ECKELMANN

E·CNC55

MMICTRL.DLL

Description of Function

Author: Name

File name: MMICTRL.doc

Version: 1.7 First edition: 05/10/2002

Release: February 2006

Modification protocol

Chapter	Date	Person in charge	Modification	Release Date / Initials
2.2.16	05/23/02	TE	Change of the result type from short to BOOL	
2.2-2.4	06/05/02	TE	Extension	
All	06/26/02	TE	Transfer from MMIDEL32.DOC	
2.1.17	03/12/03	TE	New function ncrSendMessage	
2.2, 2.3	02/13/03	TE	New callback types, definitions of task_uc	
2.1.9, 2.1.11	02/21/03	TE	Correction of the return value description	
2.3	02/24/03	TE	Update of error messages of GTW and DLL	
2.1.22-2.1.36	03/14/03	GM	Description of the service channel	
2.1.1, 2.3.1	04/08/03	TE	Extension to optional controller names	
2.1.7, 2.3.2	06/03/03	TE	New function ncrConnect, 3 new error messages	
all	02/06	WN	Formal corrections	

Table of contents

1 Introduction 1

2 Description of function 2
2.1 Functions of the MMICTRL.DLL 2
2.1.1 ncrOpenControl 2
2.1.2 ncrOpenDefaultControl 3
2.1.3 ncrGetDprAddress 4
2.1.4 ncrGetInitState 5
2.1.5 ncrLoadFirmware 6
2.1.6 ncrLoadFirmwareBlocked 7
2.1.7 ncrConnect 8
2.1.8 ncrSendFile 9
2.1.9 ncrSendFileEx 10
2.1.10 ncrSendFileBlocked 11
2.1.11 ncrReceiveFile 12
2.1.12 ncrReceiveFileBlocked 13
2.1.13 ncrWaitTransfer 14
2.1.14 ncrCloseTransfer 15
2.1.15 ncrGetTransferType 15
2.1.16 ncrGetTransferState 16
2.1.17 ncrPostMessage 17
2.1.18 ncrSendMessage 18
2.1.19 ncrReadParamArray 19
2.1.20 ncrWriteParamArray 20
2.1.21 ncrCloseControl 21
2.1.22 ncrCloseAllControls 21
2.1.23 gtwOpenServiceChannel 22
2.1.24 gtwCloseServiceChannel 23
2.1.25 gtwGetNumOfConns 23
2.1.26 gtwGetConnInfo 24
2.1.27 gtwSetConnInfo 25
2.1.28 gtwAddConn 26

2.1.29 gtwDelConn 27
2.1.30 gtwGetConnState 28
2.1.31 gtwGetConnDetails 29
2.1.32 gtwGetGtwVer 30
2.1.33 gtwTraceState 31
2.1.34 gtwTraceEnable 32
2.1.35 gtwTraceDisable 32
2.1.36 gtwGetConnExt 33
2.1.37 gtwSetConnExt 34
2.2 Callback function 35
2.3 Return codes and error messages 37
2.3.1 Error messages of the IPCOM.DLL (1.0.x) 41
2.3.2 Error messages of the MMIGTWAY.EXE (1.1.x) 41
2.3.3 Error messages of the MMICTRL.DLL (1.2.x) 44
2.3.4 Error messages of the CNCDPR55.EXE (1.17.x) 45
2.4 Definitions and structures 47
2.4.1 com_1st.h 47
2.4.2 com_def.h 51
2.4.3 com_sbx.h 58
2.4.4 mmictrl.h 61

1 Introduction

As successor device of MMIDEL32.DLL, MMICTRL.DLL is available as official interface between the new HMI applications and the Eckelmann CNC controllers E·CNC55, E·PNC55 and E·ENC55. Unlike the MMIDEL32.DLL, this DLL also supports multi-user and multi-control applications. These are applications with several HMI applications or several controllers on a single PC.

For these applications, the MMICTRL.DLL provides a completely new designed interface. Although the new DLL continues to support previous HMI applications via an adapted MMIDEL32.DLL, new HMI applications should only use this interface in order to be prepared for new extensions.

An extension in the CNC firmware is required for the coupling of several applications to a CNC controller. As far as the controllers E·CNC55, E·PNC55 and E·ENC55 are concerned, this extension is integrated starting with firmware version V1.42. For the time being, such an extension is not intended for the controllers CNC20 and PNC20 of the previous generation.

The MMIDEL32.DLL continues to be the interface to old applications. Starting with version 3.0, the MMICTRL.DLL will be needed in addition.

2 Description of function

2.1 Functions of the MMICTRL.DLL

2.1.1 ncrOpenControl

This function provides a handle for the access to a controller.

```
HANDLE ncrOpenControl (
    char *pzName,                 // Name of the controller
    MSGCALLBACK pCallback,        // Pointer to a callback function
    void *pContext                // Self-pointer of the calling instance
);
```

	Meaning	
Parameters	*pzName*	Zero-terminated string with the name of the controller to which a connection is to be made. The name of the device can have up to 30 characters, include the letters A -Z, the numbers 0-9, as well as blanks, underscores and hyphens. The assigning of device names to controllers is made by means of the file MMIGTWAY.INI or the configuration tool GtwConf.
	pCallback	Pointer to a callback function, for the evaluation of NC messages, error and state messages. For further information about the interface of this callback function please check section 2.2.
	pContext	Pointer to the context of the callback function. This is normally the self-pointer of the calling instance. At each call, this pointer is transferred to the callback function.
Return value	Upon successful implementation of the function, the latter provides the handle of the required controller. In the case of an error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides more detailed information about the cause of the error. See section 2.3.	
Description	The function ncrOpenControl makes the connection to a controller and provides the handle for all subsequent accesses to this controller. The function ncrCloseControl is used for a closing of the connection and the enabling of the handle. At a closing of the DLL, all enabled handles are disabled automatically.	
See also	ncrOpenDefaultControl, ncrCloseControl, ncrGetDprAddress, ncrPostMessage, ncrLoadFirmware, ncrSendFile, ncrReceiveFile.	

