer to the next base pointer in the stack frame

ppAddress [OUT]
Pointer pointer to return address in the code of the caller

Result
error code

25.1.11 SysCpuGetCallstackEntry2

*RTS_RESULT SysCpuGetCallstackEntry2 (int bIecCode, RTS_UINTPTR *pBP, void **ppAddress)*

Routine retrieves a callstack entry. The additional parameter bIecCode specifies, if the callstack should be retrieved in IEC-code (bIecCode = 1) or in C-code of the runtime system (bIecCode = 0). IMPLEMENTATION NOTE: The callstack in C-code is sometimes compiler dependant and must be implemented here specific to the compiler.

bIecCode [IN]

pBP [INOUT]
Pointer to last base pointer entry (or frame pointer) and returns the pointer to the next base pointer in the stack frame

ppAddress [OUT]
Pointer pointer to return address in the code of the caller

Result
error code

25.1.12 SysCpuGetInstancePointer

*RTS_RESULT SysCpuGetInstancePointer (RTS_UINTPTR *pBP, void** ppInstancePointer)*

Routine to retrieve the instance pointer of an FB. Is used to get an instance specific callstack. The position of the instance pointer depends on the CoDeSys codegenerator and is cpu dependant.

pBP [INOUT]
Pointer to last base pointer entry (or frame pointer) and returns the pointer to the next base pointer in the stack frame

ppInstancePointer [OUT]
Pointer pointer to the instance pointer of an FB

Result
error code

25.1.13 SysCpuGetMonitoringBase

*RTS_RESULT SysCpuGetMonitoringBase (RTS_UINTPTR *pBP)*

Routine to retrieve the frame pointer for monitoring data access.

pBP [INOUT]
Pointer to last base pointer entry (or frame pointer) and returns the pointer on the stack for monitoring data access

Result
error code

25.1.14 SysCpuTestAndSet

RTS_RESULT SysCpuTestAndSet (RTS_UI32 pul, int iBit)*

Obsolete: Use SysCpuTestAndSetBit instead!

OBSOLETE function to test and set a bit in an ULONG variable in one processor step. This operation must be atomic to provide a multitasking save operation.

IMPLEMENTATION NOTE: Try to use a processor opcode, that provides this operation. If such an opcode is not available, use SysCpuTestAndSetBase in your platform adaptation.

pul [IN]
Pointer to the unsigned value to test an set a bit inside in one atomic processor step

Result
Error code:
- ERR_OK: If bit could be set and was not set before
- ERR_FAILED: If bit is still set

25.1.15 SysCpuTestAndReset

RTS_RESULT SysCpuTestAndReset (RTS_UI32 pul, int iBit)*

Obsolete: Use SysCpuTestAndSetBit instead!

OBSOLETE function to reset a bit in an ULONG variable in one processor step. This operation must be atomic to provide a multitasking save operation.

IMPLEMENTATION NOTE: Try to use a processor opcode, that provides this operation. If such an opcode is not available, use SysCpuTestAndResetBase in your platform adaptation.

pul [IN]
Pointer to the unsigned value to test an reset a bit inside in one atomic processor step

iBit [IN]
Bit number inside the variable to test and reset. 0=first bit, 31=last bit

Result
- ERR_OK: If bit could be reset
- ERR_FAILED: If bit reset failed

25.1.16 SysCpuTestAndSetBit

RTS_RESULT SysCpuTestAndSetBit (void pAddress, int nLen, int iBit, int bSet)*

The function test and set or clear a bit in a variable in one processor step. This operation must be atomic to provide a multitasking save operation.

IMPLEMENTATION NOTE: Try to use a processor opcode, that provides this operation. If such an opcode is not available, use SysCpuTestAndSetBitBase in your platform adaptation. The function returns ERR_FAILED if the bit was already set or reset.

pAddress [IN]
Pointer to test and set or clear a bit inside in one atomic processor step. NOTE: The pointer must be natural aligned! nLen=2: pAddress must be 2Byte aligned; nLen=4: pAddress must be 4Byte aligned

nLen [IN]
Size of the value behind the address. Can only be 1 (unsigned char), 2 (unsigned short) or 4 (unsigned long)

iBit [IN]
Bit number inside the variable to test and set or clear:
- nLen = 1: iBit 0..7
- nLen = 2: iBit 0..15
- nLen = 4: iBit 0..31

bSet [IN]
1=Set bit, 0=Clear bit

iBit [IN]
Bit number inside the variable to test and set. 0=first bit, 31=last bit

Result
Error code: returns if Bit could be set/reset successfully, or if any problem occured

25.1.17 SysCpuGetContext

*RTS_RESULT SysCpuGetContext (RegContext *pContext)*

Get the actual register context.

pContext [OUT]
Actual register context

Result
error code

25.1.18 SysCpuAtomicAdd

*RTS_I32 SysCpuAtomicAdd (RTS_I32 *piValue, RTS_I32 nSum, RTS_RESULT *pResult)*

Function to increment the content of the given pointer by nSum in one atomic operation (task safe).

IMPLEMENTATION NOTE: Add or substract the value to/from the content of the pointer, and return the value after this operation atomically.

piValue [INOUT]
Pointer to the value to increment

nSum [IN]
Summand for the operation. greater 0 to increment, lower 0 to decrement

pResult [OUT]
Pointer to error code

Result
Returns the value after the increment operation in an atomic way!

26 SysExcept

Component for first level exception handling

Compiler Switch

- #define RTS_STRUCTURED_EXCEPTION_HANDLING Switch to enabled structired exception handling. See rts_try/rts_catch for details.

26.1 SysExceptItf

The SysExcept interface is projected to handle all exceptions in the runtime system. All available exceptions on the target should be handled. Exception handling is typically cpu and operating system dependant.

All active code parts in the runtime system (tasks, timer, interrupt handler) must be protected against software errors, because this could lead to an unpredictable behaviour or crash of the complete runtime system.

All exceptions are handled in this component as first level. If the first level handling is done in another component, the component has to call the SysExceptGenerateException() routine to enable second level handling. In this case, the first level handling must be disabled with a config setting (see below).

The SysExcept component forwards every exception to the component, that manages the active code, mostly the SysTask, SysTimer or SysInt component. These components must register exception handlers to the SysExcept component.

As the last station of exception handling, the exception handler of the component, that creates the active object, was called from the SysTask, SysTimer or SysInt component. Below you can see the structure and the layers of exception handling:

. +----------+ . | XXX component: Code that generates | . | Registered exceptionha

If RTS_STRUCTURED_EXCEPTION_HANDLING is set as a define, the structured exception handling is activated. To use this exception handling, the following functions must be imported:

- SysExceptRegisterJmpBuf
- SysExceptCatch
- SysExceptUnregisterJmpBuf

Here is an example of the usage:

. Variant 1: . ---------- . rts_try . { . ... . } . rts_catch_first(RTSEXCPT_DIVIDEBYZERO) . { . RTS_UI32 exceptionCode

26.1.1 Define: EXCPT_DEFAULT_NUM_OF_INTERFACES

Condition: #ifndef EXCPT_DEFAULT_NUM_OF_INTERFACES
Category: Static defines
Type:
Define: EXCPT_DEFAULT_NUM_OF_INTERFACES
Key: 5
Default number of static interfaces that can be registered on exceptions

26.1.2 Define: EXCPT_MAX_NUM_OF_SEH_HANDLER

Condition: #ifndef EXCPT_MAX_NUM_OF_SEH_HANDLER
Category: Static defines
Type:
Define: EXCPT_MAX_NUM_OF_SEH_HANDLER
Key: 5
Maximum number of nested structured exception handler on the same task

26.1.3 Define: EXCPT_NUM_OF_STATIC_CONTEXT

Condition: #ifndef EXCPT_NUM_OF_STATIC_CONTEXT
Category: Static defines
Type:
Define: EXCPT_NUM_OF_STATIC_CONTEXT
Key: 10
Maximum number of static contexts to store for structured exception handling

26.1.4 Define: RTSEXCPT_UNKNOWN
Category: Exception code
Type:
Define: RTSEXCPT_UNKNOWN
Key: UINT32_MAX
Invalid

26.1.5 Define: RTSEXCPT_NOEXCEPTION
Category: Exception code
Type:
Define: RTSEXCPT_NOEXCEPTION
Key: 0x00000000
No exception

26.1.6 Define: RTSEXCPT_WATCHDOG
Category: Exception code
Type:
Define: RTSEXCPT_WATCHDOG
Key: 0x00000010
Software watchdog of IEC-task expired

26.1.7 Define: RTSEXCPT_HARDWAREWATCHDOG
Category: Exception code
Type:
Define: RTSEXCPT_HARDWAREWATCHDOG
Key: 0x00000011
Hardware watchdog expired. Global software error

26.1.8 Define: RTSEXCPT_IO_CONFIG_ERROR
Category: Exception code
Type:
Define: RTSEXCPT_IO_CONFIG_ERROR
Key: 0x00000012
IO config error

26.1.9 Define: RTSEXCPT_PROGRAMCHECKSUM
Category: Exception code
Type:
Define: RTSEXCPT_PROGRAMCHECKSUM
Key: 0x00000013
Checksum error after program download

26.1.10 Define: RTSEXCPT_FIELDBUS_ERROR
Category: Exception code
Type:
Define: RTSEXCPT_FIELDBUS_ERROR
Key: 0x00000014
Fieldbus error

26.1.11 Define: RTSEXCPT_IOUPDATE_ERROR
Category: Exception code
Type:
Define: RTSEXCPT_IOUPDATE_ERROR
Key: 0x00000015
IO-update error

26.1.12 Define: RTSEXCPT_CYCLE_TIME_EXCEED
Category: Exception code
Type:
Define: RTSEXCPT_CYCLE_TIME_EXCEED
Key: 0x00000016
Cycle time exceed

26.1.13 Define: RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED
Category: Exception code

Type:
Define: RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED
Key: 0x00000017
Online change program too large

26.1.14 Define: RTSEXCPT_UNRESOLVED_EXTREFS

Category: Exception code
Type:
Define: RTSEXCPT_UNRESOLVED_EXTREFS
Key: 0x00000018
Unresolved external references

26.1.15 Define: RTSEXCPT_DOWNLOAD_REJECTED

Category: Exception code
Type:
Define: RTSEXCPT_DOWNLOAD_REJECTED
Key: 0x00000019
Download was rejected

26.1.16 Define: RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR

Category: Exception code
Type:
Define: RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR
Key: 0x0000001A
Boot project not loaded because retain variables could not be relocated

26.1.17 Define: RTSEXCPT_LOADBOOTPROJECT_FAILED

Category: Exception code
Type:
Define: RTSEXCPT_LOADBOOTPROJECT_FAILED
Key: 0x0000001B
Boot project not loaded and deleted

26.1.18 Define: RTSEXCPT_OUT_OF_MEMORY

Category: Exception code
Type:
Define: RTSEXCPT_OUT_OF_MEMORY
Key: 0x0000001C
Out of heap memory

26.1.19 Define: RTSEXCPT_RETAIN_MEMORY_ERROR

Category: Exception code
Type:
Define: RTSEXCPT_RETAIN_MEMORY_ERROR
Key: 0x0000001D
Retain memory corrupt or cannot be mapped

26.1.20 Define: RTSEXCPT_BOOTPROJECT_CRASH

Category: Exception code
Type:
Define: RTSEXCPT_BOOTPROJECT_CRASH
Key: 0x0000001E
Boot project that could be loaded but caused a crash later

26.1.21 Define: RTSEXCPT_BOOTPROJECTTARGETMISMATCH

Category: Exception code
Type:
Define: RTSEXCPT_BOOTPROJECTTARGETMISMATCH
Key: 0x00000021
Target of the bootproject doesn't match the current target

26.1.22 Define: RTSEXCPT_SCHEDULEERROR

Category: Exception code
Type:
Define: RTSEXCPT_SCHEDULEERROR
Key: 0x00000022

Error at scheduling tasks

26.1.23 Define: RTSEXCPT_FILE_CHECKSUM_ERR

Category: Exception code
Type:
Define: RTSEXCPT_FILE_CHECKSUM_ERR
Key: 0x00000023
Checksum of downloaded file does not match

26.1.24 Define: RTSEXCPT_RETAIN_IDENTITY_MISMATCH

Category: Exception code
Type:
Define: RTSEXCPT_RETAIN_IDENTITY_MISMATCH
Key: 0x00000024
Retain identity does not match to current bootproject program identity

26.1.25 Define: RTSEXCPT_IEC_TASK_CONFIG_ERROR

Category: Exception code
Type:
Define: RTSEXCPT_IEC_TASK_CONFIG_ERROR
Key: 0x00000025
Iec task configuration failed

26.1.26 Define: RTSEXCPT_APP_TARGET_MISMATCH

Category: Exception code
Type:
Define: RTSEXCPT_APP_TARGET_MISMATCH
Key: 0x00000026
Application is running on wrong target. Can be used for library protection!

26.1.27 Define: RTSEXCPT_ILLEGAL_INSTRUCTION

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_ILLEGAL_INSTRUCTION
Key: 0x00000050
Illegal instruction

26.1.28 Define: RTSEXCPT_ACCESS_VIOLATION

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_ACCESS_VIOLATION
Key: 0x00000051
Access violation

26.1.29 Define: RTSEXCPT_PRIV_INSTRUCTION

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_PRIV_INSTRUCTION
Key: 0x00000052
Privileged instruction

26.1.30 Define: RTSEXCPT_IN_PAGE_ERROR

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_IN_PAGE_ERROR
Key: 0x00000053
Page fault

26.1.31 Define: RTSEXCPT_STACK_OVERFLOW

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_STACK_OVERFLOW
Key: 0x00000054
Stack overflow

26.1.32 Define: RTSEXCPT_INVALID_DISPOSITION

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_INVALID_DISPOSITION
Key: 0x00000055
Invalid disposition

26.1.33 Define: RTSEXCPT_INVALID_HANDLE

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_INVALID_HANDLE
Key: 0x00000056
Invalid handle

26.1.34 Define: RTSEXCPT_GUARD_PAGE

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_GUARD_PAGE
Key: 0x00000057
Guard page

26.1.35 Define: RTSEXCPT_DOUBLE_FAULT

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_DOUBLE_FAULT
Key: 0x00000058
Double fault

26.1.36 Define: RTSEXCPT_INVALID_OPCODE

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_INVALID_OPCODE
Key: 0x00000059
Invalid OpCode

26.1.37 Define: RTSEXCPT_MISALIGNMENT

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_MISALIGNMENT
Key: 0x00000100
Datatype misalignment

26.1.38 Define: RTSEXCPT_ARRAYBOUNDS

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_ARRAYBOUNDS
Key: 0x00000101
Array bounds exceeded

26.1.39 Define: RTSEXCPT_DIVIDEBYZERO

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_DIVIDEBYZERO
Key: 0x00000102
Division by zero

26.1.40 Define: RTSEXCPT_OVERFLOW

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_OVERFLOW
Key: 0x00000103
Overflow

26.1.41 Define: RTSEXCPT_NONCONTINUABLE

Category: Exception code, hardware exceptions
Type:
Define: RTSEXCPT_NONCONTINUABLE

Key: 0x00000104
Non continuable

26.1.42 Define: RTSEXCPT_PROCESSORLOAD_WATCHDOG

Category: Exception code
Type:
Define: RTSEXCPT_PROCESSORLOAD_WATCHDOG
Key: 0x00000105
Processor load watchdog of all IEC-tasks detected

26.1.43 Define: RTSEXCPT_FPU_ERROR

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_ERROR
Key: 0x00000150
FPU: Unspecified error

26.1.44 Define: RTSEXCPT_FPU_DENORMAL_OPERAND

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_DENORMAL_OPERAND
Key: 0x00000151
FPU: Denormal operand

26.1.45 Define: RTSEXCPT_FPU_DIVIDEBYZERO

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_DIVIDEBYZERO
Key: 0x00000152
FPU: Division by zero

26.1.46 Define: RTSEXCPT_FPU_INEXACT_RESULT

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_INEXACT_RESULT
Key: 0x00000153
FPU: Inexact result

26.1.47 Define: RTSEXCPT_FPU_INVALID_OPERATION

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_INVALID_OPERATION
Key: 0x00000154
FPU: Invalid operation

26.1.48 Define: RTSEXCPT_FPU_OVERFLOW

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_OVERFLOW
Key: 0x00000155
FPU: Overflow

26.1.49 Define: RTSEXCPT_FPU_STACK_CHECK

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_STACK_CHECK
Key: 0x00000156
FPU: Stack check

26.1.50 Define: RTSEXCPT_FPU_UNDERFLOW

Category: Exception code, hardware FPU exceptions
Type:
Define: RTSEXCPT_FPU_UNDERFLOW
Key: 0x00000157
FPU: Underflow

26.1.51 Define: RTSEXCPT_VENDOR_EXCEPTION_BASE

Category: Exception code, vendor specific
Type:
Define: RTSEXCPT_VENDOR_EXCEPTION_BASE
Key: 0x00002000
Base number for vendor specific exception codes. VendorID must be specified as prefix inside the exception code: RTS_UI32 ulException = ADDVENDORID([VendorId], RTSEXCPT_VENDOR_EXCEPTION_BASE + [Exception]);

26.1.52 Define: EVT_EXCPT_GenerateException

Category: Events
Type:
Define: EVT_EXCPT_GenerateException
Key: MAKE_EVENTID
The event is sent if an exception occurred (see category exception code for detailed information)

26.1.53 Define: EXCPT_GET_CODE

Category:
Type:
Define: EXCPT_GET_CODE
Key:
Structured exception handling: * rts_try: macro to enter a exception handling section * rts_catch_first: macro to handle the first exception code * rts_catch_next: macro to handle the following exception codes after rts_catch_first * rts_catch_remaining: macro to handle all remaining exception codes after rts_catch_first/rts_catch_next * rts_catch: macro to handle all exception codes without previous usage of rts_catch_first/rts_catch_next/rts_catch_remaining * rts_finally: macro is called always after an exception or not to cleanup some stuff, that was occupied in the rts_try section *

26.1.54 Typedef: ExceptionCode

Stuctname: ExceptionCode
Category: Exception code

1=exception code is an operating system specific exception code

0=exception code is a runtime code. See exception code category

Typedef: typedef struct tagExceptionCode { RTS_UI32 ulCode; int bOSException; } ExceptionCode;

26.1.55 Typedef: RegContext

Stuctname: RegContext
Category: Exception context

Typedef: typedef struct RegContexttag { RTS_UINTPTR IP; RTS_UINTPTR BP; RTS_UINTPTR SP; } RegContext;

26.1.56 Typedef: EVTPARAM_SysExcept

Stuctname: EVTPARAM_SysExcept
Category: Event parameter

Typedef: typedef struct { RTS_HANDLE uiTaskOSHandle; RTS_IEC_DWORD ulException; RegContext Context; RTS_RESULT Result; } EVTPARAM_SysExcept;

26.1.57 Typedef: sysexceptenableseh2_struct

Stuctname: sysexceptenableseh2_struct

sysexceptenableseh2

Typedef: typedef struct tagsysexceptenableseh2_struct { RTS_IEC_VOID_FCTPTR pfExceptionHandler; VAR_INPUT RTS_IEC_UDINT SysExceptEnableSEH2; VAR_OUTPUT } sysexceptenableseh2_struct;

26.1.58 Typedef: sysexceptdisableseh2_struct

Stuctname: sysexceptdisableseh2_struct

sysexceptdisableseh2

Typedef: typedef struct tagsysexceptdisableseh2_struct { RTS_IEC_VOID_FCTPTR pfExceptionHandler; VAR_INPUT RTS_IEC_UDINT SysExceptDisableSEH2; VAR_OUTPUT } sysexceptdisableseh2_struct;

26.1.59 Typedef: sysexceptenableseh_struct

Stuctname: sysexceptenableseh_struct

sysexceptenableseh

Typedef: typedef struct tagsysexceptenableseh_struct { RTS_IEC_UDINT SysExceptEnableSEH; VAR_OUTPUT } sysexceptenableseh_struct;

26.1.60 Typedef: sysexceptdisableseh_struct

Stuctname: sysexceptdisableseh_struct

sysexceptdisableseh

Typedef: typedef struct tagsysexceptdisableseh_struct { RTS_IEC_UDINT SysExceptDisableSEH; VAR_OUTPUT } sysexceptdisableseh_struct;

26.1.61 Typedef: sysexceptgenerateexception_struct

Stuctname: sysexceptgenerateexception_struct

Interrupt the execution of the currently running task, and set the application into an exception state. If the runtime system supports it, CODESYS may even highlight the source position where this function was called.

RESULT: If succeeded this function doesn't return at all, otherwise it returns ERR_FAILED.

Typedef: typedef struct tagsysexceptgenerateexception_struct { RTS_IEC_UDINT udiException; VAR_INPUT RTS_IEC_RESULT SysExceptGenerateException; VAR_OUTPUT } sysexceptgenerateexception_struct;

26.1.62 SysExceptMapException

RTS_UI32 SysExceptMapException (RTS_UI32 ulOSException)

Map the operating system specific exception into the standard runtime exception code

ulOSException [IN]

Operating system specific exception code

Result

Standard exception code (see category exception code above)

26.1.63 SysExceptGenerateException

RTS_RESULT SysExceptGenerateException (RTS_HANDLE ulTaskOSHandle, ExceptionCode Exception, RegContext Context)

Structured Exception Handling

This function is called from Exception handlers of the hardware, operating systems or, in case of softerrors, within the task code.

The structured exception handling calls every registered exception handler, stops the IEC Tasks and prepares the task context of the task that generated the exception, so that it can be read by the CoDeSys programming system for analysation purposes.

OEM customers can use this function also from their own exception handlers to propagate generic as well as customer specific exceptions to the application. This way, they can profit from the debugging infrastructure of CoDeSys to find the fault adress within the IEC application.

Note: On SIL2 Platforms, this function will act only as described above, when the PLC is currently in Debug-Mode. If it is actually in the Safety-Mode, this call puts the PLC directly into the safe mode, by calling the function SIL2OEMException().

ulTaskOSHandle [IN]

System specific task or timer handle or RTS_INVALID_HANDLE if unknown

Exception [IN]

Exception code (OS or runtime specific)

Context [IN]

Context to detect the code location where the exception was generated. If ulTaskOSHandle is RTS_INVALID_HANDLE, values can be 0.

Result

error code

26.1.64 SysExceptGenerateException2

RTS_RESULT SysExceptGenerateException2 (RTS_HANDLE ulTaskOSHandle, ExceptionCode Exception, RegContext Context, int bSuspendExceptionTask)

Structured Exception Handling

This function acts the same way as SysExceptGenerateException(), except for the case, that it will try to suspend the task in the following condition:

- bSuspendExceptionTask is set to TRUE
- None of the registered exception handlers returned ERR_DONT_SUSPEND_TASK

ulTaskOSHandle [IN]
System specific task or timer handle or RTS_INVALID_HANDLE if unknown

Exception [IN]
Exception code (OS or runtime specific)

Context [IN]
Context to detect the code location where the exception was generated. If ulTaskOSHandle is RTS_INVALID_HANDLE, values can be 0.

bSuspendExceptionTask [IN]
Flag to specify, if the exception task is forced to suspended

Result
error code

26.1.65 SysExceptRegisterInterface

RTS_RESULT SysExceptRegisterInterface (PFEXCEPTIONHANDLER pExceptionHandler)

Function to register an exception handler

pExceptionHandler [IN]
Pointer to exception handler

Result
error code

26.1.66 SysExceptUnregisterInterface

RTS_RESULT SysExceptUnregisterInterface (PFEXCEPTIONHANDLER pExceptionHandler)

Function to unregister exception handler

pExceptionHandler [IN]
Pointer to exception handler

Result
error code

26.1.67 SysExceptConvertToString

*RTS_RESULT SysExceptConvertToString (RTS_UI32 ulExceptionCode, char*pszException, int iMaxExceptionLen)*

Convert an exception to string

ulExceptionCode [IN]
Exception code

pszException [OUT]
Pointer to exception string

iMaxExceptionLen [IN]
Maximum length of exception string pointer

Result
error code

26.1.68 SysExceptRegisterJmpBuf

*RTS_RESULT SysExceptRegisterJmpBuf (RTS_JMP_BUF *pJmpBuf, int bEnable)*

Functions for structured exception handling only. Must be imported, if SEH will be used!

