The Library SysLibPLCConfig.lib

This library supports the reading of the configuration data of the PLC Configuration. These data are also loaded to the controller at a download of the application and are written to structures by the runtime system. The library offers functions for getting pointers on these structures.

Due to the fact that pointers on original structures of the runtime system are provided, the following is:

- The structure (pointer to sub-elements) may not be modified !
- If default values of parameters in the structures get modified, this will be of on effect !

If supported by the target system, the following library functions can be used (Execution is synchronous):

- CfgCCGetError
- CfgCCGetHeader
- CfgCCGetRootModule
- CfgCCGetRootModuleByModuleId
- CfgCCGetRootModuleByNodeId

CfgCCGetError

Note: Currently not yet implemented in the runtime system. Error code always 0.

This function provides information on the errors which occur during the download of the configuration.

The function returns a pointer to structure CCLoadError.

Structure CCLoadError:

```
TYPE CCLoadError :
STRUCT
    ulLastError: UDINT;          (* Error code of the last error *)
    ulAddInfo1: UDINT;           (* According to ulLastError, the meaning changes.*)
    ulAddInfo2: UDINT;           (* According to ulLastError, the meaning changes. *)
    szLastError: STRING(32);     (* A possibility to make debugging easier. *)
END_STRUCT
END_TYPE
```

CfgCCGetHeader

This function returns a pointer on the header structure of the PLC configuration CCHeader.

Structure CCHeader:

```
TYPE CCHeader :
STRUCT
    szTag: STRING(10); (*Contains the zero terminated STRING "CommConf"*)
    cByteOrder: BYTE; (* The file data is in intel format ('I') OR motorola format ('M')*)
    ulSize: UDINT; (* Size of the following data *)
    lVersion: UDINT; (*Version number of the file*)
END_STRUCT
END_TYPE
```

CfgCCGetRootModule

This function provides information on the root module of the PLC configuration. It returns a pointer to structure CCModule.

Structure CCModule:

```
TYPE CCModule :
STRUCT
    ucEntryTag: BYTE;           (* 'M' = Module*)
    ucDummy1: BYTE;
    ucDummy2: BYTE;
    ucDummy3: BYTE;
    ulModuleId: UDINT;          (* Id of the module given in the configuration file *.cfg *)
    sModuleNumber: UINT;        (* Number of the module in the parent module (-1 if root) *)
    usModuleTag: UINT;          (* Describes the kind of the module ( 0=3S-Module, 1=DP-Master,
                                2=DP-Slave, 3=CAN-Master, 4=CAN-Slave, 5=DP-SingleSlave) *)
    byDeviceDriver: BYTE;       (* The module needs a device driver (0=FALSE, 1=TRUE)*)
    ucDummy4: BYTE;
    ucDummy5: BYTE;
    ucDummy6: BYTE;
    ulNodeId: UDINT;            (* NodeId of the module*)
```

```
    byDefinedWithStruct: BYTE;          (* The module was defined with a structure (0=FALSE, 1=TRUE) *)
    ucDummy7: BYTE;
    ucDummy8: BYTE;
    ucDummy9: BYTE;
    ulBitOffsetInput: UDINT;            (* Offset of the modules input area *)
    ulBitSizeInput: UDINT;              (* Size of the modules input area in bit *)
    ulBitOffsetOutput: UDINT;           (* Offset of the modules output area *)
    ulBitSizeOutput: UDINT;             (* Size of the modules output area in bit *)
    ulRefIdCommonDiag: UDINT;           (* RefId of the modules common diagnosis area *)
    ulBitOffsetCommonDiag:              (* Offset of the modules common diagnosis area *)
    UDINT;
    ulBitSizeDiag: UDINT;               (* Size of the modules diagnosis area in bit *)
    usParameterCount: UINT;             (* Number of parameters *)
    usDummy: UINT;
    ppccpModuleParams:                  (* <ccParam [0..usParameterCount]> a pointer to an array of pointers to
    POINTER TO POINTER TO               CCModuleParam-structures. (Definition of structure CCParam see
    ccParam;                            below). Dereferencing the pointer with ppccpModuleParams^ gives you
                                        the pointer to the first parameter structure. (ppccpModuleParams+4)^
                                        gives you the pointer to the next parameter structure. See also
                                        comment (*Read pointer to parameters *) in example project. *)
    ulSizeOfSpecificData: UDINT;        (* Size in bytes of the module specific data *)
    pModuleData: POINTER TO             (*<MODULE_SPECIFIC_DATA> Here the data, according to
    BYTE;                               usModuleTag is located: pModuleData is possible to be a pointer to
                                        PBSlave, CANSlave, PBMaster, PBSlave, PBSingleSlave, see
                                        definitions below.*)
    usChannelCount: UINT;               (* Number of configured channels *)
    usModuleCount: UINT;                (* Number of configured modules *)
(* In the following the Channels and Modules of this Module in the configured order are located! (DP-Slaves are
ordered by the stationnumber!) This means, it is possible that another CCModule structure is inserted here.*)
    ppcccChannels: POINTER              (* <ccChannel [0..usChannelCount]> Definition of structure CCChannel
    TO POINTER TO                       see below * Dereferencing the pointer with ppccpChannels^ gives you
    ccChannel;                          the pointer to the first parameter structure. (ppccpChannels+4)^ gives
                                        you the pointer to the next parameter structure. See also comment
                                        "(*Read pointer to parameters *)" in example project. *)
```

```
    ppccmSubModules:
    POINTER TO POINTER    TO
    BYTE;                 (* <ccModule [0..usModuleCount]> Points to an array of variables of type POINTER TO ccModule. To view the contents, you have to assign the value to a variable of type "POINTER TO CCModule". Definition of structure CCModule see below. Dereferencing the pointer with ppccpSubModules^ gives you the pointer to the first parameter structure. (ppccpSubModules+4)^ gives you the pointer to the next parameter structure. See also comment "(*Read pointer to parameters *)" in example project. *)

END_STRUCT
END_TYPE
```