2.1.2 ncrOpenDefaultControl

This function provides a handle for the default controller of the current application.

HANDLE ncrOpenDefaultControl (
    MSGCALLBACK *pCallback*, // Pointer to a callback function
    void **pContext* // Self-pointer of the calling instance
);

	Meaning	
Parameters	*pCallback*	Pointer to a callback function, for the evaluation of NC messages, error and state messages. For further information about the interface of this callback function please check section 2.2.
	pContext	Pointer to the context of the callback function. This is normally the self-pointer of the calling instance. At each call, this pointer is transferred to the callback function.
Return value	Upon successful implementation of the function, the latter provides the handle of the required controller. In the case of an error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides more detailed information about the cause of the error. See section 2.3.	
Description	The function ncrOpenControl makes the connection to a controller that is assigned to the application to be called as default controller and provides the handle for all subsequent accesses to this controller. The definition of the name of this controller is made in file MMICTRL.INI in the current directory of the application, i.e. in section *devices* under the keyword *default*. Example: [devices] default=PNC0 The function ncrCloseControl is used for a closing of the connection and the enabling of the handle. At a closing of the DLL, all enabled handles are disabled automatically.	
See also	ncrOpenControl, ncrCloseControl, ncrGetDprAddress, ncrPostMessage, ncrLoadFirmware, ncrSendFile, ncrReceiveFile.	

2.1.3 ncrGetDprAddress

The function provides a pointer to the dual-port RAM of the specified controller.

void *ncrGetDprAddress (
HANDLE *hCnc*, // Handle of the controller
BOOL *bShowErrors* // Display error messages
);

	Meaning	
Parameters	*hCnc*	Handle of the controller, the dual-port RAM of which is required.
	bShowErrors	If this variable is TRUE, errors are displayed in a message box upon request of the DPR address.
Return value	Upon successful implementation of the function, the latter provides a handle to the dual-port RAM area of the controller. In case of an error, the return value is ZERO.	
Description	Function ncrGetDprAddress determines the virtual address of the dual-port RAM area of the specified controller. To that purpose, the MMICTRL.DLL needs to have direct access to the respective CNCDPR driver. This is only possible, if the calling application and the gateway server run on the same PC.	
See also	ncrOpenControl, ncrOpenDefaultControl.	

2.1.4 ncrGetInitState

With this function, the application is able to determine whether the specified controller has already loaded its firmware.

LONG ncrGetInitState (
HANDLE *hCnc* // Handle of the controller
);

	Meaning	
Parameters	*hCnc*	Handle of the controller, the state of which is required.
Return value	The function provides 0 if the firmware is not yet included in the controller, and 1 if the firmware is already loaded into the controller. In the case of error, the function provides -1.	
Description	The function ncrGetInitState determines whether the specified controller was already initialized and booted. If not, function ncrLoadFirmware is to be implemented before a transfer of messages to this controller is possible.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrLoadFirmware.	

2.1.5 ncrLoadFirmware

Asynchronous function for the loading of the firmware and the PLC program into the specified controller.

HANDLE ncrLoadFirmware (
    HANDLE *hCnc*, // Handle of the controller
    **char ***pzConfig* // Name of the configuration file for the firmware download
);

	Meaning	
Parameters	*hCnc*	Handle of the controller that is to be booted.
	pzConfig	Zero-terminated string with the name of the configuration file for the firmware download.
Return value	Upon successful implementation of the function, the latter provides a handle for the functions ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer and ncrCloseTransfer. In the case of error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides more detailed information about the cause of error. See section 2.3.	
Description	The function ncrLoadFirmware starts the firmware download to the specified controller. Since the implementation is made asynchronously, the function provides only a transfer handle that is used to determine the current transfer state. With each change of the transfer state, the application is informed by the assigned callback function. **VK_MMI_DOWNLOAD_STATE** and the respective transfer handle are transferred. The end of the transfer is signaled by a call of the callback function with the identification **VK_MMI_DOWNLOAD_PART** (firmware was already loaded into the controller), **VK_MMI_DOWNLOAD_COMPLETE** or **VK_MMI_DOWNLOAD_ERROR** and the respective transfer handle. Subsequently, the handle is to be disabled again by means of ncrCloseTransfer.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrLoadFirmwareBlocked, ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer, ncrCloseTransfer.	

2.1.6 ncrLoadFirmwareBlocked

Blocking function for the loading of the firmware and the PLC program into the specified controller.

LONG ncrLoadFirmwareBlocked (
HANDLE *hCnc*, // Handle of the controller
char * *pzConfig* // Name of the configuration file for firmware download
);

	Meaning	
Parameters	*hCnc*	Handle of the controller that is to be booted.
	pzConfig	Zero-terminated string with the name of the configuration file for the firmware download.
Return value	The function provides 0, if the firmware download was implemented successfully and -1, if the firmware of the controller was already loaded. In the case of error, the return value is greater than 0 and includes an error number that can also be prompted with GetLastError. See section 2.3.	
Description	The function ncrLoadFirmwareBlocked implements a firmware download into the specified controller. During the implementation of the function, the application is informed by the assigned callback function about the current transfer state. **VK_MMI_DOWNLOAD_STATE** and an internal transfer handle are transferred. This handle can be used within the callback function for the determination of the transfer state with ncrGetTransferState.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrLoadFirmware, ncrGetTransferType, ncrGetTransferState.	