26.1.69 SysExceptEnableSEH

RTS_RESULT SysExceptEnableSEH (void)

Enable the exception handling, if rts_try_disabled is used

Result
error code

26.1.70 SysExceptDisableSEH

RTS_RESULT SysExceptDisableSEH (void)

Disablethe exception handling, if rts_try_disabled is used

Result
error code

26.1.71 SysExceptEnableSEH2

RTS_RESULT SysExceptEnableSEH2 (PFEXCEPTIONHANDLER pfExceptionHandler, int bIec)

Enable the exception handling, if rts_try_disabled is used! Additionally the specified exception handler is called at the exception.

pfExceptionHandler [IN]
Function pointer to the exception handler

blec [IN]
Specified, whether the function pointer is an IEC function (bIec=1) or a C function (bIec=0)

Result
error code

26.1.72 SysExceptDisableSEH2

RTS_RESULT SysExceptDisableSEH2 (PFEXCEPTIONHANDLER pfExceptionHandler, int bIec)

Disable the exception handling, if rts_try_disabled is used! The specified exception handler will be removed.

pfExceptionHandler [IN]
Function pointer to the exception handler

blec [IN]
Specified, whether the function pointer is an IEC function (bIec=1) or a C function (bIec=0)

Result
error code

27 SysFile

System component that allows access to file functions.

Compiler Switch

- #define SYSFILE_DISABLE_FILE_CACHE For each transmitted file, the CRC and size is stored in a central config file (cache). This can be disabled by this compiler switch.

27.1 SysFileItf

The SysFile interface is projected to get access to the files of a filesystem. The filesystem can be on harddisk, a flash/flash disk, a RAM disk or what ever. The only requirement is, that the filesystem is non volatile!

Please read following notes if using the SysFileFlash with our SysFlash: This component needs a global define of the file table FILE_MAP. It has to be declared in sysdefines.h. Here is an example with the neccessay initializations: #define FILE1_SIZE 0x4000 #define FILE2_SIZE 0x2000 #define FILE3_SIZE 0x2000 #define FILE4_SIZE 0x38000 #define FILE5_SIZE 0x40000 #define SYSFILEFLASH_MAX_SIZE (FILE1_SIZE + FILE2_SIZE + FILE3_SIZE + FILE4_SIZE + FILE5_SIZE) #define FILE_MAP FILE_DESC m_FileSystem[] = \ { \ * Name Offset MaxSize read index write index * \ {"file1.txt", 0x0, FILE1_SIZE, 0xFFFFFFFF, 0xFFFFFFFF}, \ {"app.crc", FILE1_SIZE, FILE2_SIZE, 0xFFFFFFFF, 0xFFFFFFFF}, \ {"file2.txt", FILE1_SIZE+FILE2_SIZE, FILE3_SIZE, 0xFFFFFFFF, 0xFFFFFFFF}, \ {"file4.txt", FILE1_SIZE+FILE2_SIZE+FILE3_SIZE, FILE4_SIZE, 0xFFFFFFFF, 0xFFFFFFFF}, \ {"app.app", FILE1_SIZE+FILE2_SIZE+FILE3_SIZE+FILE4_SIZE, FILE5_SIZE, 0xFFFFFFFF, 0xFFFFFFFF}, \ }; Please note that the offsets of the files have to correspond with sector borders of the flash. One file should be stored in one sector.

27.1.1 Define: MAX_PATH_LEN

Condition: #ifndef MAX_PATH_LEN
Category: Static defines
Type:
Define: MAX_PATH_LEN
Key: 255
Maximum length of a file path

27.1.2 Define: SYSFILE_TEMP_BUFFER_SIZE

Condition: #ifndef SYSFILE_TEMP_BUFFER_SIZE
Category: Static defines
Type:
Define: SYSFILE_TEMP_BUFFER_SIZE
Key: 256
Size for temporary buffer e.g. to calculate CRC. If size is larger than 256, buffer is allocated from heap!

27.1.3 Define: SYSFILE_MAP_SECTION_NAME

Condition: #ifndef SYSFILE_MAP_SECTION_NAME
Category: Static defines
Type:
Define: SYSFILE_MAP_SECTION_NAME
Key: SysFileMap
This is the name for a section which contains the name, CRC and size of a file. Is used as a cache for these values. Improves for example the performance of the file transfer. SYSFILE_DISABLE_FILE_CACHE disables this feature. NOTE: It is recommended to use a filereference in the configuration, to move these entries into a separate cfg-file. e.g.: [CmpSettings] FileReference.0=SysFileMap.cfg, SysFileMap

27.1.4 Define: SYSFILE_VISU_FOLDER

Condition: #ifndef SYSFILE_VISU_FOLDER
Category: Static defines
Type:
Define: SYSFILE_VISU_FOLDER
Key: visu/

Defines to specify the visu folder that is requested by every filetransfer/fileaccess on visu files. And we specify the placeholder for there files, for which the destination folder can be specified with the setting "PlaceholderFilePath" (see details above).

27.1.5 Define: ACCESS_MODE_AM_READ

Category: Access mode
Type:
Define: ACCESS_MODE_AM_READ
Key: 0

File modes to open a file.

ATTENTION: For all _PLUS modes be aware, that after reading from a file, writing can only be done after a call to SysFileGetPos() or SysFileSetPos()! If you call SysFileWrite right after SysFileRead, the file pointer could be on an invalid position!

Correct example: SysFileRead(); SysFileGetPos(); SysFileWrite();

27.1.6 Define: SYS_FILE_STATUS_FS_OK

Category: File status.
Type:
Define: SYS_FILE_STATUS_FS_OK
Key: 0

Actual file status of the specified file.

27.1.7 Define: RTS_ACCESS_MODE

27.1.8 Typedef: SYS_FILETIME

Stuctname: SYS_FILETIME
Category: File TIME in UTC.

Timestamps of the specified file.

Typedef: typedef struct tagSYS_FILETIME { RTS_IEC_UDINT tCreation; RTS_IEC_UDINT tLastAccess; RTS_IEC_UDINT tLastModification; } SYS_FILETIME;

27.1.9 Typedef: AnyType

Stuctname: AnyType
AnyType
Typedef: typedef struct tagAnyType { RTS_IEC_BYTE *pValue; RTS_IEC_DINT diSize; RTS_IEC_DWORD TypeClass; } AnyType;

27.1.10 Typedef: sysfileclose_struct

Stuctname: sysfileclose_struct
Close a file specified by handle

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfileclose_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_RESULT SysFileClose; VAR_OUTPUT } sysfileclose_struct;

27.1.11 Typedef: sysfilecopy_struct

Stuctname: sysfilecopy_struct
Copy one file to another. A standard path will be added to the filename, if no path is specified.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilecopy_struct { RTS_IEC_STRING *szDestFileName; VAR_INPUT RTS_IEC_STRING *szSourceFileName; VAR_INPUT RTS_IEC_UDINT *pulCopied; VAR_INPUT RTS_IEC_RESULT SysFileCopy; VAR_OUTPUT } sysfilecopy_struct;

27.1.12 Typedef: sysfiledelete_struct

Stuctname: sysfiledelete_struct
Delete the file specified by name. A standard path will be added in the runtime system to the filename, if no path is specified.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfiledelete_struct { RTS_IEC_STRING *szFileName; VAR_INPUT RTS_IEC_RESULT SysFileDelete; VAR_OUTPUT } sysfiledelete_struct;

27.1.13 Typedef: sysfiledeletebyhandle_struct

Stuctname: sysfiledeletebyhandle_struct

Delete the file specified by handle

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfiledeletebyhandle_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_RESULT SysFileDeleteByHandle; VAR_OUTPUT } sysfiledeletebyhandle_struct;

27.1.14 Typedef: sysfileeof_struct

Stuctname: sysfileeof_struct

Check, if end of file is reached

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfileeof_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_RESULT SysFileEOF; VAR_OUTPUT } sysfileeof_struct;

27.1.15 Typedef: sysfilegetname_struct

Stuctname: sysfilegetname_struct

Get the file name from file specified by handle

RESULT: File name of the specified file

Typedef: typedef struct tagsysfilegetname_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_STRING *SysFileGetName; VAR_OUTPUT } sysfilegetname_struct;

27.1.16 Typedef: sysfilegetname2_struct

Stuctname: sysfilegetname2_struct

Get the file name from file specified by handle

RESULT: File name of the specified file

Typedef: typedef struct tagsysfilegetname2_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_STRING *SysFileGetName2; VAR_OUTPUT } sysfilegetname2_struct;

27.1.17 Typedef: sysfilegetpath_struct

Stuctname: sysfilegetpath_struct

Get the path of this file. If a path is specified in the filename, the path will be extracted from the filename. If no path is specified in the filename, the standard path for this file extension type will be returned.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilegetpath_struct { RTS_IEC_STRING *szFileName; VAR_INPUT RTS_IEC_STRING *szPath; VAR_INPUT RTS_IEC_DINT diMaxLen; VAR_INPUT RTS_IEC_RESULT SysFileGetPath; VAR_OUTPUT } sysfilegetpath_struct;

27.1.18 Typedef: sysfilegetpos_struct

Stuctname: sysfilegetpos_struct

Get actual file pointer position

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilegetpos_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_UDINT *pulPos; VAR_INPUT RTS_IEC_RESULT SysFileGetPos; VAR_OUTPUT } sysfilegetpos_struct;

27.1.19 Typedef: sysfilegetsize_struct

Stuctname: sysfilegetsize_struct

Get file size of the file specified by name. A standard path will be added to the filename, if no path is specified.

RESULT: Size of the file in bytes.

Typedef: typedef struct tagsysfilegetsize_struct { RTS_IEC_STRING *szFileName; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_UDINT SysFileGetSize; VAR_OUTPUT } sysfilegetsize_struct;

27.1.20 Typedef: sysfilegetsizebyhandle_struct

Stuctname: sysfilegetsizebyhandle_struct

Get file size of the file specified by handle.

RESULT: Size of the file in bytes.

Typedef: typedef struct tagsysfilegetsizebyhandle_struct { RTS_IEC_UDINT hFile; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_UDINT SysFileGetSizeByHandle; VAR_OUTPUT } sysfilegetsizebyhandle_struct;

27.1.21 Typedef: sysfilegetstatus_struct

Stuctname: sysfilegetstatus_struct

Get the file status

Typedef: typedef struct tagsysfilegetstatus_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_INT SysFileGetStatus; VAR_OUTPUT, Enum: SYS_FILE_STATUS } sysfilegetstatus_struct;

27.1.22 Typedef: sysfilegetstatus2_struct

Stuctname: sysfilegetstatus2_struct

Get the file status

Typedef: typedef struct tagsysfilegetstatus2_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_INT SysFileGetStatus2; VAR_OUTPUT, Enum: SYS_FILE_STATUS } sysfilegetstatus2_struct;

27.1.23 Typedef: sysfilegettime_struct

Stuctname: sysfilegettime_struct

Get file time of the specified file. A standard path will be added to the filename, if no path is specified.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilegettime_struct { RTS_IEC_STRING *szFileName; VAR_INPUT SYS_FILETIME *ptFileTime; VAR_INPUT RTS_IEC_RESULT SysFileGetTime; VAR_OUTPUT } sysfilegettime_struct;

27.1.24 Typedef: sysfileopen_struct

Stuctname: sysfileopen_struct

Open or create file. A standard path will be added to the filename, if no path is specified in the file name.

If a file extension is specified in the settings, this path will be used (see category settings).

IMPLEMENTATION NOTE: File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\"!

RESULT: Handle to the file or RTS_INVALID_HANDLE if failed.

Typedef: typedef struct tagsysfileopen_struct { RTS_IEC_STRING *szFile; VAR_INPUT RTS_IEC_UDINT am; VAR_INPUT, Enum: ACCESS_MODE RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_HANDLE SysFileOpen; VAR_OUTPUT } sysfileopen_struct;

27.1.25 Typedef: sysfileread_struct

Stuctname: sysfileread_struct

Read number of bytes out of the file

RESULT: Number of bytes read from file. 0=if failed.

Typedef: typedef struct tagsysfileread_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_UDINT ulSize; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_UDINT SysFileRead; VAR_OUTPUT } sysfileread_struct;

27.1.26 Typedef: sysfilerename_struct

Stuctname: sysfilerename_struct

Rename the file. A standard path will be added to the filename, if no path is specified.

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilerename_struct { RTS_IEC_STRING *szOldFileName; VAR_INPUT RTS_IEC_STRING *szNewFileName; VAR_INPUT RTS_IEC_RESULT SysFileRename; VAR_OUTPUT } sysfilerename_struct;

27.1.27 Typedef: sysfilesetpos_struct

Stuctname: sysfilesetpos_struct

Set the file pointer to the specified position

RESULT: Returns the runtime system error code (see CmpErrors.library).

Typedef: typedef struct tagsysfilesetpos_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_UDINT ulOffset; VAR_INPUT RTS_IEC_RESULT SysFileSetPos; VAR_OUTPUT } sysfilesetpos_struct;

27.1.28 Typedef: sysfilewrite_struct

Stuctname: sysfilewrite_struct

Write number of bytes to the file. File must be opened with AM_WRITE or AM_APPEND.

RESULT: Number of bytes written to the file. 0=if failed.

Typedef: typedef struct tagsysfilewrite_struct { RTS_IEC_HANDLE hFile; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_UDINT ulSize; VAR_INPUT RTS_IEC_RESULT *pResult; VAR_INPUT RTS_IEC_UDINT SysFileWrite; VAR_OUTPUT } sysfilewrite_struct;

27.1.29 SysFileOpen

*RTS_HANDLE SysFileOpen (char *pszFile, RTS_ACCESS_MODE am, RTS_RESULT *pResult)*

Open or create file. A standard path will be added to the filename, if no path is specified in the file name.

If a file extension is specified in the settings, this path will be used (see category settings).

IMPLEMENTATION NOTE: File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\"!

pszFile [IN]

File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\"!

am [IN]

Requested access mode to the file:

- AM_READ: If file does not exist, an error is returned. If the file exists, the file will be opened
- AM_WRITE: If file does not exist, a new file will be created. If the file exists, it will be overwritten!
- AM_APPEND: If the file does not exist, an error is returned. If the file exists, the file will be opened

pResult [OUT]

Pointer to error code

Result

Handle to the file or RTS_INVALID_HANDLE if failed

27.1.30 SysFileGetStatus

*SYS_FILE_STATUS SysFileGetStatus (RTS_HANDLE hFile, RTS_RESULT *pResult)*

Get the file status

hFile [IN]

Handle to the file

pResult [IN]
Pointer to error code
Result
File status. See category file status

27.1.31 SysFileGetName

char SysFileGetName (RTS_HANDLE hFile, RTS_RESULT *pResult)*
Get the file name from file specified by handle
hFile [IN]
Handle to the file
pResult [IN]
Pointer to error code
Result
File name

27.1.32 SysFileRead

*RTS_SIZE SysFileRead (RTS_HANDLE hFile, unsigned char *pbyBuffer, RTS_SIZE uiSize, RTS_RESULT *pResult)*
Read number of bytes out of the file
hFile [IN]
Handle to the file
pbyBuffer [OUT]
Pointer to buffer for read data
uiSize [IN]
Number of bytes to read from file. Must be less or equal the buffer size!
pResult [OUT]
Pointer to error code
Result
Number of bytes read from file. 0=if failed

27.1.33 SysFileWrite

*RTS_SIZE SysFileWrite (RTS_HANDLE hFile, unsigned char *pbyBuffer, RTS_SIZE uiSize, RTS_RESULT *pResult)*
Write number of bytes to the file. File must be opened with AM_WRITE or AM_APPEND.
hFile [IN]
Handle to the file
pbyBuffer [IN]
Pointer to buffer with data to write to file
uiSize [IN]
Number of bytes to write in the file. Must be less or equal the buffer size!
pResult [OUT]
Pointer to error code
Result
Number of bytes written to the file. 0=if failed

27.1.34 SysFileDeleteByHandle

RTS_RESULT SysFileDeleteByHandle (RTS_HANDLE hFile)
Delete the file specified by handle
hFile [IN]
Handle to the file
Result
error code

27.1.35 SysFileSetPos

RTS_RESULT SysFileSetPos (RTS_HANDLE hFile, RTS_SIZE uiOffset)
Set the file pointer to the specified position
hFile [IN]
Handle to the file

uiOffset [IN]
Offset to set from the beginning of the file
Result
error code

27.1.36 SysFileGetPos

*RTS_RESULT SysFileGetPos (RTS_HANDLE hFile, RTS_SIZE *puiPos)*
Get actual file pointer position
hFile [IN]
Handle to the file
puiPos [OUT]
Pointer to get actual position of the file pointer from the beginning of the file
Result
error code

27.1.37 SysFileGetSizeByHandle

*RTS_SIZE SysFileGetSizeByHandle (RTS_HANDLE hFile, RTS_RESULT *pResult)*
Get file size of the actual opened file
hFile [IN]
Handle to the file
pResult [OUT]
Pointer to error code
Result
Size of the file in bytes

27.1.38 SysFileEOF

RTS_RESULT SysFileEOF (RTS_HANDLE hFile)
Check, if end of file is reached at reading from the file. IMPLEMENTATION NOTE: End of file is only checked after a read operation with SysFileRead! But after a SysFileWrite or SysFileSetPos call, the function returns ERR_FAILED (no end of file)!
hFile [IN]
Handle to the file
Result
Error code:
- ERR_OK: End of file reached at reading beyond the end of the file
- ERR_FAILED: No end of file reached
- ERR_PARAMETER: hFile is invalid

27.1.39 SysFileFlush

RTS_RESULT SysFileFlush (RTS_HANDLE hFile)
Flush the file cache and write into the file.
hFile [IN]
Handle to the file
Result
Error code:
- ERR_OK: Succeeded flushing the file
- ERR_FAILED: Error occurred during file flush
- ERR_NOTIMPLEMENTED: File flush is not implemented
- ERR_NOT_SUPPORTED: File flush not available on the target

27.1.40 SysFileOpen_

*RTS_HANDLE SysFileOpen_ (char *pszFile, RTS_ACCESS_MODE am, RTS_RESULT *pResult)*
Open or create a file. File will be opened with no standard path. The file name will be used as it is.
pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\"!
am [IN]

Requested access mode to the file:

- AM_READ: If file does not exist, an error is returned. If the file exists, the file will be opened
- AM_WRITE: If file does not exist, a new file will be created. If the file exists, it will be overwritten!
- AM_APPEND: If the file does not exist, an error is returned. If the file exists, the file will be opened

pResult [OUT]
Pointer to error code

Result
Handle to the file or RTS_INVALID_HANDLE if failed

27.1.41 SysFileDelete

*RTS_RESULT SysFileDelete (char *pszFile)*

Delete the file specified by name. A standard path will be added to the filename, if no path is specified.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

Result
error code

27.1.42 SysFileDelete_

*RTS_RESULT SysFileDelete_ (char *pszFile)*

Delete the file specified by name. Filename will be used with no standard path.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

Result
error code

27.1.43 SysFileRename

*RTS_RESULT SysFileRename (char *pszOldFileName, char *pszNewFileName)*

Rename the file. A standard path will be added to the filename, if no path is specified.

pszOldFileName [IN]
Old file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pszNewFileName [IN]
New file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

Result
error code

27.1.44 SysFileRename_

*RTS_RESULT SysFileRename_ (char *pszOldFileName, char *pszNewFileName)*

Rename the file. File will be renamed with no standard path.

pszOldFileName [IN]
Old file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pszNewFileName [IN]
New file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

Result
error code

27.1.45 SysFileGetSize

*RTS_SIZE SysFileGetSize (char *pszFile, RTS_RESULT *pResult)*

Get file size of the file specified by name A standard path will be added to the filename, if no path is specified.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pResult [OUT]
Pointer to error code: ERR_OK: Successful ERR_NO_OBJECT: File not available ERR_FAILED: Failed to get file size

Result
Size of the file in bytes

27.1.46 SysFileGetSize_

*RTS_SIZE SysFileGetSize_ (char *pszFile, RTS_RESULT *pResult)*

Get file size of the file specified by name, No standard path will be added to the file name.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pResult [OUT]
Pointer to error code

Result
Size of the file in bytes

27.1.47 SysFileGetTime

*RTS_RESULT SysFileGetTime (char *pszFile, SYS_FILETIME *pftFileTime)*

Get file time of the specified file. A standard path will be added to the filename, if no path is specified.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pftFileTime [OUT]
Pointer to get the file time results. IMPLEMENTATION NOTE: All time values must be local time!

Result
error code

27.1.48 SysFileGetTime_

*RTS_RESULT SysFileGetTime_ (char *pszFile, SYS_FILETIME *pftFileTime)*

Get file time of the file specified by name. No standard path will be added.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pftFileTime [OUT]
Pointer to get the file time results. IMPLEMENTATION NOTE: All time values must be local time!

Result
error code

27.1.49 SysFileCopy

*RTS_RESULT SysFileCopy (char *pszDestFileName, char *pszSourceFileName, RTS_SIZE *puiCopied)*

Copy one file to another. A standard path will be added to the filename, if no path is specified. IMPLEMENTATION NOTE: If the destination file exists, the file will be overwritten and the function succeeded.

pszDestFileName [IN]
Destination file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

pszSourceFileName [IN]
Source file name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\'!

puiCopied [OUT]
Number of bytes copied

Result
error code

27.1.50 SysFileCopy_

*RTS_RESULT SysFileCopy_ (char *pszDestFileName, char *pszSourceFileName, RTS_SIZE *puiCopied)*

Copy one file in another. No standard path will be added to filename.

pszFile [IN]
File name. File name can contain an absolute or relative path to the file. Path entries must be separated with a '/' and not with a '\"!

pftFileTime [OUT]
Pointer to get the file time results

Result
error code

27.1.51 SysFileGetPath

*RTS_RESULT SysFileGetPath (char *pszFileName, char *pszPath, RTS_SIZE iMaxLen)*

Get the path of this file. If a path is specified in the filename, the path will be extracted from the filename. If no path is specified in the filename, the standard path for this file extension type will be returned.

pszFileName [IN]
File name. Can contain an absolute or relative path

pszPath [OUT]
Path for this file

iMaxLen [IN]
Maximum size in bytes of path length

Result
error code

27.1.52 SysFileGetIecPath

*RTS_RESULT SysFileGetIecPath (char *pszFileName, char *pszPath, RTS_SIZE iMaxLen)*

Get the path of this file for Iec applications. If a path is specified in the filename, the path will be extracted from the filename. If no path is specified in the filename, the standard path for this file extension type will be returned.

pszFileName [IN]
File name. Can contain an absolute or relative path

pszPath [OUT]
Path for this file

iMaxLen [IN]
Maximum size in bytes of path length

Result
error code

27.1.53 SysFileGenerateCRC

*RTS_RESULT SysFileGenerateCRC (char *pszFile, RTS_SIZE ulSize, RTS_UI32 *pulCRC)*

Generate the CRC32 of a file. Can be used to check file integrity. ATTENTION: Only for backward compatibility! CRC is implemented not independant from buffer size!

pszFile [IN]
File name. Can contain an absolute or relative path

ulSize [IN]
Size to calculate checksum. 0 if real size of file should be used [default]

pulCRC [OUT]
Pointer to get CRC32 result

Result
error code

27.1.54 SysFileGenerateCRC2

*RTS_RESULT SysFileGenerateCRC2 (char *pszFile, RTS_SIZE ulSize, RTS_UI32 *pulCRC)*

Generate the CRC32 of a file. Can be used to check file integrity.

pszFile [IN]
File name. Can contain an absolute or relative path

ulSize [IN]
Size to calculate checksum. 0 if real size of file should be used [default]

pulCRC [OUT]
Pointer to get CRC32 result

Result
error code

27.1.55 SysFileGenerateCRC2_

*RTS_RESULT SysFileGenerateCRC2_ (char *pszFile, RTS_SIZE ulSize, RTS_UI32 *pulCRC)*

Generate the CRC32 of a file. Can be used to check file integrity. IMPLEMENTATION NOTE: This interface function is implemented operating system dependant! Optimizations can be done here.

pszFile [IN]
File name. Can contain an absolute or relative path

ulSize [IN]
Size to calculate checksum. 0 if real size of file should be used [default]

pulCRC [OUT]
Pointer to get CRC32 result

Result
error code

27.1.56 SysFileClose

RTS_RESULT SysFileClose (RTS_HANDLE hFile)

Close a file specified by handle

hFile [IN]
Handle to the file

Result
error code

27.2 SysDirItf

The SysDir interface is projected to handle all system dependant directory operations. If there is no filesystem on the target, the interface functions ERR_NOTIMPLEMENTED.