Structure CCChannel:

```
TYPE CCChannel :
STRUCT
    ucEntryTag: BYTE;             (* 'C' = Channel *)
    ulChannelId: UDINT;           (* Id of the channel given in the configuration file *)
    usChannelNumber: UINT;        (* Number of the channel in the parent module *)
    ulRefId: UDINT;               (* Direction of the channel (1=input, 2=output, 3=input AND output) *)
    usChannelType: UINT;          (* TYPE of the channel (coded as CoDeSys "TypeClass") *)
    ulBitOffset: UDINT;           (* Offset of the channel in in-/output area *)
    usParameterCount: UINT;       (* Number of parameters *)
    ppccpParams: POINTER  TO
    POINTER TO CCParam;           (* PARAMETER[1..usParameterCount]> POINTER TO an ARRAY OF pointers TO CCParam-structures. (Definition of structure CCParam see below) *)
END_STRUCT
END_TYPE
```

Struktur CCParam:

```
TYPE CCParam :
STRUCT
    ulParameterId: UDINT;         (* Id of the parameter given in the configuration file *.cfg *)
    usParameterNumber: UINT;      (* Number of the parameter in the module *)
    byReadOnly: BYTE;             (* 1=TRUE, 0=FALSE *)
```

```
    byDummy: BYTE;
    usParameterType: UINT;      (* Type of the parameter; CoDeSys "TypeClass" *)
    usDummy: UINT;
    ulSize: UDINT;              (* Size of the parameter in bytes *)
    byValue: BYTE;              (* The memory representation of the parameter value
                                   starts with this byte. The other bytes follow immediately, if
                                   the size of the parameter value is bigger than 1. *)
END_STRUCT
END_TYPE
```

CfgCCGetRootModuleByModuleId

This function provides information on the root module of the currently used PLC configuration, which is given by the module Id. The module Id is defined in the configuration file by entry "Id", see document PLC_Configuration_E.pdf.

The function returns a pointer to structure CCModule (see above, function CfgCCGetRootModule)

Input Variable	Data type	Description
ulModuleId	UDINT	Module Id of the root module

CfgCCGetRootModuleByNodeId

This function provides information on the root module of the currently used PLC configuration, which is given by the node Id. The node Id of the module normally results from the position of the module within the PLC Configuration. For details see document PLC_Configuration_E.pdf. The function returns a pointer to structure CCModule (see above, function CfgCCGetRootModule)

Input Variable	Data type	Description
ulNodeId	UDINT	Node Id of the root module