2.1.7 ncrConnect

Establishment of a connection to a controller with already loaded firmware for passive applications such as bus server and CodeSys.

BOOL ncrConnect (
    HANDLE *hCnc* // Handle of the controller
);

	Meaning	
Parameters	*hCnc*	Handle of the controller to which a connection is to be established.
Return value	The function provides TRUE if the connection to the controller could be established successfully and if no firmware download is successful. The function provides FALSE if no connection could be established to the controller, e.g. because a firmware download is necessary before. The detailed error cause can be called with GetLastError. See section 2.3.	
Description	The function ncrConnect tries to initialize the interface to the specified controller without making a download of the firmware or the PLC program. This can only be done if the firmware is already included in the controller. This is a blocking function, i.e. it returns only to the calling application if the operation is terminated.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrLoadFirmware, ncrLoadFirmwareBlocked.	

2.1.8 ncrSendFile

Asynchronous function for the transfer of files into the controller, such as machine constants or DIN programs.

```
HANDLE ncrSendFile (
    HANDLE hCnc,                // Handle of the controller
    char *pzFilename,           // Name of the file to be transferred
    char *pzHeader,             // Header string for program number in case of DIN programs
    int iSb1                    // Control block 1 for block transfer
);
```

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the file is to be transferred.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	pzHeader	Zero-terminated string that is to be inserted at the beginning of the transfer.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	Upon successful implementation of the function, the latter provides a handle for the functions ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer and ncrCloseTransfer. In the case of error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides detailed information about the cause of error. See section 2.3.	
Description	The function ncrSendFile starts a block transfer to the specified controller. Since the implementation is made asynchronously, it only provides a transfer handle that is able to determine the current transfer state. With each change of the transfer state, the application is informed by the assigned callback function. **VK_MMI_TRANSFER_STATE** and the respective transfer handle are transferred. End of transfer is signaled by a call of the callback function with the identification **VK_MMI_TRANSFER_OK**, **VK_MMI_TRANSFER_ERROR** or **VK_MMI_TRANSFER_BREAK** and the respective transfer handle. Subsequently, the handle is to be disabled with ncrCloseTransfer.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrSendFileEx, ncrSendFileBlocked, ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer, ncrCloseTransfer.	

2.1.9 ncrSendFileEx

Extended asynchronous function for the transfer of files, mainly online programs, into the controller, starting with a determined file offset.

```
HANDLE ncrSendFileEx (
    HANDLE hCnc,                // Handle of the controller
    char *pzFilename,           // Name of the file to be transferred
    DWORD dwOffsetLow,          // Less significant 32-bit of the file offset
    DWORD dwOffsetHigh,         // More significant 32-bit of the file offset
    char *pzHeader,             // Header string for program number in case of DIN programs
    int iSb1                    // Control block 1 for block transfer
);
```

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the file is to be transferred.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	dwOffsetLow	Less significant 32-bit of the file offset, from which the transfer is to be started.
	dwOffsetHigh	More significant 32-bit of the file offset, from which the transfer is to be started.
	pzHeader	Zero-terminated string that is to be inserted at the beginning of the transfer.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	Upon successful implementation of the function, it provides a handle for the functions ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer and ncrCloseTransfer. In the case of error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides detailed information about the cause of error. See section 2.3.	
Description	The function ncrSendFile starts a block transfer to the specified controller. Starting point of the file to be transferred is the specified file position. Since the implementation is made asynchronously, it only provides a transfer handle that is able to determine the current transfer state. With each change of the transfer state, the application is informed by the assigned callback function. The **VK_MMI_TRANSFER_STATE** and the respective transfer handle are transferred. The end of transfer is signaled by a call of the callback function with the identification **VK_MMI_TRANSFER_OK**, **VK_MMI_TRANSFER_ERROR** or **VK_MMI_TRANSFER_BREAK** and the respective transfer handle. Subsequently, the handle is to be disabled with ncrCloseTransfer	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrSendFile, ncrSendFileBlocked, ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer, ncrCloseTransfer.	

2.1.10 ncrSendFileBlocked

Blocking function for the transfer of files into the controller, such as machine constants or DIN programs.

LONG ncrSendFileBlocked (
HANDLE *hCnc*, // Handle of the controller
char * *pzFilename*, // Name of the file to be transferred
char * *pzHeader*, // Header string for program number in case of DIN programs
int *iSb1* // Control block 1 for block transfer
);

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the file is to be transferred.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	pzHeader	Zero-terminated string that is to be inserted at the beginning of the transfer.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	The function returns the final transfer state. The state is 0, if the file was transferred successfully. Value 2 signals an error that can be prompted by means of GetLastError. See section 2.3. 0 - TRANSFER_OK Transfer was completed successfully 1 - TRANSFER_BREAK Transfer aborted by the HMI or NC 2 - TRANSFER_ERROR Access error occurred, request with GetLastError() 3 - TRANSFER_TIMEOUT Timeout in case of message transfer to the controller 4 - TRANSFER_NOINIT Firmware download was not yet implemented 5 - TRANSFER_QFULL Too many transfer orders active at the same time	
Description	The function ncrSendFileBlocked transfers a file to the specified controller. During implementation of the function, the application is informed by the assigned callback function about the current transfer state. **VK_MMI_TRANSFER_STATE** and an internal transfer handle are transferred. This handle can be used within the callback function for the determination of the transfer state with ncrGetTransferState.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrSendFile, ncrSendFileEx, ncrGetTransferType, ncrGetTransferState.	

2.1.11 ncrReceiveFile

Asynchronous function for the transfer of files into the controller, such as machine constants or DIN programs.