27.2.1 Define: DF_FILE

27.2.2 Typedef: DirFileTime

Stuctname: DirFileTime
Category: Directory file time
Typedef: typedef struct { RTS_UI32 ulCreation; RTS_UI32 ulLastAccess; RTS_UI32 ulLastModification; } DirFileTime;

27.2.3 Typedef: DirInfo

Stuctname: DirInfo
Category: Directory info entry
This structure contains the description of one directory entry
Typedef: typedef struct { DirFileTime dft; RTS_UI32 ulSize; RTS_UI32 ulFlags; } DirInfo;

27.2.4 SysDirOpen

*RTS_HANDLE SysDirOpen (char *pszDir, char *pszDirEntry, int iMaxDirEntry, DirInfo *pDirInfo, RTS_RESULT *pResult)*

Opens a specified directory and returns a handle and the first directory entry

pszDir [IN]
Name of directory. Wildcards can be used for file types: "Sun*" or "Sun?hine" or "*.txt"
IMPLEMENTATION NOTE: Empty string ("") is the request for the current working directory.

pszDirEntry [OUT]
Directory entry as string

iMaxDirEntry [IN]
Max number of bytes to write in pszDirEntry

pDirInfo [OUT]
Directory information

pResult [OUT]
Pointer to error code

Result

RTS_HANDLE on directory

27.2.5 SysDirRead

*RTS_RESULT SysDirRead (RTS_HANDLE hDir, char *pszDirEntry, int iMaxDirEntry, DirInfo *pDirInfo)*

Read next directory entry. Writes the entry in pszDirEntry.

hDir [IN]
Handle to directory opened with SysDirOpen

pszDirEntry [OUT]
Directory entry as string

iMaxDirEntry [OUT]
Max number of bytes to write in pszDirEntry

pDirInfo [OUT]
Pointer to directory information. IMPLEMENTATION NOTE: Can be NULL (so only directory name is provided in pszDirEntry)

Result

- ERR_OK: Entry could be read
- ERR_END_OF_OBJECT: If end of directory list was reached
- ERR_PARAMETER: If one of the parameters is invalid
- ERR_BUFFERSIZE: If iMaxDirEntry is too short to get the complete directory entry string ATTENTION: Typically after this error, the dir-handle is set to the next entry, so this entry will be missed!

27.2.6 SysDirCreate

*RTS_RESULT SysDirCreate (char *pszDir)*

Creates a new directory with the specified name

pszDir [IN]
Name of directory

Result
error code

27.2.7 SysDirDelete

*RTS_RESULT SysDirDelete (char *pszDir)*

Deletes a directory with the specified name

pszDir [IN]
Name of directory

Result
error code

27.2.8 SysDirRename

*RTS_RESULT SysDirRename (char *pszOldDir, char *pszNewDir)*

Rename directory

pszOldDir [IN]
Name of existing directory

pszNewDir [IN]
New name

Result
error code

27.2.9 SysDirGetCurrent

*RTS_RESULT SysDirGetCurrent (char *pszDir, int iMaxDirLen)*

Get current working directory

pszDir [OUT]
Name of current directory

iMaxDirLen [IN]
Max lenght of directory buffer

Result
error code

27.2.10 SysDirSetCurrent

*RTS_RESULT SysDirSetCurrent (char *pszDir)*

Set current working directory

pszDir [IN]

Name of current directory

Result

error code

27.2.11 SysDirClose

RTS_RESULT SysDirClose (RTS_HANDLE hDir)

Close an open directory

hDir [IN]

Handle to directory opened with SysDirOpen

Result

error code

28 SysFlash

System component for flash access.

28.1 SysFlashItf

The SysFlash interface is projected to get access to flash memory of a controller. It has to be adpated to your flash.

There are functions to read and write to flash memory. It is used by some implementations of the file component (SysFileFlash) to store some files in flash, and by the application component for execution of user code in flash. The SysFlash Component divides two different kinds of flash areas: FA_CODE and FA_FILE. FA_FILE is only needed, if SysFileFlash is used. Please see further description for SysFileFlash and our Flash-Filesystem. Please note that the offsets of the files have to correspond with sector borders of the flash. One file should be stored in one sector.

28.1.1 SysFlashInit

RTS_RESULT SysFlashInit (FlashArea fa)

Init the flash system

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

Result
error code

28.1.2 SysFlashErase

RTS_RESULT SysFlashErase (FlashArea fa, RTS_SIZE ulSize, RTS_SIZE ulOffset)

Erases a block of flash memory. You must erase a flash area before writing to it. This function is implemented in the generic part of SysFlash and splits up the block to erase in several smaller pieces with the maximum size of EraseBlockSize. For each piece SysFlashErase_() and the CommCycleHook is called, to keep the rest of the RTS also on single-tasking systems alive.

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

ulSize [IN]
Size of flash area to erase

ulOffset [IN]
Offset of flash area to erase. The function adds the start address of the flash to calculate the physical address of the area.

Result
error code

28.1.3 SysFlashRead

*RTS_RESULT SysFlashRead (FlashArea fa, char *pcDest, RTS_SIZE ulSize, RTS_SIZE ulOffset)*

Reads a block of memory from the flash. This function is implemented in the generic part of SysFlash and splits up the block to read in several smaller pieces with the maximum size of ReadBlockSize. For each piece SysFlashRead_() and the CommCycleHook is called, to keep the rest of the RTS also on single-tasking systems alive.

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

pcDest [IN]
Pointer to buffer that receives the data

ulSize [IN]
Size of the buffer

ulOffset [IN]
Offset of flash area to read from. The function adds the start address of the flash to calculate the physical address of the area.

Result
error code

28.1.4 SysFlashWrite

*RTS_RESULT SysFlashWrite (FlashArea fa, char *pcSource, RTS_SIZE ulSize, RTS_SIZE ulOffset)*

Writes a block of data to the flash. The flash area has to be erased first with SysFlashErase. This function is implemented in the generic part of SysFlash and splits up the block to write in several smaller pieces with the maximum size of WriteBlockSize. For each piece SysFlashWrite() and the CommCycleHook is called, to keep the rest of the RTS also on single-tasking systems alive.

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

pcSource [IN]
Pointer to buffer that contains the data

ulSize [IN]
Size of the buffer

ulOffset [IN]
Offset of flash area to write to. The function adds the start address of the flash to calculate the physical address of the area.

Result
error code

28.1.5 SysFlashFlush

RTS_RESULT SysFlashFlush (FlashArea fa, RTS_SIZE ulSize, RTS_SIZE ulOffset)

Called when a file that was opened for writing is closed. This function is implemented in the generic part of SysFlash and splits up the block to flush in several smaller pieces with the maximum size of FlushBlockSize. For each piece SysFlashFlush() and the CommCycleHook is called, to keep the rest of the RTS also on single-tasking systems alive.

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

ulSize [IN]
Size of the buffer

ulOffset [IN]
Offset of flash area to write to. The function adds the start address of the flash to calculate the physical address of the area.

Result
error code

28.1.6 SysFlashGetPhysicalAddress

*RTS_UINTPTR SysFlashGetPhysicalAddress (FlashArea fa, RTS_RESULT *pResult)*

Retrieve the physical address of a flash area

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

pResult [OUT]
Pointer to error code

Result
Physical address of flash area

28.1.7 SysFlashGetSize

*RTS_SIZE SysFlashGetSize (FlashArea fa, RTS_RESULT *pResult)*

Retrieve the size of a flash area

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

pResult [OUT]
Pointer to error code

Result
Size of flash area

28.1.8 SysFlashErase_

RTS_RESULT SysFlashErase_ (FlashArea fa, RTS_SIZE ulSize, RTS_SIZE ulOffset)

Erases a block of flash memory. You must erase a flash area before writing to it. The content of the flash area should be erased with 0 or ff. Must be implemented in the OS-specific part of SysFlash. Should only be called by SysFlashErase().

fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.

ulSize [IN]
Size of flash area to erase
ulOffset [IN]
Offset of flash area to erase. The function adds the start address of the flash to calculate the physical address of the area.
Result
error code

28.1.9 SysFlashRead_

*RTS_RESULT SysFlashRead_ (FlashArea fa, char *pcDest, RTS_SIZE ulSize, RTS_SIZE ulOffset)*
Reads a block of memory from the flash. Must be implemented in the OS-specific part of SysFlash. Should only be called by SysFlashRead().
fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.
pcDest [IN]
Pointer to buffer that receives the data
ulSize [IN]
Size of the buffer
ulOffset [IN]
Offset of flash area to read from. The function adds the start address of the flash to calculate the physical address of the area.
Result
error code

28.1.10 SysFlashWrite_

*RTS_RESULT SysFlashWrite_ (FlashArea fa, char *pcSource, RTS_SIZE ulSize, RTS_SIZE ulOffset)*
Writes a block of data to the flash. The flash area has to be erased first with SysFlashErase. Must be implemented in the OS-specific part of SysFlash. Should only be called by SysFlashWrite().
fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.
pcSource [IN]
Pointer to buffer that contains the data
ulSize [IN]
Size of the buffer
ulOffset [IN]
Offset of flash area to write to. The function adds the start address of the flash to calculate the physical address of the area.
Result
error code

28.1.11 SysFlashFlush_

RTS_RESULT SysFlashFlush_ (FlashArea fa, RTS_SIZE ulSize, RTS_SIZE ulOffset)
Called when a file that was opened for writing is closed. Normally, this function can be left empty. It can be useful, if the data is not written to flash directly, but buffered in RAM. The call of this function indicates that the file is closed and the data has to be written to flash. Must be implemented in the OS-specific part of SysFlash. Should only be called by SysFlashFlush().
fa [IN]
Flash area that shall be used for the operation. Now, FA_CODE and FA_FILE are defined.
ulSize [IN]
Size of the buffer
ulOffset [IN]
Offset of flash area to write to. The function adds the start address of the flash to calculate the physical address of the area.
Result
error code

29 SysInt

Component for interrupt handling

29.1 SysIntItf

The SysInt interface is projected to get access to the hardware interrupts of the system.

29.1.1 Define: MAX_INT

Condition: #ifndef MAX_INT
Category: Static defines
Type:
Define: MAX_INT
Key: 2
Maximum number of interrupts

29.1.2 Define: MAX_INT_CALLBACKS

Condition: #ifndef MAX_INT_CALLBACKS
Category: Static defines
Type:
Define: MAX_INT_CALLBACKS
Key: 1
Maximum number of callback function for each interrupt

29.1.3 Define: SYS_INT_NONE

Category: Interrupt handler type
Type:
Define: SYS_INT_NONE
Key: 0
Specification, which type of routine the handler is

29.1.4 Define: EVT_INTERRUPT_0

Category: Events
Type:
Define: EVT_INTERRUPT_0
Key: MAKE_EVENTID
Hardware interrupt events

29.1.5 Define: BT_Internal

Category: Bus types
Type:
Define: BT_Internal
Key: 0
Architecture specific bus types

29.1.6 Define: IM_LevelSensitive

Category: Interrupt modes
Type:
Define: IM_LevelSensitive
Key: 0

29.1.7 Typedef: EVTPARAM_SysInt

Stuctname: EVTPARAM_SysInt
Category: Event parameter
Typedef: typedef struct { unsigned long ulInterrupt; } EVTPARAM_SysInt;

29.1.8 Typedef: SYS_INT_CALLBACK_INFO

Stuctname: SYS_INT_CALLBACK_INFO
Sys Int Callback Info Struct
Typedef: typedef struct { SYS_INT_INTHANDLER pCallback; unsigned long ulType; RTS_UINTPTR ulAdditionalInfo; }SYS_INT_CALLBACK_INFO;

29.1.9 Typedef: SYS_INT_INFO

Stuctname: SYS_INT_INFO

Sys Int Info Struct

Typedef: typedef struct { char* pszName; void* pSysData; RTS_HANDLE hCallbackPool; unsigned long ulInterrupt; }SYS_INT_INFO;

29.1.10 Typedef: PciInterrupt

Stuctname: PciInterrupt

PciInterrupt Struct

Typedef: typedef struct { RTS_IEC_UDINT ulBusNumber; RTS_IEC_UDINT ulDevciceNumber; RTS_IEC_UDINT ulFunctionNumber; RTS_IEC_UDINT ulIntLine; } PciInterrupt;

29.1.11 Typedef: IsaInterrupt

Stuctname: IsaInterrupt

Isa Interrupt Struct

Typedef: typedef struct { RTS_IEC_UDINT ulBusNumber; RTS_IEC_UDINT ulDevciceNumber; RTS_IEC_UDINT ulFunctionNumber; RTS_IEC_UDINT ulIntLine; } IsaInterrupt;

29.1.12 Typedef: SYS_INT_DESCRIPTION

Stuctname: SYS_INT_DESCRIPTION

Category: Interrupt description

Optional description for an interrupt

Typedef: typedef struct { RTS_IEC_UDINT BusType; RTS_IEC_UDINT InterruptMode; BusSpecific busSpecific; } SYS_INT_DESCRIPTION;

29.1.13 Typedef: iecinthandlerinstance_struct

Stuctname: iecinthandlerinstance_struct

Iec Int Handler Instance Struct

Typedef: typedef struct { void* __VFTABLEPOINTER; RTS_IEC_DWORD Itf; } iecinthandlerinstance_struct;

29.1.14 Typedef: sysintregisterinstance_struct

Stuctname: sysintregisterinstance_struct

SysIntRegisterInstance Struct

Typedef: typedef struct { RTS_IEC_HANDLE hInt; RTS_IEC_DWORD dwClassId; RTS_IEC_DWORD dwItfId; iecinthandlerinstance_struct *pInterface; RTS_IEC_VOIDPTR ulAdditonalInfo; RTS_IEC_UDINT Result; } sysintregisterinstance_struct;

29.1.15 Typedef: sysintunregisterinstance_struct

Stuctname: sysintunregisterinstance_struct

SysIntUnRegisterInstance Struct

Typedef: typedef struct { RTS_IEC_HANDLE hInt; iecinthandlerinstance_struct *pInterface; RTS_IEC_UDINT Result; } sysintunregisterinstance_struct;

29.1.16 Typedef: sysintopen_struct

Stuctname: sysintopen_struct

SysIntOpen Struct

Typedef: typedef struct { RTS_IEC_DWORD ulInterrupt; SYS_INT_DESCRIPTION *pIntDescription; RTS_IEC_UDINT *pResult; RTS_IEC_HANDLE out; }sysintopen_struct;

29.1.17 Typedef: sysintopenbyname_struct

Stuctname: sysintopenbyname_struct

SysIntOpenByName Struct

Typedef: typedef struct { RTS_IEC_STRING *pszIntName; RTS_IEC_UDINT *pResult; RTS_IEC_HANDLE out; }sysintopenbyname_struct;

29.1.18 Typedef: sysintclose_struct

Stuctname: sysintclose_struct

SysIntClose Struct

Typedef: typedef struct { RTS_IEC_HANDLE hInt; RTS_IEC_UDINT out; }sysintclose_struct;

29.1.19 Typedef: sysintenable_struct

Stuctname: sysintenable_struct

SysIntEnable Struct
Typedef: typedef struct { RTS_IEC_HANDLE hInt; RTS_IEC_UDINT out; }sysintenable_struct;

29.1.20 Typedef: sysintdisable_struct

Stuctname: sysintdisable_struct
SysIntDisable Struct
Typedef: typedef struct { RTS_IEC_HANDLE hInt; RTS_IEC_UDINT out; }sysintdisable_struct;

29.1.21 Typedef: sysintregister_struct

Stuctname: sysintregister_struct
SysIntRegister Struct
Typedef: typedef struct { RTS_IEC_HANDLE hInt; SYS_INT_INTHANDLER pHandler; RTS_IEC_VOIDPTR ulAdditionalInfo; RTS_IEC_UDINT out; }sysintregister_struct;

29.1.22 Typedef: sysintunregister_struct

Stuctname: sysintunregister_struct
SysIntUnregister Struct
Typedef: typedef struct { RTS_IEC_HANDLE hInt; SYS_INT_INTHANDLER pHandler; RTS_RESULT out; }sysintunregister_struct;

29.1.23 Typedef: sysintenableall_struct

Stuctname: sysintenableall_struct
SysIntEnableAll Struct
Typedef: typedef struct { RTS_IEC_UDINT *pulParam; RTS_IEC_UDINT out; }sysintenableall_struct;

29.1.24 Typedef: sysintdisableall_struct

Stuctname: sysintdisableall_struct
SysIntDisableAll Struct
Typedef: typedef struct { RTS_IEC_UDINT *pulParam; RTS_IEC_UDINT out; }sysintdisableall_struct;

29.1.25 Typedef: sysintlevel_struct

Stuctname: sysintlevel_struct
SysIntLevel Struct
Typedef: typedef struct { RTS_IEC_UDINT out; }sysintlevel_struct;

29.1.26 sysintregisterinstance

*void sysintregisterinstance (sysintregisterinstance_struct *p)*
Obsolete Functions: To be removed!

29.1.27 SysIntOpen

*RTS_HANDLE SysIntOpen (unsigned long ulInterrupt, SYS_INT_DESCRIPTION *pIntDescription, RTS_RESULT *pResult)*
Open a valid interrupt

ulInterrupt [IN]
Interrupt number

pIntDescription [IN]
Pointer to optional interrupt description

pResult [OUT]
Pointer to result

Result
Handle to interrupt, is RTS_INVALID_HANDLE if error occured, see Errorcodes.

29.1.28 SysIntOpenByName

RTS_HANDLE SysIntOpenByName (char pszIntName, RTS_RESULT* pResult)*
Open a valid interrupt specified by name

pszIntName [IN]
Interrupt name defined in the settings component

pResult [OUT]
Pointer to result
Result
Handle to interrupt

29.1.29 SysIntEnable

RTS_RESULT SysIntEnable (RTS_HANDLE hInt)
Function to enable an interrupt
hInt [IN]
Handle to the interrupt
Result
error code

29.1.30 SysIntDisable

RTS_RESULT SysIntDisable (RTS_HANDLE hInt)
Function to disable an interrupt
hInt [IN]
Handle to the interrupt
Result
error code

29.1.31 SysIntRegister

RTS_RESULT SysIntRegister (RTS_HANDLE hInt, unsigned long ulType, SYS_INT_INTHANDLER pHandler, RTS_UINTPTR ulAdditionalInfo)
Function to register an interrupt handler
hInt [IN]
Handle to the interrupt
ulType [IN]
Type of interrupt handler. See category Interrupt handler type
pHandler [IN]
Interrupt handler
ulAdditionalInfo [IN]
Info value that is transmitted to the interrupt handler
Result
error code

29.1.32 SysIntUnregister

RTS_RESULT SysIntUnregister (RTS_HANDLE hInt, SYS_INT_INTHANDLER pHandler)
Function to unregister an interrupt handler
hInt [IN]
Handle to the interrupt
pHandler [IN]
Interrupt handler
Result
error code

29.1.33 SysIntEnableAll

*RTS_RESULT SysIntEnableAll (RTS_UI32 *pulParam)*
Function to enable all interrupts
pulParam [IN]
Parameter for enabling all interrupts
Result
error code

29.1.34 SysIntDisableAll

*RTS_RESULT SysIntDisableAll (RTS_UI32 *pulParam)*
Function to disable all interrupts

pulParam [INOUT]
Parameter for disabling all interrupts
Result
error code

29.1.35 SysIntLevel

RTS_RESULT SysIntLevel (void)
Function to check, if we are running actual in an interrupt context
Result
Returns ERR_OK if running in an interrupt handler, ERR_FAILED if not

29.1.36 SysIntClose

RTS_RESULT SysIntClose (RTS_HANDLE hInt)
Close an interrupt
hInt [IN]
Interrupt handle
Result
error code

30 SysInternalLib

Implements functions used by the compiler like conversion routines, LREAL arithmetic etc.

30.1 SysInternalLibItf

The SysInternalLib interface is projected to implement all platform dependant routines for:

- Standard library routines
- Datatype conversion

This routines are used by CoDeSys for the Iec standard library implementation.

If the define SYSINTERNALLIB_DISABLE_INT_DIVBYZERO_CHECK is set, you can disable checking all int divisions on a zero divisor.

If the define SYSINTERNALLIB_DISABLE_REAL_DIVBYZERO_CHECK is set, you can disable checking all real divisions on a zero divisor.

30.1.1 Typedef: get_time_struct

Stuctname: get_time_struct
Category: External IEC interface

Struct for a time output as 32-bit value.

Typedef: typedef struct { RTS_IEC_DWORD out; }get_time_struct;

30.1.2 Typedef: get_ltime_struct

Stuctname: get_ltime_struct
Category: External IEC interface

Struct for a time output as 64-bit value.

Typedef: typedef struct { RTS_IEC_LTIME out; }get_ltime_struct;

30.1.3 Typedef: __memset_struct

Stuctname: __memset_struct
Category: External IEC interface

Struct for setting memory.

Typedef: typedef struct { RTS_IEC_BYTE *pDest; RTS_IEC_DINT iValue; RTS_IEC_DINT iCount; RTS_IEC_BYTE *__MemSet; } __memset_struct;

30.1.4 Typedef: real_cmp_struct

Stuctname: real_cmp_struct
Category: External IEC interface

Struct to compare two 32-bit real values.

Typedef: typedef struct { RTS_IEC_REAL in1; RTS_IEC_REAL in2; RTS_IEC_SINT out; }real_cmp_struct;

30.1.5 Typedef: real_3op_struct

Stuctname: real_3op_struct
Category: External IEC interface

Struct for operation with three 32-bit real values.

Typedef: typedef struct { RTS_IEC_REAL in1; RTS_IEC_REAL in2; RTS_IEC_REAL in3; RTS_IEC_REAL out; }real_3op_struct;

30.1.6 Typedef: real_2op_struct

Stuctname: real_2op_struct
Category: External IEC interface

Struct for operation with two 32-bit real values.

Typedef: typedef struct { RTS_IEC_REAL in1; RTS_IEC_REAL in2; RTS_IEC_REAL out; }real_2op_struct;

30.1.7 Typedef: real_1op_struct

Stuctname: real_1op_struct
Category: External IEC interface

Struct for operation with one 32-bit real value.