HANDLE ncrReceiveFile (
HANDLE *hCnc*, // Handle of the controller
char **pzFilename*, // Name of the file to be transferred
int *iSb1* // Control block 1 for block transfer
);

	Meaning	
Parameters	*hCnc*	Handle of the controller from which the file is to be read.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	Upon successful implementation of the function, the latter provides a handle for the functions ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer and ncrCloseTransfer. In the case of error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides detailed information about the cause of error. See section 2.3.	
Description	The function ncrReceiveFile requires a block transfer from the specified controller. Since the implementation is made asynchronously, it only provides a transfer handle that is able to determine the current transfer state. With each change of the transfer state, the application is informed by the assigned callback function. The **VK_MMI_TRANSFER_STATE** and the respective transfer handle are transferred. End of transfer is signaled by a call of the callback function with the identification **VK_MMI_TRANSFER_OK**, **VK_MMI_TRANSFER_ERROR** or **VK_MMI_TRANSFER_BREAK** and the respective transfer handle. Subsequently, the handle is to be disabled with ncrCloseTransfer.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrReceiveFileBlocked, ncrGetTransferType, ncrGetTransferState, ncrWaitTransfer, ncrCloseTransfer.	

2.1.12 ncrReceiveFileBlocked

Blocking function for the transfer of files from the controller to the PC, such as machine constants or DIN programs.

LONG ncrReceiveFileBlocked (
HANDLE *hCnc*, // Handle of the controller
char **pzFilename*, // Name of the file to be transferred
int *iSb1* // Control block 1 for block transfer
);

	Meaning	
Parameters	*hCnc*	Handle of the controller from which the file is to be read.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	The function returns the final transfer state. The state is 0, if the file was transferred successfully. Value 2 signals an error that can be prompted by means of GetLastError. See section 2.3. 0 - TRANSFER_OK Transfer was completed successfully 1 - TRANSFER_BREAK Transfer aborted by the HMI or NC 2 - TRANSFER_ERROR Access error occurred, request with GetLastError() 3 - TRANSFER_TIMEOUT Timeout in case of message transfer to the controller 4 - TRANSFER_NOINIT Firmware download was not yet implemented 5 - TRANSFER_QFULL Too many transfer orders active at the same time	
Description	The function ncrSendFileBlocked transfers a file from the specified controller to the PC. During implementation of the function, the application is informed by the assigned callback function about the current transfer state. **VK_MMI_TRANSFER_STATE** and an internal transfer handle are transferred. This handle can be used within the callback function for the determination of the transfer state with ncrGetTransferState	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrReceiveFile, ncrGetTransferType, ncrGetTransferState.	

2.1.13 ncrWaitTransfer

The function waits for the termination of the specified file transfer and returns the transfer state.

LONG ncrWaitTransfer (
HANDLE *hTrans* // Handle of data transfer
);

	Meaning	
Parameters	*hTrans*	Handle of the transfer, the end of which is to be expected.
	pzFilename	Zero-terminated string with the name of the file to be transferred.
	iSb1	Type of file to be transferred. Corresponds to control block 1 of block transfer.
Return value	The function provides 0 if the file was transferred successfully. In case of an error, the return value is not equal 0 and includes an error number. See section 2.3.	
Description	The function ncrWaitTransfer waits until the specified firmware or file transfer is terminated. During implementation of the function, the application is informed by the assigned callback function about the current transfer state. **VK_MMI_TRANSFER_STATE** and an internal transfer handle are transferred. This handle can be used within the callback function for the determination of the transfer state with ncrGetTransferState *Since the callback function in this case is called by another thread, no SendMessage must be made in the callback function to the Main Thread if the latter is already blocked by ncrWaitTransfer. Otherwise a Deadlock is caused.* Also after ncrWaitTransfer, the transfer handle is to be disabled with ncrCloseTransfer.	
See also	ncrLoadFirmware, ncrSendFile, ncrSendFileEx, ncrReceiveFile, ncrCloseTransfer	

2.1.14 ncrCloseTransfer

The function terminates the specified file transfer and re-enables the respective file transfer handle.

BOOL ncrCloseTransfer (
HANDLE *hTrans* // Handle of the file transfer
);

	Meaning	
Parameters	*hTrans*	Transfer handle that is to be terminated.
Return value	The function provides TRUE if the handle could be terminated successfully. The function provides FALSE if an invalid transfer handle was transferred.	
Description	The function ncrCloseTransfer terminates the specified firmware or file transfer and enables the handle for the next file transfer. If the respective transfer should not be terminated at that time, it is aborted on the next occasion.	
See also	ncrLoadFirmware, ncrSendFile, ncrSendFileEx, ncrReceiveFile, ncrWaitTransfer	

2.1.15 ncrGetTransferType

Provides the type of the specified file transfer.

short ncrGetTransferType (
HANDLE *hTrans* // Handle of the file transfer
);

	Meaning	
Parameters	*hTrans*	Handle of the transfer, the type of which is to be determined.
Return value	The function provides the control block 1 (iSb1) of the block transfer.	
Description	The function ncrGetTransferType provides the type of the specified file transfer. In case of a firmware transfer, the value 0xff is provided.	
See also	ncrLoadFirmware, ncrLoadFirmwareBlocked, ncrSendFile, ncrSendFileEx, ncrSendFileBlocked, ncrReceiveFile, ncrReceiveFileBlocked, ncrGetTransferState.	

2.1.16 ncrGetTransferState

The function provides the transfer state of the specified file transfer.

LONG ncrGetTransferStatus (
    HANDLE *hTrans,* // Handle of the file transfer
    TRANSFER_STATE_TR * *prState* // Pointer to a data structure for the transfer state
);

	Meaning	
Parameters	*hTrans*	Handle of the transfer, the state of which is to be determined.

	prState	Pointer to the data structure in which the state information is to be input.
		typedef struct { // Transfer state for new applications */
		DWORD *dwFilePosLow*; // Current write/read position
		DWORD *dwFilePosHigh*;
		DWORD *dwFileSizeLow*; // File size
		DWORD *dwFileSizeHigh*;
		TCHAR *azName*[MAX_PATH];
		// Complete name of the file to be transferred
		TCHAR *azMeld*[256];
		// State information for firmware download
		} **TRANSFER_STATE_TR**;

Return value	In case of a firmware download, the function provides 0 upon successful completion of the download, the function provides -1 if the firmware was already loaded, and -2, if the transfer is still in progress. In case of a file transfer, the function provides 0 upon successful completion of the transfer, the function provides -1 in case of Online Wait, and -2 if the transfer is still in progress. The function provides a value great than 0 if an error occurred during transfer. See section 2.3.	
Description	The function ncrGetTransferState fills the transferred transfer state structure and returns the current state of the transfer.	
See also	ncrLoadFirmware, ncrLoadFirmwareBlocked, ncrSendFile, ncrSendFileEx, ncrSendFileBlocked, ncrReceiveFile, ncrReceiveFileBlocked, ncrGetTransferType.	

2.1.17 ncrPostMessage

This function enters the specified message into the transmitter queue of the specified controller.

```
BOOL ncrPostMessage (
    HANDLE hCnc,              // Handle of the controller
    MSG_TR *prMsg,            // Pointer to the message to be transferred
    int iRespKey              // Response key
);
```

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the message is to be transferred.

	prMsg	Pointer to the message to be transferred.
		typedef struct { // Transfer state for new applications */
		uchar sb0_uc; // Control block 0
		uchar sb2_uc; // Control block 2 upon request
		uchar sb1_uc; // Control block 1
		uchar index_uc; // Is looped through in case of requests
		uchar modul_uc;
		// Message transmitter normally SYS_MMI_KC (15)
		uchar handle_uc; // Reserved for Gateway
		ushort len_us; // Length of the following data in bytes
		... // Used data, see MMINC.DOC
		} **MSG_TR**;

	iRespKey	If this value is unequal 0, the application is informed after the transfer of the message by the assigned callback function. **VK_MMI_NCMSG_SENT** is transferred in *ulType* if the message could be transferred or **VK_MMI_NCMSG_NOT_SENT** if the message could not be transferred due to a timeout.
Return value	The function provides TRUE if the message could be entered into the transmitter queue. The function provides FALSE in case of an error. The error cause can be requested with **GetLastError**. See section 2.3.	
Description	The function ncrPostMessage includes a message into the transmitter queue of the specified controller and returns immediately. The message is not synchronized with current block transfers. It is only safeguarded that several messages included with ncrPostMessage are transmitted to the controller in the cal sequence. In cases that require a synchronization with block transfers or signals of the virtual keyboard, synchronization is to be made by means of *iRespKey* via the callback function.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrSendMessage	

2.1.18 ncrSendMessage

The function includes the specified message into the transfer queue of the specified controller and waits until the message has been transferred to the controller.

```
BOOL ncrSendMessage (
    HANDLE hCnc,                    // Handle of the controller
    MSG_TR *prMsg                   // Pointer to the message to be transferred
);
```

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the message is to be transferred.

	prMsg	Pointer to the message to be transferred.
		typedef struct { // Transfer state for new applications */
		uchar sb0_uc; // Control block 0
		uchar sb2_uc; // Control block 2 upon request
		uchar sb1_uc; // Control block 1
		uchar index_uc; // Is looped through in case of requests
		uchar modul_uc;
		// Message transmitter normally SYS_MMI_KC (15)
		uchar handle_uc; // Reserved for Gateway
		ushort len_us; // Length of the following data in bytes
		... // Used data, see MMINC.DOC
		} **MSG_TR**;

Return value	The function provides TRUE, if the message could be transferred to the controller. The function provides FALSE in case of an error. The error cause can be requested with **GetLastError**. See section 2.3.	
Description	The function ncrSendMessage includes a message into the transfer queue of the specified controller and returns only if the message was actually transferred to the controller or if the timeout time has elapsed.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrPostMessage	

2.1.19 ncrReadParamArray

This function reads the values of the specified P-field indices from the parameter field of the controller.

BOOL ncrReadParamArray (
HANDLE *hCnc*, // Handle of the controller
WORD * *pwIndex*, // Pointer to array with parameter field indices
double * *pdValue,* // Pointer to array for the storing of the parameter field values
WORD *wCount* // Number of parameter field values to be read
);

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the message is to be transferred.
	pwIndex	Pointer to an array of type **WORD** with the indices of the parameter field values to be read
	pdValue	Pointer to an array of type **double** for the storing of the read parameter field values.
	wCount	Number of inputs in *pwIndex* and *pdValue*.
Return value	The function provides TRUE if the function could be implemented successfully. The function provides FALSE in case of an error. The error cause can be requested with **GetLastError**. See section 2.3.	
Description	With the function ncrReadParamArray, up to 2048 parameter field values can be read with a function call. The calling thread is blocked until all parameter have been read or the timeout time has elapsed.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrWriteParamArray	

2.1.20 ncrWriteParamArray

This function writes the specified P-field values into the parameter field of the controller.