Typedef: typedef struct { RTS_IEC_REAL in; RTS_IEC_REAL out; }real_1op_struct;

30.1.8 Typedef: lreal_cmp_struct

Stuctname: lreal_cmp_struct
Category: External IEC interface
Struct to compare two 64-bit real values.
Typedef: typedef struct { RTS_IEC_LREAL in1; RTS_IEC_LREAL in2; RTS_IEC_SINT out; }lreal_cmp_struct;

30.1.9 Typedef: lreal_3op_struct

Stuctname: lreal_3op_struct
Category: External IEC interface
Struct for operation with three 64-bit real values.
Typedef: typedef struct { RTS_IEC_LREAL in1; RTS_IEC_LREAL in2; RTS_IEC_LREAL in3; RTS_IEC_LREAL out; }lreal_3op_struct;

30.1.10 Typedef: real_trunc_struct

Stuctname: real_trunc_struct
Category: External IEC interface
Struct for conversion of a 32-bit real value to a 32-bit integer value.
Typedef: typedef struct { RTS_IEC_REAL in; RTS_IEC_DINT out; }real_trunc_struct;

30.1.11 Typedef: lreal_trunc_struct

Stuctname: lreal_trunc_struct
Category: External IEC interface
Struct for conversion of a 64-bit real value to a 32-bit integer value.
Typedef: typedef struct { RTS_IEC_LREAL in; RTS_IEC_DINT out; }lreal_trunc_struct;

30.1.12 Typedef: lreal_2op_struct

Stuctname: lreal_2op_struct
Category: External IEC interface
Struct for operation with two 64-bit real values.
Typedef: typedef struct { RTS_IEC_LREAL in1; RTS_IEC_LREAL in2; RTS_IEC_LREAL out; }lreal_2op_struct;

30.1.13 Typedef: lreal_1op_struct

Stuctname: lreal_1op_struct
Category: External IEC interface
Struct for operation with one 64-bit real value.
Typedef: typedef struct { RTS_IEC_LREAL in; RTS_IEC_LREAL out; }lreal_1op_struct;

30.1.14 Typedef: lint_cmp_struct

Stuctname: lint_cmp_struct
Category: External IEC interface
Struct to compare two 64-bit integer values.
Typedef: typedef struct { RTS_IEC_LINT in1; RTS_IEC_LINT in2; RTS_IEC_SINT out; }lint_cmp_struct;

30.1.15 Typedef: lint_3op_struct

Stuctname: lint_3op_struct
Category: External IEC interface
Struct for operation with three 64-bit integer values.
Typedef: typedef struct { RTS_IEC_LINT in1; RTS_IEC_LINT in2; RTS_IEC_LINT in3; RTS_IEC_LINT out; }lint_3op_struct;

30.1.16 Typedef: lint_2op_struct

Stuctname: lint_2op_struct
Category: External IEC interface
Struct for operation with two 64-bit integer values.
Typedef: typedef struct { RTS_IEC_LINT in1; RTS_IEC_LINT in2; RTS_IEC_LINT out; }lint_2op_struct;

30.1.17 Typedef: lint_1op_struct

Stuctname: lint_1op_struct
Category: External IEC interface

Struct for operation with one 64-bit integer value.
Typedef: typedef struct { RTS_IEC_LINT in; RTS_IEC_LINT out; }lint_1op_struct;

30.1.18 Typedef: ulint_cmp_struct

Stuctname: ulint_cmp_struct
Category: External IEC interface
Struct to compare two 64-bit unsigned integer values.
Typedef: typedef struct { RTS_IEC_ULINT in1; RTS_IEC_ULINT in2; RTS_IEC_SINT out; }ulint_cmp_struct;

30.1.19 Typedef: ulint_3op_struct

Stuctname: ulint_3op_struct
Category: External IEC interface
Struct for operation with three 64-bit unsigned integer values.
Typedef: typedef struct { RTS_IEC_ULINT in1; RTS_IEC_ULINT in2; RTS_IEC_ULINT in3; RTS_IEC_ULINT out; }ulint_3op_struct;

30.1.20 Typedef: ulint_2op_struct

Stuctname: ulint_2op_struct
Category: External IEC interface
Struct for operation with two 64-bit unsigned integer values.
Typedef: typedef struct { RTS_IEC_ULINT in1; RTS_IEC_ULINT in2; RTS_IEC_ULINT out; }ulint_2op_struct;

30.1.21 Typedef: ulint_shift_struct

Stuctname: ulint_shift_struct
Category: External IEC interface
Struct for shift operations on a 64-bit unsigned integer value.
Typedef: typedef struct { RTS_IEC_ULINT in1; RTS_IEC_UINT in2; #ifdef RTS_SIXTEENBITBYTES RTS_IEC_USINT dummy[3]; #else RTS_IEC_USINT dummy[6]; #endif RTS_IEC_ULINT out; }ulint_shift_struct;

30.1.22 Typedef: lint_shift_struct

Stuctname: lint_shift_struct
Category: External IEC interface
Struct for shift operations on a 64-bit integer value.
Typedef: typedef struct { RTS_IEC_LINT in1; RTS_IEC_UINT in2; #ifdef RTS_SIXTEENBITBYTES RTS_IEC_USINT dummy[3]; #else RTS_IEC_USINT dummy[6]; #endif RTS_IEC_LINT out; }lint_shift_struct;

30.1.23 Typedef: ulint_1op_struct

Stuctname: ulint_1op_struct
Category: External IEC interface
Struct for operation with one 64-bit unsigned integer value.
Typedef: typedef struct { RTS_IEC_ULINT in; RTS_IEC_ULINT out; }ulint_1op_struct;

30.1.24 Typedef: dint_1op_struct

Stuctname: dint_1op_struct
Category: External IEC interface
Struct for operation with one 32-bit integer value.
Typedef: typedef struct { RTS_IEC_DINT in; RTS_IEC_DINT out; }dint_1op_struct;

30.1.25 Typedef: dint_2op_struct

Stuctname: dint_2op_struct
Category: External IEC interface
Struct for operation with two 32-bit integer values.
Typedef: typedef struct { RTS_IEC_DINT in1; RTS_IEC_DINT in2; RTS_IEC_DINT out; }dint_2op_struct;

30.1.26 Typedef: dint_3op_struct

Stuctname: dint_3op_struct
Category: External IEC interface

Struct for operation with three 32-bit integer values.
Typedef: typedef struct { RTS_IEC_DINT in1; RTS_IEC_DINT in2; RTS_IEC_DINT in3; RTS_IEC_DINT out; }dint_3op_struct;

30.1.27 Typedef: dint_shift_struct

Stuctname: dint_shift_struct
Category: External IEC interface
Struct for shift operations on a 32-bit integer value.
Typedef: typedef struct { RTS_IEC_DINT in1; RTS_IEC_INT in2; #ifndef TRG_C16X RTS_IEC_INT dummy; #endif RTS_IEC_DINT out; }dint_shift_struct;

30.1.28 Typedef: udint_2op_struct

Stuctname: udint_2op_struct
Category: External IEC interface
Struct for operation with two 32-bit unsigned integer values.
Typedef: typedef struct { RTS_IEC_UDINT in1; RTS_IEC_UDINT in2; RTS_IEC_UDINT out; }udint_2op_struct;

30.1.29 Typedef: udint_3op_struct

Stuctname: udint_3op_struct
Category: External IEC interface
Struct for operation with three 32-bit unsigned integer values.
Typedef: typedef struct { RTS_IEC_UDINT in1; RTS_IEC_UDINT in2; RTS_IEC_UDINT in3; RTS_IEC_UDINT out; }udint_3op_struct;

30.1.30 Typedef: udint_shift_struct

Stuctname: udint_shift_struct
Category: External IEC interface
Struct for shift operations on a 32-bit unsigned integer value.
Typedef: typedef struct { RTS_IEC_UDINT in1; RTS_IEC_INT in2; #ifndef TRG_C16X RTS_IEC_INT dummy; #endif RTS_IEC_UDINT out; }udint_shift_struct;

30.1.31 Typedef: int64_to_any32_struct

Stuctname: int64_to_any32_struct
Category: External IEC interface
Struct for conversion of a 64-bit integer value to any 32-bit value.
Typedef: typedef struct { RTS_IEC_ULINT in; RTS_IEC_UDINT uiType; RTS_IEC_UDINT out; }int64_to_any32_struct;

30.1.32 Typedef: any32_to_int64_struct

Stuctname: any32_to_int64_struct
Category: External IEC interface
Struct for conversion of any 32-bit value to a 64-bit integer value.
Typedef: typedef struct { RTS_IEC_UDINT in; RTS_IEC_UDINT uiType; RTS_IEC_ULINT out; }any32_to_int64_struct;

30.1.33 Typedef: real32_to_any32_struct

Stuctname: real32_to_any32_struct
Category: External IEC interface
Struct for conversion of a 32-bit real value to any 32-bit value.
Typedef: typedef struct { RTS_IEC_REAL in; RTS_IEC_UDINT uiType; RTS_IEC_UDINT out; }real32_to_any32_struct;

30.1.34 Typedef: any32_to_real32_struct

Stuctname: any32_to_real32_struct
Category: External IEC interface
Struct for conversion of any 32-bit value to a 32-bit real value.
Typedef: typedef struct { RTS_IEC_UDINT in; RTS_IEC_UDINT uiType; RTS_IEC_REAL out; }any32_to_real32_struct;

30.1.35 Typedef: real32_to_any64_struct

Stuctname: real32_to_any64_struct

Category: External IEC interface

Struct for conversion of a 32-bit real value to any 64-bit value.

Typedef: typedef struct { RTS_IEC_REAL in; RTS_IEC_UDINT uiType; RTS_IEC_ULINT out; }real32_to_any64_struct;

30.1.36 Typedef: any64_to_real32_struct

Stuctname: any64_to_real32_struct

Category: External IEC interface

Struct for conversion of any 64-bit value to a 32-bit real value.

Typedef: typedef struct { RTS_IEC_ULINT in; RTS_IEC_UDINT uiType; RTS_IEC_REAL out; }any64_to_real32_struct;

30.1.37 Typedef: real64_to_any32_struct

Stuctname: real64_to_any32_struct

Category: External IEC interface

Struct for conversion of a 64-bit real value to any 32-bit value.

Typedef: typedef struct { RTS_IEC_LREAL in; RTS_IEC_UDINT uiType; RTS_IEC_UDINT out; }real64_to_any32_struct;

30.1.38 Typedef: any32_to_real64_struct

Stuctname: any32_to_real64_struct

Category: External IEC interface

Struct for conversion of any 32-bit value to a 64-bit real value.

Typedef: typedef struct { RTS_IEC_UDINT in; RTS_IEC_UDINT uiType; RTS_IEC_LREAL out; }any32_to_real64_struct;

30.1.39 Typedef: real64_to_any64_struct

Stuctname: real64_to_any64_struct

Category: External IEC interface

Struct for conversion of a 64-bit real value to any 64-bit value.

Typedef: typedef struct { RTS_IEC_LREAL in; RTS_IEC_UDINT uiType; RTS_IEC_UDINT dummy; RTS_IEC_ULINT out; }real64_to_any64_struct;

30.1.40 Typedef: any64_to_real64_struct

Stuctname: any64_to_real64_struct

Category: External IEC interface

Struct for conversion of any 64-bit value to a 64-bit real value.

Typedef: typedef struct { RTS_IEC_ULINT in; RTS_IEC_UDINT uiType; RTS_IEC_UDINT dummy; RTS_IEC_LREAL out; }any64_to_real64_struct;

30.1.41 Typedef: real64_to_real32_struct

Stuctname: real64_to_real32_struct

Category: External IEC interface

Struct for conversion of a 64-bit real value to a 32-bit real value.

Typedef: typedef struct { RTS_IEC_LREAL in; RTS_IEC_REAL out; }real64_to_real32_struct;

30.1.42 Typedef: real32_to_real64_struct

Stuctname: real32_to_real64_struct

Category: External IEC interface

Struct for conversion of a 32-bit real value to a 64-bit real value.

Typedef: typedef struct { RTS_IEC_REAL in; RTS_IEC_UDINT dummy; RTS_IEC_LREAL out; }real32_to_real64_struct;

30.1.43 Typedef: TypeClass3

Category: Type class

Describes the type class correspoding to an IIECType instance.

Typedef: typedef enum { TYPE3_BOOL, TYPE3_BIT, TYPE3_BYTE, TYPE3_WORD, TYPE3_DWORD, TYPE3_LWORD, TYPE3_SINT, TYPE3_INT, TYPE3_DINT, TYPE3_LINT, TYPE3_USINT, TYPE3_UINT, TYPE3_UDINT, TYPE3_ULINT, TYPE3_REAL, TYPE3_LREAL, TYPE3_STRING, TYPE3_WSTRING, TYPE3_TIME, TYPE3_DATE, TYPE3_DATEANDTIME, TYPE3_TIMEOFDAY, TYPE3_POINTER, TYPE3_REFERENCE, TYPE3_SUBRANGE, TYPE3_ENUM, TYPE3_ARRAY, TYPE3_PARAMS, TYPE3_USERDEF, TYPE3_NONE, TYPE3_ANY, all types TYPE3_ANYBIT, all "bit"-types: bit, byte, word, dword, lword

TYPE3_ANYDATE, time, dat, tod, date TYPE3_ANYINT, TYPE3_ANYNUM, TYPE3_ANYREAL, TYPE3_LAZY, TYPE3_LTIME, TYPE3_BITCONST, TYPE3_MAX_TYPE } TypeClass3;

30.1.44 real32__eq

void real32__eq (real_cmp_struct p)*

This function determines if two operands equal. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.45 real32__ne

void real32__ne (real_cmp_struct p)*

This function determines if two operands are not equal. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.46 real32__lt

void real32__lt (real_cmp_struct p)*

This function determines if an operand is lower than another. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.47 real32__le

void real32__le (real_cmp_struct p)*

This function determines if an operand is lower or equal than another. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.48 real32__gt

void real32__gt (real_cmp_struct p)*

This function determines if an operand is greater than another. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.49 real32__ge

void real32__ge (real_cmp_struct p)*

This function determines if an operand is greater or equal than another. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.50 real32__add

void real32__add (real_2op_struct p)*

This function makes an addition of two 32-bit real values.

p [IN]
Pointer to the input structure

30.1.51 real32__sub

void real32__sub (real_2op_struct p)*

This function makes a subtraction of one variable from another one. The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.52 real32__mul

void real32__mul (real_2op_struct p)*

This function makes a multiplication of two variables. The inputs are 32-bit real values.

p [IN]

Pointer to the input structure

30.1.53 real32__div

void real32__div (real_2op_struct p)*

This function makes a division of two variables. The inputs are 32-bit real values.

The behaviour for divisor = 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.54 real32__min

void real32__min (real_2op_struct p)*
This function builds the minimum of two variables. The inputs are 32-bit real values.
p [IN]
Pointer to the input structure

30.1.55 real32__max

void real32__max (real_2op_struct p)*
This function builds the maximum of two variables. The inputs are 32-bit real values.
p [IN]
Pointer to the input structure

30.1.56 real32__limit

void real32__limit (real_3op_struct p)*
This function limits an input value to a lower and an upper bound. The inputs are 32-bit real values.
p [IN]
Pointer to the input structure

30.1.57 real32__trunc

void real32__trunc (real_trunc_struct p)*
This function converts a 32-bit real value to a 32-bit integer value.
p [IN]
Pointer to the input structure

30.1.58 real32__tan

void real32__tan (real_1op_struct p)*

This function calculates the tangent of a 32-bit real value.

The behaviour for input values that are multiples of PI_HALF might be platform dependent.

p [IN]
Pointer to the input structure

30.1.59 real32__sin

void real32__sin (real_1op_struct p)*
This function calculates the sine of a 32-bit real value.
p [IN]
Pointer to the input structure

30.1.60 real32__cos

void real32__cos (real_1op_struct p)*
This function calculates the cosine of a 32-bit real value.
p [IN]
Pointer to the input structure

30.1.61 real32__atan

void real32__atan (real_1op_struct p)*
This function calculates the arc tangent (inverse function of tangent) of a 32-bit real value.

p [IN]
Pointer to the input structure

30.1.62 real32__asin

void real32__asin (real_1op_struct p)*

This function calculates the arc sine (inverse function of sine) of a 32-bit real value.

The behaviour for input values < -1.0 and > 1.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.63 real32__acos

void real32__acos (real_1op_struct p)*

This function calculates the arc cosine (inverse function of cosine) of a 32-bit real value.

The behaviour for input values < -1.0 and > 1.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.64 real32__ln

void real32__ln (real_1op_struct p)*

This function calculates the natural logarithm of a 32-bit real value.

The behaviour for input values <= 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.65 real32__log

void real32__log (real_1op_struct p)*

This function calculates the logarithm in Base 10 of a 32-bit real value.

The behaviour for input values <= 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.66 real32__exp

void real32__exp (real_1op_struct p)*
This function calculates the exponential function of a 32-bit real value.
p [IN]
Pointer to the input structure

30.1.67 real32__sqrt

void real32__sqrt (real_1op_struct p)*

This function calculates the square root of a 32-bit real value.

The behaviour for input values < 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.68 real32__abs

void real32__abs (real_1op_struct p)*
This function calculates the absolute value of a 32-bit real value.
p [IN]
Pointer to the input structure

30.1.69 real32__expt

void real32__expt (real_2op_struct p)*

This function calculates the exponation of a variable with another variable.

The behaviour for input values in1 = 0.0 and in2 < 0.0 might be platform dependent.

The inputs are 32-bit real values.

p [IN]
Pointer to the input structure

30.1.70 real64__eq

void real64__eq (lreal_cmp_struct p)*
This function determines if two operands equal. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.71 real64__ne

void real64__ne (lreal_cmp_struct p)*
This function determines if two operands are not equal. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.72 real64__lt

void real64__lt (lreal_cmp_struct p)*
This function determines if an operand is lower than another. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.73 real64__le

void real64__le (lreal_cmp_struct p)*
This function determines if an operand is lower or equal than another. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.74 real64__gt

void real64__gt (lreal_cmp_struct p)*
This function determines if an operand is greater than another. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.75 real64__ge

void real64__ge (lreal_cmp_struct p)*
This function determines if an operand is greater or equal than another. The inputs are 64-bit real values.
p [IN]
Pointer to the input structure

30.1.76 real64__add

void real64__add (lreal_2op_struct p)*
This function makes an addition of two 64-bit real values.
p [IN]
Pointer to the input structure

30.1.77 real64__sub

void real64__sub (lreal_2op_struct p)*
This function makes a subtraction of one variable from another one. The inputs are 64-bit real values.
p [IN]

Pointer to the input structure

30.1.78 real64__mul

void real64__mul (lreal_2op_struct p)*

This function makes a multiplication of two variables. The inputs are 64-bit real values.

p [IN]
Pointer to the input structure

30.1.79 real64__div

void real64__div (lreal_2op_struct p)*

This function makes a division of two variables. The inputs are 64-bit real values.

The behaviour for divisor = 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.80 real64__min

void real64__min (lreal_2op_struct p)*

This function builds the minimum of two variables. The inputs are 64-bit real values.

p [IN]
Pointer to the input structure

30.1.81 real64__max

void real64__max (lreal_2op_struct p)*

This function builds the maximum of two variables. The inputs are 64-bit real values.

p [IN]
Pointer to the input structure

30.1.82 real64__limit

void real64__limit (lreal_3op_struct p)*

This function limits an input value to a lower and an upper bound. The inputs are 64-bit real values.

p [IN]
Pointer to the input structure

30.1.83 real64__trunc

void real64__trunc (lreal_trunc_struct p)*

This function converts a 64-bit real value to a 64-bit integer value.

p [IN]
Pointer to the input structure

30.1.84 real64__tan

void real64__tan (lreal_1op_struct p)*

This function calculates the tangent of a 64-bit real value.

The behaviour for input values that are multiples of PI_HALF might be platform dependent.

p [IN]
Pointer to the input structure

30.1.85 real64__sin

void real64__sin (lreal_1op_struct p)*

This function calculates the sine of a 64-bit real value.

p [IN]
Pointer to the input structure

30.1.86 real64__cos

void real64__cos (lreal_1op_struct p)*

This function calculates the cosine of a 64-bit real value.

p [IN]
Pointer to the input structure

30.1.87 real64__atan

void real64__atan (lreal_1op_struct p)*
This function calculates the arc tangent (inverse function of tangent) of a 64-bit real value.
p [IN]
Pointer to the input structure

30.1.88 real64__asin

void real64__asin (lreal_1op_struct p)*

This function calculates the arc sine (inverse function of sine) of a 64-bit real value.

The behaviour for input values < -1.0 and > 1.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.89 real64__acos

void real64__acos (lreal_1op_struct p)*

This function calculates the arc cosine (inverse function of cosine) of a 64-bit real value.

The behaviour for input values < -1.0 and > 1.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.90 real64__ln

void real64__ln (lreal_1op_struct p)*

This function calculates the natural logarithm of a 64-bit real value.

The behaviour for input values <= 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.91 real64__log

void real64__log (lreal_1op_struct p)*

This function calculates the logarithm in Base 10 of a 64-bit real value.

The behaviour for input values <= 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.92 real64__exp

void real64__exp (lreal_1op_struct p)*
This function calculates the exponential function of a 64-bit real value.
p [IN]
Pointer to the input structure

30.1.93 real64__sqrt

void real64__sqrt (lreal_1op_struct p)*

This function calculates the square root of a 64-bit real value.

The behaviour for input values < 0.0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.94 real64__abs

void real64__abs (lreal_1op_struct p)*
This function calculates the absolute value of a 64-bit real value.
p [IN]
Pointer to the input structure

30.1.95 real64__expt

void real64__expt (lreal_2op_struct p)*

This function calculates the exponation of a variable with another variable.

The behaviour for input values in1 = 0.0 and in2 < 0.0 might be platform dependent.

The inputs are 64-bit real values.

p [IN]
Pointer to the input structure

30.1.96 int64__add

void int64__add (lint_2op_struct p)*
This function makes an addition of two 64-bit integer values.
p [IN]
Pointer to the input structure

30.1.97 int64__sub

void int64__sub (lint_2op_struct p)*
This function makes a subtraction of one variable from another one. The inputs are 64-bit integer values.
p [IN]
Pointer to the input structure

30.1.98 int64__mul

void int64__mul (lint_2op_struct p)*
This function makes a multiplication of two variables. The inputs are 64-bit integer values.
p [IN]
Pointer to the input structure

30.1.99 int64__div

void int64__div (lint_2op_struct p)*

This function makes a division of two variables. The inputs are 64-bit integer values.

The behaviour for divisor = 0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.100 int64__mod

void int64__mod (lint_2op_struct p)*
This function calculates the remainder of division of one variable by another. The inputs are 64-bit integer values.
p [IN]
Pointer to the input structure

30.1.101 int64__abs

void int64__abs (lint_1op_struct p)*
This function calculates the absolute value of a 64-bit integer value.
p [IN]
Pointer to the input structure

30.1.102 int64__min

void int64__min (lint_2op_struct p)*
This function builds the minimum of two variables. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.103 int64__max

void int64__max (lint_2op_struct p)*

This function builds the maximum of two variables. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.104 int64__limit

void int64__limit (lint_3op_struct p)*

This function limits an input value to a lower and an upper bound. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.105 int64__eq

void int64__eq (lint_cmp_struct p)*

This function determines if two operands equal. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.106 int64__ne

void int64__ne (lint_cmp_struct p)*

This function determines if two operands are not equal. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.107 int64__lt

void int64__lt (lint_cmp_struct p)*

This function determines if an operand is lower than another. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.108 int64__le

void int64__le (lint_cmp_struct p)*

This function determines if an operand is lower or equal than another. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.109 int64__gt

void int64__gt (lint_cmp_struct p)*

This function determines if an operand is greater than another. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.110 int64__ge

void int64__ge (lint_cmp_struct p)*

This function determines if an operand is greater or equal than another. The inputs are 64-bit integer values.

p [IN]
Pointer to the input structure

30.1.111 int64__shr

void int64__shr (lint_shift_struct p)*

This function makes a bitwise right-shift on a 64-bit integer value.