BOOL ncrReadParamArray (
HANDLE *hCnc*, // Handle of the controller
WORD * *pwIndex*, // Pointer to array with parameter field indices
double * *pdValue,* // Pointer to array with the parameter field values
WORD *wCount* // Number of parameter field values to be written
);

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the message is to be transferred.
	pwIndex	Pointer to an array of type **WORD** with the indices of the parameter field values to be read
	pdValue	Pointer to an array of type **double** for the storing of the read parameter field values.
	wCount	Number of inputs in *pwIndex* and *pdValue*.
Return value	The function provides TRUE if the function could be implemented successfully. The function provides FALSE in case of an error. The error cause can be requested with **GetLastError**. See section 2.3.	
Description	With the function ncrWriteParamArray, up to 2048 parameter field values can be written with a function call. The calling thread is blocked until all parameter have been written or the timeout time has elapsed.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrReadParamArray	

2.1.21 ncrCloseControl

This function disconnects the connection to the specified controller.

void ncrCloseControl (
HANDLE *hCnc* // Handle of the controller
);

	Meaning	
Parameters	*hCnc*	Handle of the controller to which the message is to be transferred.
Return value	-	
Description	The function ncrCloseControl disconnects the connection to the specified controller and re-enables the respective handle. The gateway is also closed with the disconnection of the last connection.	
See also	ncrOpenControl, ncrOpenDefaultControl	

2.1.22 ncrCloseAllControls

This function disconnects the connection to all controllers.

void ncrCloseControl (void);

	Meaning
Parameters	-
Return value	-
Description	The function ncrCloseAllControls disconnects all connections to all controllers connected to the applications and re-enables the respective handles. The gateway is also closed with the disconnection of the last connection.
See also	ncrOpenControl, ncrOpenDefaultControl

2.1.23 gtwOpenServiceChannel

This function provides a handle for the access to the gateway configuration.

HANDLE gtwOpenServiceChannel (
char **pzName* // Name of the gateway

);

	Meaning	
Parameters	*pzName*	Zero-terminated string with the name of the gateway to which a connection is to be made. Presently, only an empty string "" is supported as gateway name, since only the gateway on the local PC can be configured.
Return value	The function provides the handle of the gateway upon its successful implementation. In the case of error, the return value is **INVALID_HANDLE_VALUE**. In this case, the function **GetLastError** provides further information about the error cause. See section 2.3.	
Description	The function gtwOpenServiceChannel makes the connection to a gateway and provides the handle for all subsequent accesses to the gateway configuration. The function gtwCloseServiceChannel is used for disconnecting the connection and for the enabling of the handle. All open handles are closed automatically at a closing of the DLL.	
See also	gtwGetNumOfConns, gtwGetConnInfo, gtwCloseServiceChannel.	

2.1.24 gtwCloseServiceChannel

This function disconnects the connection to the specified gateway.

void gtwCloseServiceChannel (
    HANDLE *hGtwHndl* // Handle of the gateways
);

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
Return value	-	
Description	The function gtwCloseServiceChannel disconnects the connection to the gateway and re-enables the respective handle. The gateway is also closed with the disconnection of the last connection.	
See also	GtwOpenServiceChannel	

.

2.1.25 gtwGetNumOfConns

This function provides the number of configured connections.

Int gtwGetNumOfConns (
    HANDLE *hGtwHndl,* // Handle of the gateway
    int **piNumOfConns* // Pointer to a 32 bit variable
);

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	piNumOfConns	Pointer to the variable in which the number of configured connections is to be included.
Return value	The function provides 0 if the variable *iNumOfConns* has a valid value or –1 if an error occurred during the access to the gateway. In this case, the function GetLastError provides more detailed information about the cause of error.	
Description	The function ncrGetNumOfConns provides the number of connections available in the configuration of the gateway.	
See also	gtwGetConnInfo, GtwGetConnState.	

2.1.26 gtwGetConnInfo

This function provides the communication parameters of the specified connection.

```
int gtwGetConnInfo (
    HANDLE          hGtwHndl,       // Handle of the gateway
    CONN_INFO_TR *prConnInfo        // Pointer to the data structure of the
                                    // communication parameters
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.

	prConnInfo	Pointer to a data structure in which the communication parameters are to be included.
		typedef struct {
		long lIndex; // Index in the internal gateway array
		long lType; // Driver type
		char azName[32]; // Name of the controller
		char azParam[216]; // Parameter, e.g. the IP address
		} CONN_INFO_TR;

Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetConnInfo provides the communication parameters of the specified connection. When calling the function, the index of the connection in the range 0..MAX_IPC_CHANNELS is to be transferred into the data structure. Upon configuration of the connection, the function returns the type, the name and the communication parameter. The following driver types are defined: DRIVER_VXD_VERSION 1 - Windows95/98 VxD driver for DPR cards DRIVER_WDM_VERSION2 - Windows98/2000/XP WDM driver for PNC55. DRIVER_NT_VERSION 3 - Windows NT driver for PNC55. DRIVER_UDP_VERSION 4 - UDP/IP driver for CNC55e/ENC55. DRIVER_DEMO_VERSION 5 - Simulation of the controller The following parameters are implemented: „PCI Card“ - for DPR cards and PNC55 controllers. „<IP Adresse>“ - for CNC55e/ENC55. „Simulation“ - for simulated connections.	
See also	gtwAddConn , gtwSetConnInfo.	

2.1.27 gtwSetConnInfo

This function changes the communication parameters of the specified connection.

int gtwGetConnInfo (
HANDLE *hGtwHndl,* // Handle of the gateway
CONN_INFO_TR * *prConnInfo* // Pointer to the data structure of the
// communication parameter
);

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that includes the communication parameters.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwSetConnInfo changes the communication parameters of the specified connections. Please observe that only the IP addresses of the UDP connections can be changed. The data structure prConnInfo is to be returned by the gateway at the call of the function gtwGetConnInfo.	
See also	gtwGetConnInfo, gtwAddConn.	

2.1.28 gtwAddConn

This function adds a new connection of the gateway configuration.

```
int gtwAddConn (
    HANDLE          hGtwHndl,     // Handle of the gateway
    CONN_INFO_TR    *prConnInfo   // Pointer to the data structure of the
                                  // communication parameter
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that includes the communication parameters.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	New connections can be defined with the function gtwAddConn. When calling the function, the name and the communication parameter of the connection must be available in the data structure *prConnInfo*. Upon successful implementation of the function, the index of the new connection is returned to the data structure.	
See also	gtwGetConnInfo, gtwSetConnInfo, gtwDelConn.	

2.1.29 gtwDelConn

This function deletes the specified connections from the gateway connection.

```
int gtwDelConn (
    HANDLE          hGtwHndl,     // Handle of the gateway
    CONN_INFO_TR    *prConnInfo   // Pointer to the data structure of the
                                  // communication parameter
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that includes the communication parameters.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	Function gtwDelConn deletes the connection specified in the data structure *prConnInfo* from the gateway configuration. Only the UDP connections can be deleted.	
See also	gtwAddConn, gtwGetConnInfo, gtwSetConnInfo.	

2.1.30 gtwGetConnState

This function provides the state of the specified connection.

```
int gtwGetConnState (
    HANDLE          hGtwHndl,     // Handle of the gateway
    CONN_INFO_TR    *prConnInfo,  // Pointer to the communication parameters
    int             *piState      // Pointer to a variable of the state
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that characterizes the connection.
	piState	Pointer to a variable in which the state of the communication is to be included.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetConnState provides the value of the counter that is incremented during the message transfer. In this way, the application is able to safeguard that the communication is made with the specified controller.	
See also	gtwGetConnDetails	

.

2.1.31 gtwGetConnDetails

This function provides detailed information about the specified connection.

```
int gtwGetConnDetails (
    HANDLE          hGtwHndl,     // Handle of the gateway
    CONN_INFO_TR    *prConnInfo,  // Pointer to the communication parameters
    CONN_DETAILS_TR *prDetails    // Pointer to the data structure of the
                                  // communication information
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that characterizes the connection.

	prDetails	Pointer to a data structure in which the communication information is to be input.
		
		typedef struct {
		ulong rc_cnt_ul; // received messages
		ulong tr_cnt_ul; // transferred messages
		ushort mmi2t_order_us; //
		ushort t2mmi_quitt_us; //
		ushort t2mmi_status_us; //
		ushort mmi2t_quitt_us; //
		ushort msg2nc_r_qc_us; //
		ushort msg2nc_r_mc1_us; //
		ushort msg2nc_r_sb0_us; //
		ushort msg2nc_r_sb1_us; //
		ushort msg2nc_r_mc2_us; //
		ushort nc2t_order_us; //
		ushort t2nc_quitt_us; //
		ushort t2nc_status_us; //
		ushort nc2t_quitt_us; //
		ushort msg2mmi_r_qc_us; //
		ushort msg2mmi_r_mc1_us; //
		ushort msg2mmi_r_sb0_us; //
		ushort msg2mmi_r_sb1_us; //
		ushort msg2mmi_r_mc2_us; //
		} CONN_DETAILS_TR;

Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetConnExt provides detailed information about the state of communication. The data are read from the DPR.	
See also	gtwGetConnState	

.

2.1.32 gtwGetGtwVer

This function provides the version of the gateway.

```
int gtwGetGtwVer (
    HANDLE   hGtwHndl,    // Handle of the gateway
    CHAR     *pcBuffer,   // Pointer to a buffer for version information
    INT      iBufLen      // Length of the buffer
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	pcBuffer	Pointer to a buffer for version information.
	iBufLen	Length of the buffer.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetGtwVer returns the version of the gateway in the text format. In this way, versions of the installed components can be checked in the application.	
See also	-	

2.1.33 gtwTraceState

This function provides the state of the specified trace.

```
int gtwTraceState (
    HANDLE   hGtwHndl,    // Handle of the gateway.
    int      iTrNum,      // Index of the trace.
    int      *piState     // Pointer to a variable of the state.
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	iTrNum	Index of the trace.
	piState	Pointer to a variable in which a trace is to be included.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	Function gtwTraceState returns the state of the specified trace. The following trace numbers are defined: 20 OnAddConn 30 OnChCreate 31 OnConnect 32 OnRead 33 OnWrite 34 OnWrCh 50 OnLoadFw 70 OnFileOpen 71 OnFileClose 96 OnMsg2Nc 97 OnMsg2Mmi 255 OnError	
See also	GtwTraceEnable, gtwTraceDisable.	

2.1.34 gtwTraceEnable

This function enables the specified trace.

int gtwTraceEnable (
HANDLE *hGtwHndl,* // Handle of the gateway
INT *iTrNum* // Index of the trace

);

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	iTrNum	Index of the trace.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	Function gtwTraceEnable enables the specified trace.	
See also	GtwTraceState, gtwTraceDisable.	

2.1.35 gtwTraceDisable

This function disables the specified trace.

int gtwTraceDisable (
HANDLE *hGtwHndl,* // Handle of the gateway
INT *iTrNum* // Index of the trace

);

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	iTrNum	Index of the trace.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	Function gtwTraceDisable disables the specified trace	
See also	GtwTraceState, gtwTraceEnable	

.

2.1.36 gtwGetConnExt

This function provides the extended communication parameters of the specified connection.