The the newly exposed bits will be filled with the value of the topmost bit, that is 1 for negative and 0 for positive input values.

p [IN]
Pointer to the input structure

30.1.112 uint64__add

void uint64__add (ulint_2op_struct p)*

This function makes an addition of two 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.113 uint64__sub

void uint64__sub (ulint_2op_struct p)*

This function makes a subtraction of one variable from another one. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.114 uint64__mul

void uint64__mul (ulint_2op_struct p)*

This function makes a multiplication of two variables. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.115 uint64__div

void uint64__div (ulint_2op_struct p)*

This function makes a division of two variables. The inputs are 64-bit unsigned integer values.

The behaviour for divisor = 0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.116 uint64__mod

void uint64__mod (ulint_2op_struct p)*

This function calculates the remainder of division of one variable by another. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.117 uint64__min

void uint64__min (ulint_2op_struct p)*

This function builds the minimum of two variables. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.118 uint64__max

void uint64__max (ulint_2op_struct p)*

This function builds the maximum of two variables. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.119 uint64__limit

void uint64__limit (ulint_3op_struct p)*

This function limits an input value to a lower and an upper bound. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.120 uint64__ror

void uint64__ror (ulint_shift_struct p)*

This function makes a bitwise rotation to the right of a 64-bit unsigned integer value.

The input operand will be shifted one bit position to the right n times while the bit that is furthest to the left will be reinserted from the left.

p [IN]
Pointer to the input structure

30.1.121 uint64__rol

void uint64__rol (ulint_shift_struct p)*

This function makes a bitwise rotation to the left of a 64-bit unsigned integer value.

The input operand will be shifted one bit position to the left n times while the bit that is furthest to the right will be reinserted from the right.

p [IN]
Pointer to the input structure

30.1.122 uint64__shl

void uint64__shl (ulint_shift_struct p)*

This function makes a bitwise left-shift on a 64-bit unsigned integer value.

The the newly exposed bits will be filled with 0.

p [IN]
Pointer to the input structure

30.1.123 uint64__shr

void uint64__shr (ulint_shift_struct p)*

This function makes a bitwise right-shift on a 64-bit unsigned integer value.

The the newly exposed bits will be filled with 0.

p [IN]
Pointer to the input structure

30.1.124 uint64__and

void uint64__and (ulint_2op_struct p)*

This function makes a bitwise AND of two 64-bit unsigned integer values.

If the input bits each are 1, then the resulting bit will be 1, otherwise 0.

p [IN]
Pointer to the input structure

30.1.125 uint64__or

void uint64__or (ulint_2op_struct p)*

This function makes a bitwise OR of two 64-bit unsigned integer values.

If at least one of the input bits is 1, the resulting bit will be 1, otherwise 0.

p [IN]
Pointer to the input structure

30.1.126 uint64__xor

void uint64__xor (ulint_2op_struct p)*

This function makes a bitwise XOR of two 64-bit unsigned integer values.

If only one of the input bits is 1, then the resulting bit will be 1; if both or none are 1, the resulting bit will be 0.

p [IN]
Pointer to the input structure

30.1.127 uint64__not

void uint64__not (ulint_1op_struct p)*

This function makes a bitwise NOT of a 64-bit unsigned integer value.

The resulting bit will be 1, if the corresponding input bit is 0 and vice versa.

p [IN]
Pointer to the input structure

30.1.128 uint64__eq

void uint64__eq (ulint_cmp_struct p)*

This function determines if two operands equal. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.129 uint64__ne

void uint64__ne (ulint_cmp_struct p)*

This function determines if two operands are not equal. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.130 uint64__lt

void uint64__lt (ulint_cmp_struct p)*

This function determines if an operand is lower than another. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.131 uint64__le

void uint64__le (ulint_cmp_struct p)*

This function determines if an operand is lower or equal than another. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.132 uint64__gt

void uint64__gt (ulint_cmp_struct p)*

This function determines if an operand is greater than another. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.133 uint64__ge

void uint64__ge (ulint_cmp_struct p)*

This function determines if an operand is greater or equal than another. The inputs are 64-bit unsigned integer values.

p [IN]
Pointer to the input structure

30.1.134 int32__mul

void int32__mul (dint_2op_struct p)*

This function makes a multiplication of two variables. The inputs are 32-bit integer values.

p [IN]
Pointer to the input structure

30.1.135 uint32__mul

void uint32__mul (udint_2op_struct p)*
This function makes a multiplication of two variables. The inputs are 32-bit unsigned integer values.
p [IN]
Pointer to the input structure

30.1.136 int32__div

void int32__div (dint_2op_struct p)*

This function makes a division of two variables. The inputs are 32-bit integer values.

The behaviour for divisor = 0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.137 int32__mod

void int32__mod (dint_2op_struct p)*
This function calculates the remainder of division of one variable by another. The inputs are 32-bit integer values.
p [IN]
Pointer to the input structure

30.1.138 uint32__div

void uint32__div (udint_2op_struct p)*

This function makes a division of two variables. The inputs are 32-bit unsigned integer values.

The behaviour for divisor =0 might be platform dependent.

p [IN]
Pointer to the input structure

30.1.139 uint32__mod

void uint32__mod (udint_2op_struct p)*
This function calculates the remainder of division of one variable by another. The inputs are 32-bit unsigned integer values.
p [IN]
Pointer to the input structure

30.1.140 int32__abs

void int32__abs (dint_1op_struct p)*
This function calculates the absolute value of a 32-bit integer value.
p [IN]
Pointer to the input structure

30.1.141 uint32__limit

void uint32__limit (udint_3op_struct p)*
This function limits an input value to a lower and an upper bound. The inputs are 32-bit unsigned integer values.
p [IN]
Pointer to the input structure

30.1.142 int32__limit

void int32__limit (dint_3op_struct p)*
This function limits an input value to a lower and an upper bound. The inputs are 32-bit integer values.
p [IN]

Pointer to the input structure

30.1.143 uint32__rol

void uint32__rol (udint_shift_struct p)*

This function makes a bitwise rotation to the left of a 32-bit unsigned integer value.

The input operand will be shifted one bit position to the left n times while the bit that is furthest to the right will be reinserted from the right.

p [IN]
Pointer to the input structure

30.1.144 uint32__ror

void uint32__ror (udint_shift_struct p)*

This function makes a bitwise rotation to the right of a 32-bit unsigned integer value.

The input operand will be shifted one bit position to the right n times while the bit that is furthest to the left will be reinserted from the left.

p [IN]
Pointer to the input structure

30.1.145 uint32__shl

void uint32__shl (udint_shift_struct p)*

This function makes a bitwise left-shift on a 32-bit unsigned integer value.

The the newly exposed bits will be filled with 0.

p [IN]
Pointer to the input structure

30.1.146 uint32__shr

void uint32__shr (udint_shift_struct p)*

This function makes a bitwise right-shift on a 32-bit unsigned integer value.

The the newly exposed bits will be filled with 0.

p [IN]
Pointer to the input structure

30.1.147 int32__shr

void int32__shr (dint_shift_struct p)*

This function makes a bitwise right-shift on a 32-bit integer value.

The the newly exposed bits will be filled with the value of the topmost bit, that is 1 for negative and 0 for positive input values.

p [IN]
Pointer to the input structure

30.1.148 any32__to__int64

void any32__to__int64 (any32_to_int64_struct p)*

This function Converts any 32-bit numeric data type to a 64-bit integer value.

Data types with less than 32 bits are allowed but the input value has to be casted to 32 bits before.

Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.149 int64__to__any32

void int64__to__any32 (int64_to_any32_struct p)*

This function Converts a 64-bit integer value to any 32-bit numeric data type.
Data types with less than 32 bits are allowed but the output value is casted to 32 bits.
Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.150 real32__to__any32

void real32__to__any32 (real32_to_any32_struct p)*

This function Converts a 32-bit real value to any 32-bit numeric data type.
Data types with less than 32 bits are allowed but the output value is casted to 32 bits.
Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.151 any32__to__real32

void any32__to__real32 (any32_to_real32_struct p)*

This function Converts any 32-bit numeric data type to a 32-bit real value.
Data types with less than 32 bits are allowed but the input value has to be casted to 32 bits before.
Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.152 real32__to__any64

void real32__to__any64 (real32_to_any64_struct p)*

This function Converts a 32-bit real value to any 64-bit numeric data type.
Data types with less than 64 bits are allowed but the output value is casted to 64 bits.

p [IN]
Pointer to the input structure

30.1.153 any64__to__real32

void any64__to__real32 (any64_to_real32_struct p)*

This function Converts any 64-bit numeric data type to a 32-bit real value.
Data types with less than 64 bits are allowed but the input value has to be casted to 64 bits before.
Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.154 real64__to__any32

void real64__to__any32 (real64_to_any32_struct p)*

This function Converts a 64-bit real value to any 32-bit numeric data type.
Data types with less than 32 bits are allowed but the output value is casted to 32 bits.
Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.155 any32__to__real64

void any32__to__real64 (any32_to_real64_struct p)*

This function Converts any 32-bit numeric data type to a 64-bit real value.

Data types with less than 32 bits are allowed but the input value has to be casted to 32 bits before.

Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.156 real64__to__any64

void real64__to__any64 (real64_to_any64_struct p)*

This function Converts a 64-bit real value to any 64-bit numeric data type.

Data types with less than 64 bits are allowed but the output value is casted to 64 bits.

p [IN]
Pointer to the input structure

30.1.157 any64__to__real64

void any64__to__real64 (any64_to_real64_struct p)*

This function Converts any 64-bit numeric data type to a 64-bit real value.

Data types with less than 64 bits are allowed but the input value has to be casted to 64 bits before.

Only integer datatypes are allowed.

p [IN]
Pointer to the input structure

30.1.158 real64__to__real32

void real64__to__real32 (real64_to_real32_struct p)*

This function Converts a 64-bit real value to a 32-bit real value.

p [IN]
Pointer to the input structure

30.1.159 real32__to__real64

void real32__to__real64 (real32_to_real64_struct p)*

This function Converts a 32-bit real value to a 64-bit real value.

p [IN]
Pointer to the input structure

30.1.160 get__time

void get__time (get_time_struct p)*

This function returns a monotonic rising millisecond tick. This tick can be used for timeout and relative time measurements.

p [IN]
Pointer to the input structure

30.1.161 get__ltime

void get__ltime (get_ltime_struct p)*

This function returns a monotonic rising nanosecond tick. This tick can be used for very high resolution time measurements.

p [IN]
Pointer to the input structure

30.1.162 __memset

*void __memset (__memset_struct *p)*

Fill the buffer with a specified value. Routine is used as external library function for the plc program.

p [IN]
Pointer to the input structure

30.1.163 systimeunlock

*void systimeunlock (systimeunlock_struct *p)*

systimeunlock: Use SysTimeLock.libaray

p [IN]

Pointer to the input structure

30.1.164 systimelock

*void systimelock (systimelock_struct *p)*

systimelock: Use SysTimeLock.libaray

p [IN]

Pointer to the input structure

30.1.165 systimeunset

*void systimeunset (systimeunset_struct *p)*

Function to un-set the actual timestamp for all IEC timers.

p [IN]

Pointer to the input structure

30.1.166 systimeset

*void systimeset (systimeset_struct *p)*

Function to set the actual timestamp for all IEC timers. Differnt to SysTimeLock, the timer continues

p [IN]

Pointer to the input structure

30.1.167 SysGetTypeSize

unsigned int SysGetTypeSize (TypeClass3 tc)

Returns the size in bytes, of the specified IEC data type.

tc [IN]

Pointer to the input structure

Result

Size in Bytes of Datatype, or 0 if unspecified

31 SysMem

System component that allows access to memory functions.

31.1 SysMemItf

The SysMem interface is projected to get access to heap memory or special memory areas for the plc program.

31.1.1 Define: DA_NONE

Category: Area Types
Type:
Define: DA_NONE
Key: 0x0000
Application Area Type: None

31.1.2 Define: DA_DATA

Category: Area Types
Type:
Define: DA_DATA
Key: 0x0001
Application Area Type: Data

31.1.3 Define: DA_CONSTANT

Category: Area Types
Type:
Define: DA_CONSTANT
Key: 0x0002
Application Area Type: Constant

31.1.4 Define: DA_INPUT

Category: Area Types
Type:
Define: DA_INPUT
Key: 0x0004
Application Area Type: Input

31.1.5 Define: DA_OUTPUT

Category: Area Types
Type:
Define: DA_OUTPUT
Key: 0x0008
Application Area Type: Output

31.1.6 Define: DA_MEMORY

Category: Area Types
Type:
Define: DA_MEMORY
Key: 0x0010
Application Area Type: Memory

31.1.7 Define: DA_RETAIN

Category: Area Types
Type:
Define: DA_RETAIN
Key: 0x0020
Application Area Type: Retain

31.1.8 Define: DA_CODE

Category: Area Types
Type:
Define: DA_CODE
Key: 0x0040
Application Area Type: Code

31.1.9 Define: DA_PERSISTENT

Category: Area Types
Type:
Define: DA_PERSISTENT
Key: 0x0100
Application Area Type: Persistent

31.1.10 Define: DA_ALL

Category: Area Types
Type:
Define: DA_ALL
Key: 0xFFFF
Application Area Type: All

31.1.11 Define: IsArea

Condition: #ifndef DA
Category: Area Types
Type:
Define: IsArea
Key: type
Define for checking Area Types

31.1.12 Typedef: sysmemallocdata_struct

Stuctname: sysmemallocdata_struct
Category: External IEC interface

Struct for data allocation.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_DWORD ulSize; RTS_IEC_DWORD *pResult; RTS_IEC_BYTE *pMemResult; } sysmemallocdata_struct;

31.1.13 Typedef: sysmemreallocdata_struct

Stuctname: sysmemreallocdata_struct
Category: External IEC interface

Struct for data reallocation.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_BYTE *pMem; RTS_IEC_DWORD ulSize; RTS_IEC_DWORD *pResult; RTS_IEC_BYTE *pMemResult; } sysmemreallocdata_struct;

31.1.14 Typedef: sysmemfreedata_struct

Stuctname: sysmemfreedata_struct
Category: External IEC interface

Struct for freeing data.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_BYTE *pMem; RTS_IEC_DWORD Result; } sysmemfreedata_struct;

31.1.15 Typedef: sysmemallocarea_struct

Stuctname: sysmemallocarea_struct
Category: External IEC interface

Struct for area allocation.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_WORD usType; RTS_IEC_DWORD ulSize; RTS_IEC_DWORD *pResult; RTS_IEC_BYTE *pMemResult; } sysmemallocarea_struct;

31.1.16 Typedef: sysmemfreearea_struct

Stuctname: sysmemfreearea_struct
Category: External IEC interface

Struct for free area.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_BYTE *pMem; RTS_IEC_DWORD Result; } sysmemfreearea_struct;

31.1.17 Typedef: sysmemalloccode_struct

Stuctname: sysmemalloccode_struct
Category: External IEC interface

Struct for code allocation.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_DWORD ulSize; RTS_IEC_DWORD *pResult; RTS_IEC_BYTE *pMemResult; } sysmemalloccode_struct;

31.1.18 Typedef: sysmemfreecode_struct

Stuctname: sysmemfreecode_struct
Category: External IEC interface

Struct for free code.

Typedef: typedef struct { RTS_IEC_STRING *pszComponentName; RTS_IEC_BYTE *pCode; RTS_IEC_DWORD Result; } sysmemfreecode_struct;

31.1.19 Typedef: sysmemisvalidpointer_struct

Stuctname: sysmemisvalidpointer_struct
Category: External IEC interface

Struct for valid pointer check.

Typedef: typedef struct { RTS_IEC_BYTE* ptr; RTS_IEC_DWORD ulSize; RTS_IEC_BOOL bWrite; RTS_IEC_DINT Result; } sysmemisvalidpointer_struct;

31.1.20 Typedef: sysmemcpy_struct

Stuctname: sysmemcpy_struct
Category: External IEC interface

Struct for memcpy.

Typedef: typedef struct { RTS_IEC_BYTE *pDest; RTS_IEC_BYTE *pSrc; RTS_IEC_DINT iCount; RTS_IEC_BYTE *pMemResult; } sysmemcpy_struct;

31.1.21 Typedef: sysmemmove_struct

Stuctname: sysmemmove_struct
Category: External IEC interface

Struct for moving memory.

Typedef: typedef struct { RTS_IEC_BYTE *pDest; RTS_IEC_BYTE *pSrc; RTS_IEC_DINT iCount; RTS_IEC_BYTE *pMemResult; } sysmemmove_struct;

31.1.22 Typedef: sysmemcmp_struct

Stuctname: sysmemcmp_struct
Category: External IEC interface

Struct for memory comparision.

Typedef: typedef struct { RTS_IEC_BYTE *pBuffer1; RTS_IEC_BYTE *pBuffer2; RTS_IEC_DINT iCount; RTS_IEC_DINT out; } sysmemcmp_struct;

31.1.23 Typedef: sysmemset_struct

Stuctname: sysmemset_struct
Category: External IEC interface

Struct for setting memory.

Typedef: typedef struct { RTS_IEC_BYTE *pDest; RTS_IEC_DINT iValue; RTS_IEC_DINT iCount; RTS_IEC_BYTE *pMemResult; } sysmemset_struct;

31.1.24 Typedef: sysmemswap_struct

Stuctname: sysmemswap_struct
Category: External IEC interface

Struct for swapping memory.

Typedef: typedef struct { RTS_IEC_BYTE *pbyBuffer; RTS_IEC_DINT iSize; RTS_IEC_DINT iCount; RTS_IEC_DINT out; } sysmemswap_struct;

31.1.25 Typedef: sysmemforceswap_struct

Stuctname: sysmemforceswap_struct
Category: External IEC interface

Struct for data allocation.

Typedef: typedef struct { RTS_IEC_BYTE *pbyBuffer; RTS_IEC_DINT iSize; RTS_IEC_DINT iCount; RTS_IEC_DINT out; } sysmemforceswap_struct;

31.1.26 sysmemcpy

*void sysmemcpy (sysmemcpy_struct *p)*

Copy the content from source (pSrc) to destination buffer (pDest). Routine is used as external library function for the plc program.

31.1.27 sysmemmove

*void sysmemmove (sysmemmove_struct *p)*

Copy the content from source (pSrc) to destination buffer (pDest). This routine works for overlapping buffers too! Routine is used as external library function for the plc program.

31.1.28 sysmemcmp

*void sysmemcmp (sysmemcmp_struct *p)*

Compares the content of two buffers. Returns 0 if equal, else !=0 Routine is used as external library function for the plc program.

31.1.29 sysmemset

*void sysmemset (sysmemset_struct *p)*

Fill the buffer with a specified value. Routine is used as external library function for the plc program.

31.1.30 SysMemAllocData

void SysMemAllocData (char *pszComponentName, RTS_SIZE ulSize, RTS_RESULT *pResult)*

Allocates data memory of the specified size. IMPLEMENTATION NOTE: New allocated memory must be initialized with 0!

pszComponentName [IN]
Name of the component

ulSize [IN]
Requested size of the memory

pResult [OUT]
Pointer to error code

Result
Pointer to the memory block. NULL if no memory is available.

31.1.31 SysMemReallocData

void SysMemReallocData (char *pszComponentName, void* pData, RTS_SIZE ulSize, RTS_RESULT *pResult)*

Reallocate data memory with the specified size

pszComponentName [IN]
Name of the component

pData [IN]
Pointer to memory to resize

ulSize [IN]
Requested size of the memory

pResult [OUT]
Pointer to error code

Result
Pointer to the memory block. NULL if no memory is available.

31.1.32 SysMemFreeData

*RTS_RESULT SysMemFreeData (char *pszComponentName, void* pData)*

Release data memory

pszComponentName [IN]
Name of the component

pData [IN]
Pointer to memory to be released

Result
error code

31.1.33 SysMemAllocArea

void SysMemAllocArea (char *pszComponentName, unsigned short usType, RTS_SIZE ulSize, RTS_RESULT *pResult)*

Allocates data area of an application. Can be used to set an application area to a specific memory. IMPLEMENTATION NOTE: New allocated memory must be initialized with 0, but _not_ the retain memory!

pszComponentName [IN]
Name of the component

usType [IN]
Type of the area (see category Area Types)

ulSize [IN]
Requested size of the memory

pResult [OUT]
Pointer to error code

Result
Pointer to the memory block. NULL if no memory is available (allocate area on heap).

31.1.34 SysMemFreeArea

*RTS_RESULT SysMemFreeArea (char *pszComponentName, void* pData)*

Release data area of an application.

Note: On SIL2 Runtimes, this function may generate an exception, when called in SAFE-Mode.

pszComponentName [IN]
Name of the component

pData [IN]
Pointer to area to free

Result
error code

31.1.35 SysMemAllocCode

void SysMemAllocCode (char *pszComponentName, RTS_SIZE ulSize, RTS_RESULT *pResult)*

Allocate code memory with the specified size (in the memory, code can be executed). NOTE: This routine can be used on architectures, where standard data memory is protected against code execution! IMPLEMENTATION NOTE: New allocated memory must be initialized with 0!

pszComponentName [IN]
Name of the component

ulSize [IN]
Requested size of the memory

pResult [OUT]
Pointer to error code

Result
Pointer to the memory block. NULL if no memory is available.

31.1.36 SysMemFreeCode

*RTS_RESULT SysMemFreeCode (char *pszComponentName, void* pCode)*

Release code memory

pszComponentName [IN]
Name of the component

pData [IN]
Pointer to memory to resize

Result
error code

31.1.37 SysMemIsValidPointer

RTS_RESULT SysMemIsValidPointer (void ptr, RTS_SIZE ulSize, int bWrite)*

Check if a pointer points to a valid address.

ptr [IN]
Pointer to the memory to be checked

ulSize [IN]
Size of the memory to be checked

bWrite [IN]

1=Check, if memory can be written, 0=Check only for read access

Result

Error code:

- ERR_OK: Memory is valid
- ERR_FAILED: Memory is invalid. Cannot be accessed with the requested access mode bWrite

31.1.38 SysMemSwap

*int SysMemSwap (unsigned char *pbyBuffer, int iSize, int iCount)*

Routine to swap memory. If little endian (intel) byteorder is received and platform has big endian (motorola) byteorder. On little endian byteorder platforms, routine does nothing.

pbyBuffer [IN]

Pointer to data to swap. You can check, which order is selected by calling the routine with pbyBuffer=NULL

iSize [IN]

Size of one element to swap

iCount [IN]

Number of elements to swap

Result

-1 = failed (iSize too large) 0 = no swapping necessary (little endian byteorder) >0 = Number of bytes swapped (big endian byteorder) 1 = big endian byteorder, if pbyBuffer=NULL

31.1.39 SysMemForceSwap

*int SysMemForceSwap (unsigned char *pbyBuffer, int iSize, int iCount)*

Routine to force swapping memory independant of the byteorder of the system!

pbyBuffer [IN]

Pointer to data to swap. You can check, which order is selected by calling the routine with pbyBuffer=NULL

iSize [IN]

Size of one element to swap

iCount [IN]

Number of elements to swap

Result

-1 = failed (iSize too large) >0 = Number of bytes swapped

32 SysOut

System component that allows access to time functions.

32.1 SysOutItf

The SysOut interface is projected to get access to the console output routines. This can be used for debugging or logging needs.

32.1.1 Define: SysOutPrintfArg

Category:
Type:
Define: SysOutPrintfArg
Key: SysOutVPrintf
Function to print out an argument list on the standard console output

32.1.2 Define: CPT

Condition: #ifndef CPT
Category:
Type:
Define: CPT
Key: SysOutDebug
Checkpoints for debugging needs. Prints the actual file and line number

32.1.3 SysOutPrintf

*RTS_RESULT SysOutPrintf (char *szFormat, ...)*

Function to print out a formatted string on the standard console output

szFormat [IN]

Format string with optional arguments

Result

error code

32.1.4 SysOutDebug

*RTS_RESULT SysOutDebug (char *szFormat, ...)*

Debug output fo a formatted string on the standard console output. NOTE: Actual time should be added in front of each debug output string

szFormat [IN]

Format string with optional arguments

Result

error code

32.1.5 SysOutDebugArg

*RTS_RESULT SysOutDebugArg (char *szFormat, va_list *pargList)*

Debug output of an argument list on the standard console output. NOTE: Actual time should be added in front of each debug output string

szFormat [IN]

Format string

pargList [IN]

Pointer to argument list for the format string

Result

error code

32.2 CmpLogBackendItf

Interface of a logger backend, to store and dump log entries.

32.2.1 LogBackendCreate

*RTS_HANDLE LogBackendCreate (RTS_HANDLE hICmpLogBackend, CLASSID ClassId, struct tagLogOptions *pOptions)*

Create a logger

pOptions [IN]

Options for logger

pResult [OUT]
Pointer to get the result

Result
Handle to the logger, or RTS_INVALID_HANDLE if failed

32.2.2 LogBackendAdd

*RTS_HANDLE LogBackendAdd (RTS_HANDLE hICmpLogBackend, struct tagLogOptions *pOptions, struct tagLogEntry *pLog, RTS_RESULT *pResult)*

Add a new log entry

hLog [IN]
Handle to logger

pOptions [IN]
Log options

pLog [IN]
Log entry

pResult [OUT]
Result

Result
Handle to logger (logger could be split into a new logger)

32.2.3 LogBackendDelete

RTS_RESULT LogBackendDelete (RTS_HANDLE hICmpLogBackend)

Delete a logger

hLog [IN]
Handle to logger

Result
ERR_OK

33 SysSocket

System specific implementation of the sockets interface.

33.1 SysSocketItf

The SysSocket interface is projected to handle access to ethernet socket layer. TCP, UDP and RAW sockets can be used.

33.1.1 Define: SOCKET_AF_UNSPEC

Category: AddressFamily
Type:
Define: SOCKET_AF_UNSPEC
Key: 0
Socket family definitions

33.1.2 Define: SOCKET_SOL

Category: Socket level
Type:
Define: SOCKET_SOL
Key: 0xffff
Level number for SysSockGetOption()/SysSockSetOption() to apply to socket itself

33.1.3 Define: SOCKET_SO_DEBUG

Category: Socket options
Type:
Define: SOCKET_SO_DEBUG
Key: 0x0001
Socket options for SysSockGetOption()/SysSockSetOption()

33.1.4 Define: SOCKET_TCP_NODELAY

Category: Socket TCP options
Type:
Define: SOCKET_TCP_NODELAY
Key: 0x0001
Socket options for SysSockGetOption()/SysSockSetOption()

33.1.5 Define: SOCKET_STREAM

Category: Socket types
Type:
Define: SOCKET_STREAM
Key: 1
Different socket types

33.1.6 Define: SOCKET_IN_CLASSA

Category: Socket class handling
Type:
Define: SOCKET_IN_CLASSA
Key: i
Definitions of bits in internet address integers. On subnets, the decomposition of addresses to host and net parts is done according to subnet mask, not the masks here.

33.1.7 Define: SOCKET_IPPROTO_IP

Category: Socket protocols
Type:
Define: SOCKET_IPPROTO_IP
Key: 0
Socket protocols

33.1.8 Define: SOCKET_FIONREAD

Category: Ioctl commands
Type:
Define: SOCKET_FIONREAD
Key: 1
Control commands to set sockets to blocking or non-blocking

33.1.9 Define: SOCKET_SD_RECEIVE

Category: Shutdown flags
Type:
Define: SOCKET_SD_RECEIVE
Key: 0x00
Flags to specify, which operations are no longer be allowed

33.1.10 Define: SOCKET_MSG_NONE

Category: TCP flags
Type:
Define: SOCKET_MSG_NONE
Key: 0x00

33.1.11 Define: SOCKET_MTU_SIZE

Condition: #ifndef SOCKET_MTU_SIZE
Category: Static defines
Type:
Define: SOCKET_MTU_SIZE
Key: 1500
Only for SysSocketEmbedded: Set Transmit MTU size.

33.1.12 Define: SOCKET_BUFFER_SIZE

Condition: #ifndef SOCKET_BUFFER_SIZE
Category: Static defines
Type:
Define: SOCKET_BUFFER_SIZE
Key: SOCKET_MTU_SIZE*3
Only for SysSocketEmbedded: Set Receive Buffer Size.

33.1.13 Define: SYSSOCKET_NUM_OF_STATIC_SOCKETS

Condition: #ifndef SYSSOCKET_NUM_OF_STATIC_SOCKETS
Category: Static defines
Type:
Define: SYSSOCKET_NUM_OF_STATIC_SOCKETS
Key: 1
Only for SysSocketEmbedded: Number of supported sockets (one is enough for communication).

33.1.14 Define: SYSSOCKET_NUM_OF_STATIC_ARP_ENTRIES

Condition: #ifndef SYSSOCKET_NUM_OF_STATIC_ARP_ENTRIES
Category: Static defines
Type:
Define: SYSSOCKET_NUM_OF_STATIC_ARP_ENTRIES
Key: 5
Only for SysSocketEmbedded: Number of ARP entries in our cache.

33.1.15 Define: SYSSOCKET_DEFAULT_IP

Condition: #ifndef SYSSOCKET_DEFAULT_IP
Category: Static defines
Type:
Define: SYSSOCKET_DEFAULT_IP
Key: UINT32_C
Only for SysSocketEmbedded: Default IP Address (can be overwritten by setting).

33.1.16 Define: SYSSOCKET_DEFAULT_SUBNET

Condition: #ifndef SYSSOCKET_DEFAULT_SUBNET
Category: Static defines
Type:
Define: SYSSOCKET_DEFAULT_SUBNET
Key: UINT32_C
Only for SysSocketEmbedded: Default Subnet Mask (can be overwritten by setting).

33.1.17 Typedef: RTS_SOCKET_SO_VALUE_TCP_KEEPALIVE

Stuctname: RTS_SOCKET_SO_VALUE_TCP_KEEPALIVE
Category: TCP keepalve options

Parameters for the socket option SOCKET_SO_KEEPALIVE. NOTE: If one of the parameters is not supported, the result of SysSockSetOption() is ERR_NOT_SUPPORTED. In this case, the corresponding result of the option contains the error result.

Typedef: typedef struct RTS_SOCKET_SO_VALUE_TCP_KEEPALIVE_T { RTS_I32 bOn; RTS_UI32 probes; RTS_RESULT probesResult; RTS_UI32 timeout; RTS_RESULT timeoutResult; RTS_UI32 interval; RTS_RESULT intervalResult; } RTS_SOCKET_SO_VALUE_TCP_KEEPALIVE;

33.1.18 Typedef: INADDR

Stuctname: INADDR
Category: INADDR

Numeric IP-Address union to access different parts of the IP-address:

Typedef: typedef struct { union { struct { RTS_IEC_BYTE s_b1 ; RTS_IEC_BYTE s_b2 ; RTS_IEC_BYTE s_b3 ; RTS_IEC_BYTE s_b4 ; } S_un_b; struct { RTS_IEC_WORD s_w1 ; RTS_IEC_WORD s_w2 ; } S_un_w; RTS_IEC_UDINT S_addr ; } S_un; } INADDR;

33.1.19 Typedef: SOCKADDRESS

Stuctname: SOCKADDRESS
Category: INADDR

Numeric IP-Address union to access different parts of the IP-address:

Typedef: typedef struct { RTS_IEC_INT sin_family ; RTS_IEC_UINT sin_port ; INADDR sin_addr ; RTS_IEC_BYTE sin_zero[8] ; } SOCKADDRESS;

33.1.20 Typedef: UDP_REPLY

Stuctname: UDP_REPLY
Category: Udp reply

Udp reply information

Typedef: typedef struct UDP_REPLYtag { RTS_IEC_DWORD ulSourceAddress; RTS_IEC_STRING szSourceAddress[32]; RTS_IEC_DINT iRecv; RTS_IEC_WORD usRecvPort; } UDP_REPLY;

33.1.21 Typedef: syssockhtons_struct

Stuctname: syssockhtons_struct

syssockhtons

Typedef: typedef struct tagsyssockhtons_struct { RTS_IEC_WORD usHost; VAR_INPUT RTS_IEC_WORD SysSockHtons; VAR_OUTPUT } syssockhtons_struct;

33.1.22 Typedef: syssockntohl_struct

Stuctname: syssockntohl_struct

syssockntohl

Typedef: typedef struct tagsyssockntohl_struct { RTS_IEC_UDINT ulNet; VAR_INPUT RTS_IEC_UDINT SysSockNtohl; VAR_OUTPUT } syssockntohl_struct;

33.1.23 Typedef: syssockntohs_struct

Stuctname: syssockntohs_struct

syssockntohs

Typedef: typedef struct tagsyssockntohs_struct { RTS_IEC_WORD usNet; VAR_INPUT RTS_IEC_WORD SysSockNtohs; VAR_OUTPUT } syssockntohs_struct;

33.1.24 Typedef: syssockhtonl_struct

Stuctname: syssockhtonl_struct

syssockhtonl

Typedef: typedef struct tagsyssockhtonl_struct { RTS_IEC_UDINT ulHost; VAR_INPUT RTS_IEC_UDINT SysSockHtonl; VAR_OUTPUT } syssockhtonl_struct;

33.1.25 Typedef: syssockbind_struct

Stuctname: syssockbind_struct

syssockbind

Typedef: typedef struct tagsyssockbind_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT SOCKADDRESS *pSockAddr; VAR_INPUT RTS_IEC_DINT diSockAddrSize; VAR_INPUT RTS_IEC_UDINT SysSockBind; VAR_OUTPUT } syssockbind_struct;

33.1.26 Typedef: syssockaccept_struct

Stuctname: syssockaccept_struct

syssockaccept

Typedef: typedef struct tagsyssockaccept_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT SOCKADDRESS *pSockAddr; VAR_INPUT RTS_IEC_DINT *pdiSockAddrSize; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysSockAccept; VAR_OUTPUT } syssockaccept_struct;

33.1.27 Typedef: syssockclose_struct

Stuctname: syssockclose_struct

syssockclose

Typedef: typedef struct tagsyssockclose_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_UDINT SysSockClose; VAR_OUTPUT } syssockclose_struct;

33.1.28 Typedef: syssockcloseudp_struct

Stuctname: syssockcloseudp_struct

syssockcloseudp

Typedef: typedef struct tagsyssockcloseudp_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_UDINT SysSockCloseUdp; VAR_OUTPUT } syssockcloseudp_struct;

33.1.29 Typedef: syssockconnect_struct

Stuctname: syssockconnect_struct

syssockconnect

Typedef: typedef struct tagsyssockconnect_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT SOCKADDRESS *pSockAddr; VAR_INPUT RTS_IEC_DINT diSockAddrSize; VAR_INPUT RTS_IEC_UDINT SysSockConnect; VAR_OUTPUT } syssockconnect_struct;

33.1.30 Typedef: syssockcreate_struct

Stuctname: syssockcreate_struct

syssockcreate

Typedef: typedef struct tagsyssockcreate_struct { RTS_IEC_INT iAddressFamily; VAR_INPUT RTS_IEC_DINT diType; VAR_INPUT RTS_IEC_DINT diProtocol; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysSockCreate; VAR_OUTPUT } syssockcreate_struct;

33.1.31 Typedef: syssockcreateudp_struct

Stuctname: syssockcreateudp_struct

syssockcreateudp

Typedef: typedef struct tagsyssockcreateudp_struct { RTS_IEC_DINT diSendPort; VAR_INPUT RTS_IEC_DINT diRecvPort; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysSockCreateUdp; VAR_OUTPUT } syssockcreateudp_struct;

33.1.32 Typedef: syssockgethostbyname_struct

Stuctname: syssockgethostbyname_struct

syssockgethostbyname

Typedef: typedef struct tagsyssockgethostbyname_struct { RTS_IEC_STRING *szHostName; VAR_INPUT SOCK_HOSTENT *pHost; VAR_INPUT RTS_IEC_UDINT SysSockGetHostByName; VAR_OUTPUT } syssockgethostbyname_struct;

33.1.33 Typedef: syssockgethostname_struct

Stuctname: syssockgethostname_struct

syssockgethostname

Typedef: typedef struct tagsyssockgethostname_struct { RTS_IEC_STRING *szHostName; VAR_INPUT RTS_IEC_DINT diNameLen; VAR_INPUT RTS_IEC_UDINT SysSockGetHostName; VAR_OUTPUT } syssockgethostname_struct;

33.1.34 Typedef: syssockgetoption_struct

Stuctname: syssockgetoption_struct

syssockgetoption

Typedef: typedef struct tagsyssockgetoption_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_DINT diLevel; VAR_INPUT RTS_IEC_DINT diOption; VAR_INPUT RTS_IEC_DINT *pdiOptionValue; VAR_INPUT RTS_IEC_DINT *pdiOptionLen; VAR_INPUT RTS_IEC_UDINT SysSockGetOption; VAR_OUTPUT } syssockgetoption_struct;

33.1.35 Typedef: syssockgetoshandle_struct

Stuctname: syssockgetoshandle_struct

syssockgetoshandle

Typedef: typedef struct tagsyssockgetoshandle_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_HANDLE SysSockGetOSHandle; VAR_OUTPUT } syssockgetoshandle_struct;

33.1.36 Typedef: syssockgetrecvsizeudp_struct

Stuctname: syssockgetrecvsizeudp_struct

syssockgetrecvsizeudp

Typedef: typedef struct tagsyssockgetrecvsizeudp_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_DINT diTimeout; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockGetRecvSizeUdp; VAR_OUTPUT } syssockgetrecvsizeudp_struct;

33.1.37 Typedef: syssockgetsubnetmask_struct

Stuctname: syssockgetsubnetmask_struct

syssockgetsubnetmask

Typedef: typedef struct tagsyssockgetsubnetmask_struct { RTS_IEC_STRING *szIPAddress; VAR_INPUT RTS_IEC_STRING *szSubnetMask; VAR_INPUT RTS_IEC_DINT diMaxSugnetMask; VAR_INPUT RTS_IEC_UDINT SysSockGetSubnetMask; VAR_OUTPUT } syssockgetsubnetmask_struct;

33.1.38 Typedef: syssockinetaddr_struct

Stuctname: syssockinetaddr_struct

syssockinetaddr

Typedef: typedef struct tagsyssockinetaddr_struct { RTS_IEC_STRING *szIPAddress; VAR_INPUT RTS_IEC_UDINT *pInAddr; VAR_INPUT RTS_IEC_UDINT SysSockInetAddr; VAR_OUTPUT } syssockinetaddr_struct;

33.1.39 Typedef: syssockinetntoa_struct

Stuctname: syssockinetntoa_struct

syssockinetntoa

Typedef: typedef struct tagsyssockinetntoa_struct { INADDR *pInAddr; VAR_INPUT RTS_IEC_STRING *szIPADDR; VAR_INPUT RTS_IEC_DINT diIPAddrSize; VAR_INPUT RTS_IEC_UDINT SysSockInetNtoa; VAR_OUTPUT } syssockinetntoa_struct;

33.1.40 Typedef: syssockioctl_struct

Stuctname: syssockioctl_struct

syssockioctl

Typedef: typedef struct tagsyssockioctl_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_DINT diCommand; VAR_INPUT RTS_IEC_DINT *pdiParameter; VAR_INPUT RTS_IEC_UDINT SysSockIoctl; VAR_OUTPUT } syssockioctl_struct;

33.1.41 Typedef: syssocklisten_struct

Stuctname: syssocklisten_struct

syssocklisten

Typedef: typedef struct tagsyssocklisten_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_DINT diMaxConnections; VAR_INPUT RTS_IEC_UDINT SysSockListen; VAR_OUTPUT } syssocklisten_struct;

33.1.42 Typedef: syssockping_struct

Stuctname: syssockping_struct

syssockping

Typedef: typedef struct tagsyssockping_struct { RTS_IEC_STRING *szIPAddress; VAR_INPUT RTS_IEC_UDINT ulTimeout; VAR_INPUT RTS_IEC_UDINT *pulReplyTime; VAR_INPUT RTS_IEC_UDINT SysSockPing; VAR_OUTPUT } syssockping_struct;

33.1.43 Typedef: syssockrecv_struct

Stuctname: syssockrecv_struct

syssockrecv

Typedef: typedef struct tagsyssockrecv_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_DINT diBufferSize; VAR_INPUT RTS_IEC_DINT diFlags; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockRecv; VAR_OUTPUT } syssockrecv_struct;

33.1.44 Typedef: syssockrecvfrom_struct

Stuctname: syssockrecvfrom_struct

syssockrecvfrom

Typedef: typedef struct tagsyssockrecvfrom_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_DINT diBufferSize; VAR_INPUT RTS_IEC_DINT diFlags; VAR_INPUT SOCKADDRESS *pSockAddr; VAR_INPUT RTS_IEC_DINT diSockAddrSize; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockRecvFrom; VAR_OUTPUT } syssockrecvfrom_struct;

33.1.45 Typedef: syssockrecvfromudp_struct

Stuctname: syssockrecvfromudp_struct

syssockrecvfromudp

Typedef: typedef struct tagsyssockrecvfromudp_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_BYTE *pbyData; VAR_INPUT RTS_IEC_DINT diDataSize; VAR_INPUT UDP_REPLY_OLD *pReply; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockRecvFromUdp; VAR_OUTPUT } syssockrecvfromudp_struct;

33.1.46 Typedef: syssockrecvfromudp2_struct

Stuctname: syssockrecvfromudp2_struct

syssockrecvfromudp2

Typedef: typedef struct tagsyssockrecvfromudp2_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_BYTE *pbyData; VAR_INPUT RTS_IEC_DINT diDataSize; VAR_INPUT UDP_REPLY *pReply; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockRecvFromUdp2; VAR_OUTPUT } syssockrecvfromudp2_struct;

33.1.47 Typedef: syssockselect_struct

Stuctname: syssockselect_struct

syssockselect

Typedef: typedef struct tagsyssockselect_struct { RTS_IEC_DINT diWidth; VAR_INPUT SOCKET_FD_SET *pfdRead; VAR_INPUT SOCKET_FD_SET *pfdWrite; VAR_INPUT SOCKET_FD_SET *pfdExcept; VAR_INPUT SOCKET_TIMEVAL *ptvTimeout; VAR_INPUT RTS_IEC_DINT *pdiReady; VAR_INPUT RTS_IEC_UDINT SysSockSelect; VAR_OUTPUT } syssockselect_struct;

33.1.48 Typedef: syssocksend_struct

Stuctname: syssocksend_struct

syssocksend

Typedef: typedef struct tagsyssocksend_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_DINT diBufferSize; VAR_INPUT RTS_IEC_DINT diFlags; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockSend; VAR_OUTPUT } syssocksend_struct;

33.1.49 Typedef: syssocksendto_struct

Stuctname: syssocksendto_struct

syssocksendto

Typedef: typedef struct tagsyssocksendto_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_BYTE *pbyBuffer; VAR_INPUT RTS_IEC_DINT diBufferSize; VAR_INPUT RTS_IEC_DINT diFlags; VAR_INPUT SOCKADDRESS *pSockAddr; VAR_INPUT RTS_IEC_DINT diSockAddrSize; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockSendTo; VAR_OUTPUT } syssocksendto_struct;

33.1.50 Typedef: syssocksendtoudp_struct

Stuctname: syssocksendtoudp_struct

syssocksendtoudp

Typedef: typedef struct tagsyssocksendtoudp_struct { RTS_IEC_BYTE *hSocketUdp; VAR_INPUT RTS_IEC_DINT diPort; VAR_INPUT RTS_IEC_STRING *szDestAddress; VAR_INPUT RTS_IEC_BYTE *pbyData; VAR_INPUT RTS_IEC_DINT diDataSize; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_DINT SysSockSendToUdp; VAR_OUTPUT } syssocksendtoudp_struct;

33.1.51 Typedef: syssocksetipaddress_struct

Stuctname: syssocksetipaddress_struct

syssocksetipaddress

Typedef: typedef struct tagsyssocksetipaddress_struct { RTS_IEC_STRING *szCard; VAR_INPUT RTS_IEC_STRING *szIPAddress; VAR_INPUT RTS_IEC_UDINT SysSockSetIPAddress; VAR_OUTPUT } syssocksetipaddress_struct;

33.1.52 Typedef: syssocksetoption_struct

Stuctname: syssocksetoption_struct

syssocksetoption

Typedef: typedef struct tagsyssocksetoption_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_DINT diLevel; VAR_INPUT RTS_IEC_DINT diOption; VAR_INPUT RTS_IEC_DINT *pdiOptionValue; VAR_INPUT RTS_IEC_DINT diOptionLen; VAR_INPUT RTS_IEC_UDINT SysSockSetOption; VAR_OUTPUT } syssocksetoption_struct;

33.1.53 Typedef: syssocksetsubnetmask_struct

Stuctname: syssocksetsubnetmask_struct

syssocksetsubnetmask

Typedef: typedef struct tagsyssocksetsubnetmask_struct { RTS_IEC_STRING *szIPAddress; VAR_INPUT RTS_IEC_STRING *szSubnetMask; VAR_INPUT RTS_IEC_UDINT SysSockSetSubnetMask; VAR_OUTPUT } syssocksetsubnetmask_struct;

33.1.54 Typedef: syssockshutdown_struct

Stuctname: syssockshutdown_struct

syssockshutdown

Typedef: typedef struct tagsyssockshutdown_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT RTS_IEC_DINT diHow; VAR_INPUT RTS_IEC_UDINT SysSockShutdown; VAR_OUTPUT } syssockshutdown_struct;

33.1.55 Typedef: syssockfdisset_struct

Stuctname: syssockfdisset_struct

syssockfdisset

Typedef: typedef struct tagsyssockfdisset_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT SOCKET_FD_SET *pfs; VAR_INPUT RTS_IEC_BOOL SysSockFdIsset; VAR_OUTPUT } syssockfdisset_struct;

33.1.56 Typedef: syssockfdinit_struct

Stuctname: syssockfdinit_struct

syssockfdinit

Typedef: typedef struct tagsyssockfdinit_struct { RTS_IEC_BYTE *hSocket; VAR_INPUT SOCKET_FD_SET *pfs; VAR_INPUT RTS_IEC_UDINT SysSockFdInit; VAR_OUTPUT } syssockfdinit_struct;

33.1.57 Typedef: syssockfdzero_struct

Stuctname: syssockfdzero_struct

syssockfdzero

Typedef: typedef struct tagsyssockfdzero_struct { SOCKET_FD_SET *pfs; VAR_INPUT RTS_IEC_UDINT SysSockFdZero; VAR_OUTPUT } syssockfdzero_struct;

33.1.58 SysSockCreate

*RTS_HANDLE SysSockCreate (int iAddressFamily, int iType, int iProtocol, RTS_RESULT *pResult)*

Create a new socket and return the socket handle. In case of an error, RTS_INVALID_HANDLE is returned.

iAddressFamily [IN]
Socket address family

iType [IN]
Socket type

iProtocol [IN]
Socket protocol

pResult [OUT]
Pointer to error code

Result
Handle to the socket

33.1.59 SysSockCreateUdp

*RTS_HANDLE SysSockCreateUdp (int iSendPort, int iRecvPort, RTS_RESULT *pResult)*

Higher level function, to create a complete UDP socket

iSendPort [IN]
Port number to send (host byte order)

iRecvPort [IN]
Port number to receive (host byte order)

pResult [OUT]
Pointer to error code

Result
Handle to the UDP socket

33.1.60 SysSockSetOption

*RTS_RESULT SysSockSetOption (RTS_HANDLE hSocket, int iLevel, int iOption, char *pcOptionValue, int iOptionLen)*

Set options of a specified socket

hSocket [IN]
Handle to the socket

iLevel [IN]
Level of the socket

iOption [IN]
Socket option command

pcOptionValue [IN]
Pointer to the option value

iOptionLen [IN]
Lenght of option value

Result
error code

33.1.61 SysSockGetOption

*RTS_RESULT SysSockGetOption (RTS_HANDLE hSocket, int iLevel, int iOption, char *pcOptionValue, int *piOptionLen)*

Get options of a specified socket

hSocket [IN]
Handle to the socket

iLevel [IN]
Level of the socket

iOption [IN]
Socket option command

pcOptionValue [OUT]
Pointer to get the option value

piOptionLen [OUT]
Pointer to the option length. Real length is returned

Result
error code

33.1.62 SysSockBind

*RTS_RESULT SysSockBind (RTS_HANDLE hSocket, SOCKADDRESS *pSockAddr, int iSockAddrSize)*

Bind a socket to a socket address and port number

hSocket [IN]
Handle to the socket

pSockAddr [IN]
Spcket address

iSockAddrSize [IN]
Size of the socket address structure

Result

error code

33.1.63 SysSockListen

RTS_RESULT SysSockListen (RTS_HANDLE hSocket, int iMaxConnections)

Listen on a TCP server socket for new connection

hSocket [IN]
Handle to the socket

iMaxConnections [IN]
Maximum number of connections allowed

Result
error code

33.1.64 SysSockAccept

*RTS_HANDLE SysSockAccept (RTS_HANDLE hSocket, SOCKADDRESS *pSockAddr, int *piSockAddrSize, RTS_RESULT *pResult)*

Accept the next incoming TCP connection. Returns the socket for the newly created connection or RTS_INVALID_HANDLE if failed.

hSocket [IN]
Handle to the socket

pSockAddr [OUT]
Socket address of the client, who is connected

piSockAddrSize [INOUT]
Pointer to socket address structure

pResult [OUT]
Pointer to error code

Result
Handle to the new accepted socket

33.1.65 SysSockConnect

*RTS_RESULT SysSockConnect (RTS_HANDLE hSocket, SOCKADDRESS *pSockAddr, int iSockAddrSize)*

Connect as a client to a TCP server

hSocket [IN]
Handle to the socket

pSockAddr [IN]
Socket address of the client, who is connected

iSockAddrSize [IN]
Size of socket address structure

Result
error code

33.1.66 SysSockShutdown

RTS_RESULT SysSockShutdown (RTS_HANDLE hSocket, int iHow)

Shutdown a socket

hSocket [IN]
Handle to the socket

iHow [IN]
Specified, which operations are no longer be allowed. See category shutdown flags

Result
error code

33.1.67 SysSockIoctl

*RTS_RESULT SysSockIoctl (RTS_HANDLE hSocket, int iCommand, int *piParameter)*

Io-control of a socket

hSocket [IN]
Handle to the socket

iCommand [IN]
Io-control command

piParameter [INOUT]
Parameter value of the command
Result
error code

33.1.68 SysSockRecv

*int SysSockRecv (RTS_HANDLE hSocket, char *pbyBuffer, int iBufferSize, int iFlags, RTS_RESULT *pResult)*
Receive data from a TCP socket
hSocket [IN]
Handle to the socket
pbyBuffer [OUT]
Buffer to read data from the socket
iBufferSize [IN]
Maximum length of the buffer
iFlags [IN]
The flags parameter can be used to influence the behavior of the function beyond the options specified for the associated socket. The semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the SOCKET_MSG values. See category TCP flags.
pResult [OUT]
Pointer to error code
Result
Returns number of bytes received. 0 if failed.

33.1.69 SysSockSend

*int SysSockSend (RTS_HANDLE hSocket, char *pbyBuffer, int iBufferSize, int iFlags, RTS_RESULT *pResult)*
Sent data to a TCP socket
hSocket [IN]
Handle to the socket
pbyBuffer [IN]
Buffer with data to sent
iBufferSize [IN]
Maximum length of the buffer
iFlags [IN]
The flags parameter can be used to influence the behavior of the function beyond the options specified for the associated socket. The semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the SOCKET_MSG values.
pResult [OUT]
Pointer to error code
Result
Returns number of sent bytes. 0 if failed.

33.1.70 SysSockRecvFrom

*int SysSockRecvFrom (RTS_HANDLE hSocket, char *pbyBuffer, int iBufferSize, int iFlags, SOCKADDRESS *pSockAddr, int iSockAddrSize, RTS_RESULT *pResult)*
Receive a message from a connectionless socket (UDP)
hSocket [IN]
Handle to the socket
pbyBuffer [OUT]
Buffer to read data from the socket
iBufferSize [IN]
Maximum length of the buffer
iFlags [IN]
The flags parameter can be used to influence the behavior of the function beyond the options specified for the associated socket. The semantics of this function are determined by the socket

options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the SOCKET_MSG values.

pSockAddr [IN]
Socket address and port to receive data from

iSockAddrSize [IN]
Size of socket address structure

pResult [OUT]
Pointer to error code

Result
Returns number of bytes received

33.1.71 SysSockSendTo

*int SysSockSendTo (RTS_HANDLE hSocket, char *pbyBuffer, int iBufferSize, int iFlags, SOCKADDRESS *pSockAddr, int iSockAddrSize, RTS_RESULT *pResult)*

Send a message over a connectionless socket (UDP)

hSocket [IN]
Handle to the socket

pbyBuffer [IN]
Buffer with send data

iBufferSize [IN]
Length of data to send

iFlags [IN]
The flags parameter can be used to influence the behavior of the function beyond the options specified for the associated socket. The semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the SOCKET_MSG values.

pSockAddr [IN]
Socket address and port to sent data to

iSockAddrSize [IN]
Size of socket address structure

pResult [OUT]
Pointer to error code

Result
Returns number of bytes received

33.1.72 SysSockCloseUdp

RTS_RESULT SysSockCloseUdp (RTS_HANDLE hSocket)

Close a UDP socket

hSocket [IN]
Handle to the UDP socket. Must be opened with SysSockCreateUdp!

Result
error code

33.1.73 SysSockSendToUdp

*int SysSockSendToUdp (RTS_HANDLE hSocket, int iPort, char *pszDestAddress, unsigned char *pbyData, int iDataSize, RTS_RESULT *pResult)*

Send a paket to a UDP socket

hSocket [IN]
Handle to the UDP socket

iPort [IN]
Port number to send in host byteorder!

pszDestAddress [IN]
Destination IP address ot send data to

pbyData [IN]
Pointer to data to send

iDataSize [IN]
Size of data to send

pResult [OUT]
Pointer to error code

Result
Number of bytes sent

33.1.74 SysSockRecvFromUdp

*int SysSockRecvFromUdp (RTS_HANDLE hSocket, unsigned char *pbyData, int iDataSize, UDP_REPLY *pReply, RTS_RESULT *pResult)*
Receive a paket from a UDP socket
hSocket [IN]
Handle to the UDP socket
pbyData [OUT]
Pointer to data to receive
iDataSize [IN]
Size of data to receive
pReply [OUT]
Description of the client that has sent this packet. See category "Udp reply".
pResult [OUT]
Pointer to error code
Result
Number of bytes received

33.1.75 SysSockGetRecvSizeUdp

*int SysSockGetRecvSizeUdp (RTS_HANDLE hSocket, int iTimeout, RTS_RESULT *pResult)*
Check actual received data on the UDP socket
hSocket [IN]
Handle to the UDP socket
iTimeout [IN]
Timeout to wait for received data. -1=Infinite wait, 0=no wait
pResult [OUT]
Pointer to error code
Result
Number of bytes actual available in the socket

33.1.76 SysSockGetOSHandle

RTS_HANDLE SysSockGetOSHandle (RTS_HANDLE hSocket)
Get operating system handle of the UDP socket
hSocket [IN]
Handle to the UDP socket
Result
Operating system handle

33.1.77 SysSockFdIsset

*RTS_BOOL SysSockFdIsset (RTS_HANDLE hSocket, SOCKET_FD_SET *pfs)*
Check if a socket is inside of a set.
hSocket [IN]
Socket to check.
pfs [IN]
Socket Set
Result
TRUE if it is inside the set, FALSE if not

33.1.78 SysSockFdInit

*void SysSockFdInit (RTS_HANDLE hSocket, SOCKET_FD_SET *pfs)*
Add a socket to a socket set.
hSocket [IN]
Socket to add.
pfs [IN]
Socket Set

33.1.79 SysSockGetHostName

*RTS_RESULT SysSockGetHostName (char *pszHostName, int iNameLength)*

Get host name of the target

pszHostName [OUT]
Pointer to get host name

iNameLength [IN]
Maximum length of hostname

Result
error code

33.1.80 SysSockGetHostByName

*RTS_RESULT SysSockGetHostByName (char *pszHostName, SOCK_HOSTENT *pHost)*

Get host description specified by host name

pszHostName [IN]
Pointer to host name

pHost [OUT]
Pointer to host description

Result
error code

33.1.81 SysSockInetNtoa

*RTS_RESULT SysSockInetNtoa (INADDR *pInAddr, char *pszIPAddr, int iIPAddrSize)*

Convert IP address to a string

pInAddr [IN]
Pointer to IP address description

pszIPAddr [OUT]
Pointer to get IP address string (must be at least 16 bytes long)

iIPAddrSize [IN]
Maximum length of pszIPAddr

Result
error code

33.1.82 SysSockInetAddr

*RTS_RESULT SysSockInetAddr (char *pszIPAddress, RTS_UI32 *pInAddr)*

Convert an IP address string into an IP address

pszIPAddr [IN]
Pointer to get IP address string (must be at least 16 bytes long)

pInAddr [OUT]
Pointer to IP address description

Result
Error code:

- ERR_PARAMETER: if pszIPAddress=NULL or pInAddr=NULL
- ERR_OK: IP-address could be converted IMPLEMENTATION NOTE: If pszIPAddress="255.255.255.255", the error code must be ERR_OK with *pInAddr=0xFFFFFFFF (SOCKET_INADDR_BROADCAST).
- ERR_FAILED: IP-address invalid or empty IMPLEMENTATION NOTE: If pszIPAddress="", the error code must be ERR_FAILED with *pInAddr=0xFFFFFFFF (SOCKET_INADDR_NONE).

33.1.83 SysSockHtons

unsigned short SysSockHtons (unsigned short usHost)

Convert a host unsigned short value into the ethernet byte order

usHost [IN]
Host unsigned short value

Result
Returns the converted unsigned short value

33.1.84 SysSockHtonl

RTS_UI32 SysSockHtonl (RTS_UI32 ulHost)
Convert a host unsigned long value into the ethernet byte order
usHost [IN]
Host unsigned long value
Result
Returns the converted unsigned long value

33.1.85 SysSockNtohs

unsigned short SysSockNtohs (unsigned short usNet)
Convert a unsigned short value from ethernet byte order into host format
usNet [IN]
Ethernet unsigned short value
Result
Returns the converted unsigned short value

33.1.86 SysSockNtohl

RTS_UI32 SysSockNtohl (RTS_UI32 ulNet)
Convert a unsigned long value from ethernet byte order into host format
usNet [IN]
Ethernet unsigned long value
Result
Returns the converted unsigned long value

33.1.87 SysSockSetIPAddress

*RTS_RESULT SysSockSetIPAddress (char *pszCard, char *pszIPAddress)*
Set IP address of the specified ethernet device. Is not available on all platforms!
pszCard [IN]
Name of the ethernet card
pszIPAddress [IN]
IP address to set as string
Result
error code

33.1.88 SysSockSelect

*RTS_RESULT SysSockSelect (int iWidth, SOCKET_FD_SET *fdRead, SOCKET_FD_SET *fdWrite, SOCKET_FD_SET *fdExcept, SOCKET_TIMEVAL *ptvTimeout, int *pnReady)*
Check a number of sockets for activity
iWidth [IN]
Number of sockets in the SOCKET_FD_SET structure, so SOCKET_FD_SETSIZE must be used here.
fdRead [IN]
Read socket
fdWrite [IN]
Write socket
fdExcept [IN]
Exception socket
ptvTimeout [IN]
Pointer to specify the timeout of the operation. ptvTimeout=NULL: Infinite wait ptvTimeout->tv_sec=-1, ptvTimeout->tv_usec=-1: Infinite wait ptvTimeout->tv_sec=0, ptvTimeout->tv_usec=0: No wait
pnReady [OUT]
Number of sockets that are ready for IO
Result
ERR_OK or ERR_SOCK_TIMEDOUT, if timeout expired

33.1.89 SysSockPing

*RTS_RESULT SysSockPing (char *pszIPAddress, RTS_UI32 ulTimeout, RTS_UI32 *pulReplyTime)*
Check the availability of the communication partner with a ping request

pszIPAddress [IN]
IP address of the communication partner as string
ulTimeout [IN]
Timeout in milliseconds to wait until reply
pulReplyTime [OUT]
Pointer to get the reply time of the ping request in milliseconds or NULL
Result
ERR_OK: Partner available, ERR_TIMEOUT: Partner could not be reached during the specified timeout. All other results: Ping could not be sent because of other errors, so we don't know, if the partner is available.

33.1.90 SysSockSetSubnetMask

*RTS_RESULT SysSockSetSubnetMask (char *pszIPAddress, char *pszSubnetMask)*
Set subnetmask of a specified IP address adapter
pszIPAddress [IN]
IP address of the communication partner as string
pszSubnetMask [IN]
Subnet mask as string
Result
error code

33.1.91 SysSockGetSubnetMask

*RTS_RESULT SysSockGetSubnetMask (char *pszIPAddress, char *pszSubnetMask, int iMaxSubnetMask)*
Get subnetmask of a specified IP address adapter
pszIPAddress [IN]
IP address of the communication partner as string
pszSubnetMask [OUT]
Subnet mask as string
iMaxSubnetMask [IN]
Maximum length of the subnet mask string
Result
error code

33.1.92 SysSockFdZero

*void SysSockFdZero (SOCKET_FD_SET *pfs)*
Clear a Socket set.
pfs [IN]
Socket Set

33.1.93 SysSockClose

RTS_RESULT SysSockClose (RTS_HANDLE hSocket)
Close a socket.
hSocket [IN]
Handle to the socket
Result
error code

34 SysTarget

Target specific functions

Compiler Switch

- #define TRG_16BIT 16 Bit platform
- #define TRG_32BIT 32 Bit platform
- #define TRG_64BIT 64 Bit platform

34.1 SysTargetItf

The SysTarget interface is projected to get access to target specific informations. With this informations a target can be recognized unique in the complete network.

34.1.1 Define: SYSTARGET_DEVICE_MASK

Condition: #ifndef SYSTARGET_DEVICE_MASK
Category: DeviceID mask
Type:
Define: SYSTARGET_DEVICE_MASK
Key: 0x0000
Specifices, which parts of the DeviceID are checked in the signature. So a range of devices can use the same signature.

34.1.2 Define: SYSTARGET_SN_CPUID

Category: Serial number elements
Type:
Define: SYSTARGET_SN_CPUID
Key: CPUID
Specifies the elements of the device serial number. Each element is a single hardware charachterisic of a device and is optional.

34.1.3 Define: SYSTARGET_TYPE_SPECIAL_UNREGISTERED_SLOT

Category: Device Types
Type:
Define: SYSTARGET_TYPE_SPECIAL_UNREGISTERED_SLOT
Key: 0x0000
Special device with unregistered slots

34.1.4 Define: SYSTARGET_TYPE_PROGRAMMABLE

Category: Device Types
Type:
Define: SYSTARGET_TYPE_PROGRAMMABLE
Key: 0x1000
Programmable device

34.1.5 Define: SYSTARGET_TYPE_3S_SPECIAL_DEVICE

Category: Device Types
Type:
Define: SYSTARGET_TYPE_3S_SPECIAL_DEVICE
Key: 0x1001
3S special device (e.g. OfflineVisuClient)

34.1.6 Define: SYSTARGET_TYPE_SAFETY_DEVICE

Category: Device Types
Type:
Define: SYSTARGET_TYPE_SAFETY_DEVICE
Key: 0x1002
Safety device

34.1.7 Define: SYSTARGET_TYPE_DRIVE

Category: Device Types
Type:

Define: SYSTARGET_TYPE_DRIVE
Key: 0x1003
Drive device

34.1.8 Define: SYSTARGET_TYPE_PARAMETRIZABLE

Category: Device Types
Type:
Define: SYSTARGET_TYPE_PARAMETRIZABLE
Key: 0x1004
Parametrizable device

34.1.9 Define: SYSTARGET_TYPE_HMI

Category: Device Types
Type:
Define: SYSTARGET_TYPE_HMI
Key: 0x1005
Pure HMI device

34.1.10 Define: SYSTARGET_TYPE_3S_SOFTMOTION

Category: Device Types
Type:
Define: SYSTARGET_TYPE_3S_SOFTMOTION
Key: 0x1006
3S SoftMotion device

34.1.11 Define: SYSTARGET_TYPE_COMMUNICATION

Category: Device Types
Type:
Define: SYSTARGET_TYPE_COMMUNICATION
Key: 0x1007
Communication device (e.g. CoDeSys Gateway")

34.1.12 Typedef: systargetgettype2_struct

Stuctname: systargetgettype2_struct

Returns the target class

Typedef: typedef struct tagsystargetgettype2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_DWORD *pulType; VAR_INPUT RTS_IEC_UDINT SysTargetGetType2; VAR_OUTPUT } systargetgettype2_struct;

34.1.13 Typedef: systargetgetnodename2_struct

Stuctname: systargetgetnodename2_struct

Get a human readable name that identifies this node in the network. IMPLEMENTATION NOTE: This could be the registered host name of the target in the network.

Typedef: typedef struct tagsystargetgetnodename2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetNodeName2; VAR_OUTPUT } systargetgetnodename2_struct;

34.1.14 Typedef: systargetgetserialnumber_struct

Stuctname: systargetgetserialnumber_struct

Returns the serial number of the target. This can be a list of hardware specific signs (processor number, board number, mac-address, etc.).

Typedef: typedef struct tagsystargetgetserialnumber_struct { RTS_IEC_STRING **ppsSerialNumber; VAR_INPUT RTS_IEC_DINT *pnMaxLen; VAR_INPUT RTS_IEC_UDINT SysTargetGetSerialNumber; VAR_OUTPUT } systargetgetserialnumber_struct;

34.1.15 Typedef: systargetgetid_struct

Stuctname: systargetgetid_struct

Returns the TargetId. IMPLEMENTATION NOTE: Highword of the TargetId must be the VendorId! The VendorId is managed by 3S.

Typedef: typedef struct tagsystargetgetid_struct { RTS_IEC_DWORD *pulTargetId; VAR_INPUT RTS_IEC_UDINT SysTargetGetId; VAR_OUTPUT } systargetgetid_struct;

34.1.16 Typedef: systargetgetoperatingsystemid_struct

Stuctname: systargetgetoperatingsystemid_struct

Returns the ID of the operating system.

Typedef: typedef struct tagsystargetgetoperatingsystemid_struct { RTS_IEC_UDINT *pudiOperatingSystemId; VAR_INPUT RTS_IEC_UDINT SysTargetGetOperatingSystemId; VAR_OUTPUT } systargetgetoperatingsystemid_struct;

34.1.17 Typedef: systargetgetprocessorid_struct

Stuctname: systargetgetprocessorid_struct

Returns the ID of the processor

Typedef: typedef struct tagsystargetgetprocessorid_struct { RTS_IEC_UDINT *pudiProcessorId; VAR_INPUT RTS_IEC_UDINT SysTargetGetProcessorId; VAR_OUTPUT } systargetgetprocessorid_struct;

34.1.18 Typedef: systargetgetvendorname2_struct

Stuctname: systargetgetvendorname2_struct

Returns the vendor name

Typedef: typedef struct tagsystargetgetvendorname2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetVendorName2; VAR_OUTPUT } systargetgetvendorname2_struct;

34.1.19 Typedef: systargetgetdevicename2_struct

Stuctname: systargetgetdevicename2_struct

Returns the device name

Typedef: typedef struct tagsystargetgetdevicename2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetDeviceName2; VAR_OUTPUT } systargetgetdevicename2_struct;

34.1.20 Typedef: systargetgetnodename_struct

Stuctname: systargetgetnodename_struct

Get a human readable name that identifies this node in the network. IMPLEMENTATION NOTE: This could be the registered host name of the target in the network.

Typedef: typedef struct tagsystargetgetnodename_struct { RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetNodeName; VAR_OUTPUT } systargetgetnodename_struct;

34.1.21 Typedef: systargetgettype_struct

Stuctname: systargetgettype_struct

Returns the target class

Typedef: typedef struct tagsystargetgettype_struct { RTS_IEC_DWORD *pulType; VAR_INPUT RTS_IEC_UDINT SysTargetGetType; VAR_OUTPUT } systargetgettype_struct;

34.1.22 Typedef: systargetgetversion_struct

Stuctname: systargetgetversion_struct

Returns the target version

Typedef: typedef struct tagsystargetgetversion_struct { RTS_IEC_DWORD *pulVersion; VAR_INPUT RTS_IEC_UDINT SysTargetGetVersion; VAR_OUTPUT } systargetgetversion_struct;

34.1.23 Typedef: systargetgetversion2_struct

Stuctname: systargetgetversion2_struct

Returns the target version

Typedef: typedef struct tagsystargetgetversion2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_DWORD *pulVersion; VAR_INPUT RTS_IEC_UDINT SysTargetGetVersion2; VAR_OUTPUT } systargetgetversion2_struct;

34.1.24 Typedef: systargetgetdevicename_struct

Stuctname: systargetgetdevicename_struct

Returns the device name

Typedef: typedef struct tagsystargetgetdevicename_struct { RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetDeviceName; VAR_OUTPUT } systargetgetdevicename_struct;

34.1.25 Typedef: systargetgetid2_struct

Stuctname: systargetgetid2_struct

Returns the TargetId. IMPLEMENTATION NOTE: Highword of the TargetId must be the VendorId! The VendorId is managed by 3S.

Typedef: typedef struct tagsystargetgetid2_struct { RTS_IEC_STRING *pszRouterName; VAR_INPUT RTS_IEC_DWORD *pulTargetId; VAR_INPUT RTS_IEC_UDINT SysTargetGetId2; VAR_OUTPUT } systargetgetid2_struct;

34.1.26 Typedef: systargetgetvendorname_struct

Stuctname: systargetgetvendorname_struct

Returns the vendor name

Typedef: typedef struct tagsystargetgetvendorname_struct { RTS_IEC_WSTRING *pwszName; VAR_INPUT RTS_IEC_UDINT *pnMaxLength; VAR_INPUT RTS_IEC_UDINT SysTargetGetVendorName; VAR_OUTPUT } systargetgetvendorname_struct;

34.1.27 SysTargetGetNodeName

*RTS_RESULT SysTargetGetNodeName (RTS_WCHAR *pwszName, unsigned int *pnMaxLength)*

Get a human readable name that identifies this node in the network. IMPLEMENTATION NOTE: This could be the registered host name of the target in the network.

pwszName [IN]
Buffer that is filled with the name of the node. Type is 2 byte unicode!

pnMaxLength [INOUT]
Pointer to maximum length in unicode characters (not bytes!). Returns the number of bytes copied into the buffer including the trailing zero.

Result
error code

34.1.28 SysTargetGetConfiguredNodeName

*RTS_RESULT SysTargetGetConfiguredNodeName (RTS_WCHAR *pwszName, unsigned int *pnMaxLength)*

Get a human readable name for the target. This can be configured by the name setting (see category above).

pwszName [IN]
Buffer that is filled with the name of the node. Can be NULL to get the necessary length.

pnMaxLength [INOUT]
Pointer to maximum length in unicode characters (not bytes!). Returns the number of bytes copied into the buffer including the trailing zero.

Result
error code

34.1.29 SysTargetGetType

*RTS_RESULT SysTargetGetType (RTS_UI32 *pulType)*

Returns the target type

The possible target types are specified in the section "Device Types" (e.g. SYSTARGET_TYPE_PROGRAMMABLE).

Note: On SIL2 runtimes, this should return the value of the define SYSTARGET_DEVICE_TYPE.

pulType [INOUT]
Pointer to target type. See corresponding category "Device Types"

Result
error code

34.1.30 SysTargetGetId

*RTS_RESULT SysTargetGetId (RTS_UI32 *pulTargetId)*

Returns the TargetId of the PLC.

Note: Highword of the TargetId must be the VendorId! The VendorId is managed by 3S.

Note2: On SIL2 runtimes, this should return the combination of the defines SYSTARGET_VENDOR_ID and SYSTARGET_DEVICE_ID.

pulTargetId [INOUT]
Pointer to the TargetId

Result
error code

34.1.31 SysTargetGetVersion

*RTS_RESULT SysTargetGetVersion (RTS_UI32 *pulVersion)*

Returns the target version

Note: On SIL2 runtimes, this should return the value of the define SYSTARGET_DEVICE_VERSION.

pulVersion [INOUT]
Pointer to version of the target

Result
error code

34.1.32 SysTargetGetDeviceName

*RTS_RESULT SysTargetGetDeviceName (RTS_WCHAR *pwszName, unsigned int *pnMaxLength)*
Returns the device name

pwszName [INOUT]
Pointer to the device name. Can be NULL to get the necessary length.

pnMaxLength [INOUT]
Pointer to maximum length of the name in unicode characters (not bytes!). Returns the number of bytes copied into the buffer including the trailing zero.

Result
error code

34.1.33 SysTargetGetVendorName

*RTS_RESULT SysTargetGetVendorName (RTS_WCHAR *pwszName, unsigned int *pnMaxLength)*
Returns the vendor name

pwszName [INOUT]
Pointer to the device name. Can be NULL to get the necessary length.

pnMaxLength [INOUT]
Pointer to maximum length of the name in unicode characters (not bytes!). Returns the number of bytes copied into the buffer including the trailing zero.

Result
error code

34.1.34 SysTargetGetOperatingSystemId

*RTS_RESULT SysTargetGetOperatingSystemId (RTS_UI32 *pulOperatingSystemId)*
Returns the ID of the operating system.

pulOperatingSystemID [INOUT]
Pointer to operating system Id. See category "Operating System" above

Result
error code

34.1.35 SysTargetGetProcessorId

*RTS_RESULT SysTargetGetProcessorId (RTS_UI32 *pulProcessorId)*
Returns the ID of the processor

pulProcessorID [INOUT]
Pointer to processor ID. See category "Processor ID" above

Result
error code

34.1.36 SysTargetGetSerialNumber

*RTS_RESULT SysTargetGetSerialNumber (char **ppszSerialNumber, int *pnMaxLen)*

Returns the serial number of the target. This can be a list of hardware specific signs (processor number, board number, mac-address, etc.).

The serial number must contain device unique identifiers! It can contain a comma separated list of different identifiers like: "CPUID=0x000006FB | 0x00020800 | 0x0000E3BD | 0xBFEBFBFF, HDDSN=0x12345678" Some keywords are predefined in the macros SYSTARGET_SN_XXX. See category "Serial number elements".

- If ppszSerialNumber==NULL, the length of the serial number can be retrieved in *pnMaxLen.
- If *ppszSerialNumber==NULL, the pointer will be set to the static serial nubmer. *pnMaxLen contains the real length of the serial number.
- If *ppszSerialNumber!=NULL, the serial number will be written into the buffer. *pnMaxLen must specify the max length of the buffer!

IMPLEMENTATION NOTE: The length of the serial number string must be limited to 512 bytes!

ppszSerialNumber [INOUT]
Pointer to pointer to serial number.

pnMaxLen [INOUT]
Pointer to the max length of the string (if *ppszSerialNumber!=NULL) or the length that is returned by the function.

Result
error code

34.1.37 SysTargetGetSignature

*RTS_RESULT SysTargetGetSignature (RTS_UI32 ulChallenge, RTS_UI32 *pulSignature)*

Returns the signature of SysTarget

ulChallenge [IN]
Challenge to get the signature

pulSignature [INOUT]
Signature of the SysTarget entries Type, Id, OperatingSystem, ProcessorType

Result
error code

34.1.38 SysTargetGetDeviceMask

*RTS_RESULT SysTargetGetDeviceMask (RTS_UI16 *pusDeviceMask)*

Returns the device mask. It is used to use the same signature for a range of devices. Example:
*pusDeviceMask = 0x0000: All parts of the DeviceID is used to generate the signature
*pusDeviceMask = 0x00FF: Only the high BYTE of the DeviceID is used to generate the signature. So a range of 255 devices can be used with the same signature.

pusDeviceMask [INOUT]
Pointer to return the device mask

Result
error code

35 SysTime

System component that allows access to time functions.

35.1 SysTimeItf

The SysTime interface is projected to get access to time tick values with different resolutions (millisecond, microsecond, nanosecond).

All different ticks are wrapping around their natural data type limits (RTS_UI32 or RTS_SYSTIME). Therefore, they can be used to measure difference timings. They can't either be used to measure absolute timings nor can they be calculated from one to each other.

Implementation Notes:

- All three different timers need their own handling of wrap arounds, because they have a wrap around at different points in time at different boundaries. In practice, this implies the need for an own, static offset counter for every timer.
- If the timer is based on a periodic interrupt (e.g. millisecond tick), it works only as long as no interrupt lock is held. This might be the case in some flash drivers for example.
- Some timers are called very regularly. So if the wrap around of the timer source itself is very late (low frequency + large timer register), it might be enough for the system to detect timer overruns only at every call.

35.1.1 SysTimeGetMs

RTS_UI32 SysTimeGetMs (void)

Return a monotonic, rising millisecond tick.

Result
Returns the millisecond tick

35.1.2 SysTimeGetUs

RTS_RESULT SysTimeGetUs (RTS_SYSTIME pTime)*

Return a monotonic, rising microsecond tick.

pTime [INOUT]
Pointer to the time tick result

Result
error code

35.1.3 SysTimeGetNs

RTS_RESULT SysTimeGetNs (RTS_SYSTIME pTime)*

Return a monotonic, rising nanosecond tick.

pTime [INOUT]
Pointer to the time tick result

Result
error code

36 SysTimer

36.1 SysTimerItf

The SysTimer interface is projected to access timer devices on target.

36.1.1 Define: TIMER_NO_ERROR

Category: Timer error codes
Type:
Define: TIMER_NO_ERROR
Key: 0x0000
Possible Error codes: TIMER_NO_ERROR TIMER_HANDLE_INVALID TIMER_SYS_SPEC_ERROR TIMER_MANUF_SPEC_ERROR

36.1.2 Define: RTS_TIMER_NONE

Category: Timer type
Type:
Define: RTS_TIMER_NONE
Key: 0
Possible type of a Timer: RTS_TIMER_NONE: Not Defined RTS_TIMER_PERIODIC: Periodical timer RTS_TIMER_ONESHOT: Oneshot timer

36.1.3 Define: SYSTIMER_NUM_OF_STATIC_TIMER

Condition: #ifndef SYSTIMER_NUM_OF_STATIC_TIMER
Category: Static defines
Type:
Define: SYSTIMER_NUM_OF_STATIC_TIMER
Key: 2
Maximum number of timers

36.1.4 Typedef: SYS_TIMER_INFO

Stuctname: SYS_TIMER_INFO
Category: Timer info structure

Timer information structure that contains all information for the SysTimerOS implementation to handle one sepcific timer object.

Typedef: typedef struct { PFTIMERCALLBACK pTimerCallback; PFTIMEREXCEPTIONHANDLER pExceptionHandler; RTS_HANDLE hParam; RTS_HANDLE hSysTimer; RTS_HANDLE hTimerToReset; RTS_SYSTIME tIntervalNs; RTS_SYSTIME tStartTime; RTS_SYSTIME tLastExecuteNs; unsigned long ulType; unsigned long ulPriority; SYS_TIMER_CALL_CONTEXT tCallContext; SYS_TIMER_BP_CONTEXT tBPContext; int bIECFunction; int iState; RTS_HANDLE hEvent; unsigned long ulIRQ; } SYS_TIMER_INFO;

36.1.5 Typedef: systimercreatecallback2_struct

Stuctname: systimercreatecallback2_struct

This function creates a timer and calls a callback function. The scheduler creates a new timer for the schedule tick. If the creation fails, the scheduler sets up a task instead. If the scheduler should use a task on default, SysTimerCreateCallback must return the error code ERR_NOTIMPLEMENTED.

Typedef: typedef struct tagsystimercreatecallback2_struct { PFTIMEREXCEPTIONHANDLER pfTimerCallback; VAR_INPUT RTS_IEC_BYTE *hParam; VAR_INPUT RTS_IEC_ULINT tIntervalNs; VAR_INPUT RTS_IEC_UDINT ulPriority; VAR_INPUT RTS_IEC_UDINT ulType; VAR_INPUT PFTIMEREXCEPTIONHANDLER pfExceptionHandler; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysTimerCreateCallback2; VAR_OUTPUT } systimercreatecallback2_struct;

36.1.6 Typedef: systimerstop_struct

Stuctname: systimerstop_struct

This function stops a timer

Typedef: typedef struct tagsystimerstop_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_UDINT SysTimerStop; VAR_OUTPUT } systimerstop_struct;

36.1.7 Typedef: systimerdelete_struct

Stuctname: systimerdelete_struct

This function deletes a timer

Typedef: typedef struct tagsystimerdelete_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_UDINT SysTimerDelete; VAR_OUTPUT } systimerdelete_struct;

36.1.8 Typedef: systimergettimestamp_struct

Stuctname: systimergettimestamp_struct

This function returns the timestamp in ticks since timer start

Typedef: typedef struct tagsystimergettimestamp_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_ULINT *ptTimestampNs; VAR_INPUT RTS_IEC_UDINT SysTimerGetTimeStamp; VAR_OUTPUT } systimergettimestamp_struct;

36.1.9 Typedef: systimersetinterval_struct

Stuctname: systimersetinterval_struct

This function returns the interval of a timer

Typedef: typedef struct tagsystimersetinterval_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_ULINT tIntervalNs; VAR_INPUT RTS_IEC_UDINT SysTimerSetInterval; VAR_OUTPUT } systimersetinterval_struct;

36.1.10 Typedef: systimergetinterval_struct

Stuctname: systimergetinterval_struct

Returns the interval of a timer

Typedef: typedef struct tagsystimergetinterval_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_ULINT *ptIntervalNs; VAR_INPUT RTS_IEC_UDINT SysTimerGetInterval; VAR_OUTPUT } systimergetinterval_struct;

36.1.11 Typedef: systimercreateevent_struct

Stuctname: systimercreateevent_struct

This function creates a timer and sets an event

Typedef: typedef struct tagsystimercreateevent_struct { RTS_IEC_BYTE *hEvent; VAR_INPUT RTS_IEC_ULINT tIntervalNs; VAR_INPUT RTS_IEC_UDINT ulPriority; VAR_INPUT PFTIMEREXCEPTIONHANDLER pfExceptionHandler; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysTimerCreateEvent; VAR_OUTPUT } systimercreateevent_struct;

36.1.12 Typedef: systimermaxtimer_struct

Stuctname: systimermaxtimer_struct

This function returns the maximal number of timers

Typedef: typedef struct tagsystimermaxtimer_struct { RTS_IEC_UDINT *pulMaxTimer; VAR_INPUT RTS_IEC_UDINT SysTimerMaxTimer; VAR_OUTPUT } systimermaxtimer_struct;

36.1.13 Typedef: systimerstart_struct

Stuctname: systimerstart_struct

This function starts a timer

Typedef: typedef struct tagsystimerstart_struct { RTS_IEC_BYTE *hTimer; VAR_INPUT RTS_IEC_UDINT ulType; VAR_INPUT RTS_IEC_UDINT SysTimerStart; VAR_OUTPUT } systimerstart_struct;

36.1.14 Typedef: systimercreatecallback_struct

Stuctname: systimercreatecallback_struct

This function creates a timer and calls a callback function

Typedef: typedef struct tagsystimercreatecallback_struct { PFTIMERCALLBACK pfTimerCallback; VAR_INPUT RTS_IEC_BYTE *hParam; VAR_INPUT RTS_IEC_ULINT tIntervalNs; VAR_INPUT RTS_IEC_UDINT ulPriority; VAR_INPUT PFTIMEREXCEPTIONHANDLER pfExceptionHandler; VAR_INPUT RTS_IEC_UDINT *pResult; VAR_INPUT RTS_IEC_BYTE *SysTimerCreateCallback; VAR_OUTPUT } systimercreatecallback_struct;

36.1.15 SysTimerCreateEvent

*RTS_HANDLE SysTimerCreateEvent (RTS_HANDLE hEvent, RTS_SYSTIME tIntervalNs, unsigned long ulPriority, PFTIMEREXCEPTIONHANDLER pfExceptionHandler, RTS_RESULT *pResult)*

This function creates a timer and sets an event

hEvent [IN]

Handle to the event that is sent after the interval expires

tIntervalNs [IN]

Period of the timer (timebase = 1 ns)

ulPriority [IN]
Priority of the timer object
pfExceptionHandler [IN]
Pointer to an optional exception handler. Can be NULL.
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Handle of the timer or RTS_INVALID_HANDLE

36.1.16 SysTimerCreateCallback

*RTS_HANDLE SysTimerCreateCallback (PFTIMERCALLBACK pfTimerCallback, RTS_HANDLE hParam, int bIECFunction, RTS_SYSTIME tIntervalNs, unsigned long ulPriority, PFTIMEREXCEPTIONHANDLER pfExceptionHandler, RTS_RESULT *pResult)*

This function creates a system timer, which cyclically calls the callback function that is passed with the parameter pfTimerCallback.

This callback has to be of type PFTIMERCALLBACK and gets a the handle which is specified by hParam passed on every call.

The supported timer intervals may be limited by the hardware and/or underlying operating systems.

pfTimerCallback [IN]
Pointer to a callback function
hParam [IN]
Parameter for callback routine
bIECFunction [IN]
Is IEC function
tIntervalNs [IN]
Interval of the timer (timebase = 1 ns)
ulPriority [IN]
Priority of the timer object
pfExceptionHandler [IN]
Pointer to an optional exception handler. Can be NULL.
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Handle of the timer or RTS_INVALID_HANDLE

36.1.17 SysTimerCreateCallback2

*RTS_HANDLE SysTimerCreateCallback2 (PFTIMERCALLBACK pfTimerCallback, RTS_HANDLE hParam, int bIECFunction, RTS_SYSTIME tIntervalNs, unsigned long ulPriority, unsigned long ulType, PFTIMEREXCEPTIONHANDLER pfExceptionHandler, RTS_RESULT *pResult)*

This function creates a system timer, which calls the callback function that is passed with the parameter pfTimerCallback.

This callback has to be of type PFTIMERCALLBACK and gets a the handle which is specified by hParam passed on every call.

Supported are timers with a different behavior (e.g. cyclic timers and one-shot timers). They are specified in the categorie "Timer type"

The supported timer intervals may be limited by the hardware and/or underlying operating systems. For one-shot timers, the interval just specifies the time for the next shot.

For the case, that the underlying operating system supports only exception handling based on tasks, you can pass a task as an exception handler with the parameter pfExceptionHandler.

Timers are priorized like IEC Tasks. There may be 256 Priorities, but this number is in fact limited by the system adaptation.

pfTimerCallback [IN]
Pointer for the Timer callback. This callback is called when ever the timer event occured.
hParam [IN]
Parameter that is passed to the timer callback.

bIECFunction [IN]
Specify if the Callback is an IEC Function or a C Function. This parameter is only used internally and may be 0 in most cases.

tIntervalNs [IN]
Interval of the timer (timebase = 1 ns). The resolution may vary because of limitations from the hardware or underlying operating system.

ulPriority [IN]
Priority of the timer object

ulType [IN]
Not all timers from the category "Timer type" might be supported by the system. The only relyable timer is the periodic timer.

pfExceptionHandler [IN]
Pointer to an optional exception handler. Can be NULL.

pResult [OUT]
Result pointer containing the error code. Might be NULL.

Result
Handle of the timer or RTS_INVALID_HANDLE if there was an error.

36.1.18 SysTimerOpen

*RTS_HANDLE SysTimerOpen (SYS_TIMER_INFO *pTimerInfo, RTS_RESULT *pResult)*

This function creates a system specific timer. IMPLEMENTATION NOTE: The timer must be disabled (in stop state) after returning this routine! The timer must be started explicitly with SysTimerStart.

pTimerInfo [IN]
Parameter for timer

pResult [OUT]
Result pointer containing the error code. Might be NULL.

Result
Handle of the timerinfo or RTS_INVALID_HANDLE

36.1.19 SysTimerClose

RTS_RESULT SysTimerClose (RTS_HANDLE hTimer)

This function closes a timer

hTimer [IN]
Handle of the timer

Result
Error code

36.1.20 SysTimerStart

RTS_RESULT SysTimerStart (RTS_HANDLE hTimer, unsigned long ulType)

This function starts a timer

hTimer [IN]
Handle of the timer

ulType [IN]
Timer type. See corresponding category.

Result
Error Code

36.1.21 SysTimerStop

RTS_RESULT SysTimerStop (RTS_HANDLE hTimer)

This function stops a timer

hTimer [IN]
Handle of the timer

Result
Error Code

36.1.22 SysTimerGetInterval

*RTS_RESULT SysTimerGetInterval (RTS_HANDLE hTimer, RTS_SYSTIME *ptIntervalNs)*

Returns the interval of a timer

hTimer [IN]
Handle of the timer
ptIntervalNs [OUT]
Pointer to Interval of the timer in nanoseconds
Result
Error code

36.1.23 SysTimerSetInterval

RTS_RESULT SysTimerSetInterval (RTS_HANDLE hTimer, RTS_SYSTIME tIntervalNs)
This function set the interval of a timer
hTimer [IN]
Handle of the timer
tIntervalNs [IN]
Interval of the timer in nanoseconds to set
Result
Error code

36.1.24 SysTimerGetTimeStamp

*RTS_RESULT SysTimerGetTimeStamp (RTS_HANDLE hTimer, RTS_SYSTIME *ptTimestampNs)*
This function returns the timestamp in ticks since timer start
hTimer [IN]
Handle of the timer
ptTimestampNs [OUT]
Timestamp in nanoseconds
Result
Error code

36.1.25 SysTimerFitTimer

*unsigned int SysTimerFitTimer (RTS_HANDLE hTimer, unsigned long ulPriority, RTS_SYSTIME tInterval, RTS_RESULT *pResult)*
This function checks a given timer
hTimer [IN]
Timer to check
ulPriority [IN]
Priority to check
tInterval [IN]
Interval to check
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Returns if Timer fits

36.1.26 SysTimerGetMinResolution

*RTS_RESULT SysTimerGetMinResolution (RTS_HANDLE hTimer, RTS_SYSTIME *ptMinResolutionNs)*
Get the minimum resolution of the timer
hTimer [IN]
Handle of the timer
ptMinResolutionNs [OUT]
Minimum timer resolution in nanoseconds
Result
Error code

36.1.27 SysTimerGetContext

*RTS_RESULT SysTimerGetContext (RTS_HANDLE hTimer, RegContext *pContext)*
This function returns the context of the current timer handling
hTimer [IN]
Handle of the Timer

pContext [OUT]
Pointer to Timer Context
Result
error code

36.1.28 SysTimerSetCallbackParameter

RTS_RESULT SysTimerSetCallbackParameter (RTS_HANDLE hTimer, RTS_HANDLE hParam)
Sets the Callback Parameter
hTimer [IN]
Handle of the timer
hParam [IN]
Parameter for callback routine
Result
Error code

36.1.29 SysTimerSetResetFollowing

RTS_RESULT SysTimerSetResetFollowing (RTS_HANDLE hTimer)
With this function the specified function is reseted (Needed for CmpScheduleTimer)
hTimer [IN]
Handle of the timer to reset
Result
Error code

36.1.30 SysTimerExistsTimer

RTS_HANDLE SysTimerExistsTimer (unsigned long ulPriority, RTS_SYSTIME tInterval, RTS_RESULT pResult)*
This function looks for timers with the given properties
ulPriority [IN]
Priority of the timer
tInterval [IN]
Interval of the timer
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Handle to the searched timer or RTS_INVALID_HANDLE

36.1.31 SysTimerGenerateException

RTS_RESULT SysTimerGenerateException (RTS_HANDLE hTimerHandle, RTS_UI32 ulException, RegContext Context)
Calls the corresponding exception handler of the timer.
hTimerHandle [IN]
Handle to timer
ulException [IN]
Rts standard exception
Context [IN]
Context to detect the code location where the exception occurred
Result
ERR_OK

36.1.32 SysTimerMaxTimer

*RTS_RESULT SysTimerMaxTimer (RTS_UI32 *pulMaxTimers)*
This function returns the maximal number of timers
pulMaxTimer [OUT]
Number of Timers
Result
Error code

36.1.33 SysTimerRegisterBasePointer

RTS_RESULT SysTimerRegisterBasePointer (RTS_UINTPTR ulBP)
Register the base pointer of the main routine.
pulBP [IN]
base pointer of main
Result
ERR_OK

36.1.34 SysTimerGetFirst

*RTS_HANDLE SysTimerGetFirst (RTS_RESULT *pResult)*
Get first registered timer object
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Pointer to first timer object

36.1.35 SysTimerGetNext

*RTS_HANDLE SysTimerGetNext (SYS_TIMER_INFO *pTimerPrev, RTS_RESULT *pResult)*
Get next registered timer object
pTimerPrev [IN]
Pointer to previous timer object
pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Pointer to next timer object

36.1.36 SysTimerCallCallback

*void SysTimerCallCallback (SYS_TIMER_INFO *pTimer)*
Call the registered callback handler with the corresponding hParam. IEC- and C-handlers are supported.
pTimer [IN]
Secified timer object
Result

36.1.37 SysTimerGetCurrent

*RTS_HANDLE SysTimerGetCurrent (RTS_RESULT *pResult)*

Returns the handle of the currently active timer, or RTS_INVALID_HANDLE if we are outside of a timer context.

For SIL2 systems, this function is not implemented and returns always RTS_INVALID_HANDLE.

pResult [OUT]
Result pointer containing the error code. Might be NULL.
Result
Handle of the currently active timer or RTS_INVALID_HANDLE

36.1.38 SysTimerDelete

RTS_RESULT SysTimerDelete (RTS_HANDLE hTimer)
This function deletes a timer
hTimer [IN]
Handle of the timer
Result
Error code