```
int gtwGetConnExt (
    HANDLE          hGtwHndl,   // Handle of the gateway
    CONN_INFO_TR    *prConnInfo // Pointer to communication parameters
    CONN_EXT_TR     *prConnExt  // Pointer to a data structure of extended
                                // parameters
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that characterizes the connection.

	prConnExt	Pointer to a data structure in which the extended parameters are to be input.
		
		typedef struct CONN_EXTENSION_R {
		int iExtType; // Type of parameter
		union nExtData {
		char azPcDir[128]; // Path of PC disc
		….
		}
		} CONN_EXTENSION_TR;

Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetConnExt provides additional parameters of the specified connection. When calling the function, the type of parameter must be transferred into the data structure. Upon successful implementation of the function, the current value of the specified parameter is returned in the data structure *prConnExt*.	
See also	gtwSetConnExt	

2.1.37 gtwSetConnExt

This function changes extended communication parameters of the specified connection.

```
int gtwGetConnExt (
    HANDLE          hGtwHndl,     // Handle of the gateway
    CONN_INFO_TR    *prConnInfo   // Pointer to the communication parameter
    CONN_EXT_TR     *prConnExt    // Pointer to a data structure of
                                  // extended parameters
);
```

	Meaning	
Parameters	*hGtwHndl*	Handle of the gateway.
	prConnInfo	Pointer to a data structure that characterizes the connection.
	prConnExt	Pointer to a data structure in which the extended parameter is input.
Return value	0 if the function could be implemented successfully, -1 in the case of error. The function **GetLastError** provides more detailed information about the cause of error.	
Description	The function gtwGetConnExt changes extended parameters of the specified connection.	
See also	gtwGetConnExt	

.

2.2 Callback function

This function is called by the MMICTRL.DLL to inform the application about changes of transfer state, error messages or messages of the controller.

void WINAPI MsgCallback (
ULONG *ulType*, // Cause of the call
ULONG *ulParam*, // Additional parameters depending on the value of *ulType*
void * *pContext* // of the context pointers transferred with ncrOpenControl
);

	Meaning	
Parameters	*ulType*	This function informs the application about the cause of a call of the callback function.
	ulParam	The meaning of this parameter depends on the value of *ulType*. See above.
	pContext	This parameter includes the value that was specified with ncrOpenControl or ncrOpenDefaultControl. The application can use this parameter in order to differentiate different instances of the same method.
Return value	-	
Description	The callback function is always requested by the MMICTRL.DLL if the transfer state of a file or firmware transfer in progress changes or if an error message is signaled by the controller, the gateway or the DLL. The function is also requested if a message is received by the controller that cannot be processed by the DLL but is to be evaluated by the application.	
See also	ncrOpenControl, ncrOpenDefaultControl, ncrPostMessage	

Messages of *ulType:*

VK_MMI_DOWNLOAD_STATE
VK_MMI_DOWNLOAD_PART
VK_MMI_DOWNLOAD_COMPLETE
VK_MMI_DOWNLOAD_ERROR

These messages are state messages during the firmware transfer to the controller. The parameter *ulParam* includes the transfer handle of the respective transfer. The functions ncrGetTransferType and ncrGetTransferState can be used within the callback function to call the transfer state.

VK_MMI_DOWNLOAD_COMPLETE is signaled upon successful transfer of the firmware to the controller. **VK_MMI_DOWNLOAD_PART** is signaled if the controller was already booted and no firmware transfer was, therefore, necessary and **VK_MMI_DOWNLOAD_ERROR** is signaled if an error occurred during the firmware transfer.

VK_MMI_TRANSFER_STATE
VK_MMI_TRANSFER_OK
VK_MMI_TRANSFER_ERROR
VK_MMI_TRANSFER_BREAK

These messages are state messages of a data transfer in progress between the PC and the controller. The parameter *ulParam* includes the transfer handle of the respective transfer. The functions ncrGetTransferType and ncrGetTransferState can be used within the callback function to request the transfer state.

VK_MMI_TRANSFER_OK is signaled upon successful completion of the transfer. **VK_MMI_TRANSFER_ERROR** is signaled if an error occurred during the file transfer and **VK_MMI_TRANSFER_BREAK** is signaled if the transfer has been aborted prematurely for a another reason.

VK_MMI_NCMSG_SENT
VK_MMI_NCMSG_NOT_SENT

If a value unequal 0 was specified in *iRespKey* with ncrPostMessage, the transfer of the respective message to the controller with **VK_MMI_NCMSG_SENT** is signaled to the application. In case of **VK_MMI_NCMSG_NOT_SENT**, the message could not be transferred to the controller due to a timeout. The parameter *ulParam* includes the value of *iRespKey* that was transferred with ncrPostMessage.

VK_MMI_NCMSG_RECEIVED

This message signals the reception of a new message from the controller to the application. In this way, only messages are signaled that cannot be evaluated by the MMICTRL.DLL. The parameter *ulParam* includes the pointer to a structure of type MSG_TR with the respective message. The pointer is only valid within the callback function since the memory of the DLL is again enabled at the end of the callback function.

VK_MMI_ERROR_MSG

This message is used for the transfer of error messages from the controller, the gateway or the DLL to the application. The parameter *ulParam* includes the pointer to a structure of type MSG_TR with the respective error message. The pointer is valid only within the callback function, since the memory of the DLL is again enabled at the end of the callback function. If *modul_uc* is unequal 0, the error message comes from the controller, otherwise the error message was generated by the PC.

Error messages of class 3 and 4 of the controller are to be acknowledged by the application, in order to return the controller to the state ready-for-operation. A message of type **SB1_FEHLERQUITTUNG_KUC** or **SB1_FEHLERQUITTUNG_ALLE_KUC** is to be transferred to the controller.

VK_MMI_CYCLIC_CALL

Cyclic call of the callback function every 100ms, when blocking MMICTRL.DLL functions are used.

2.3 Return codes and error messages

Error messages are generated in various points of the PC and in the controller and are signaled via the callback function (see section 2.2) to the application. Each error message is unambiguously identified by means of the computer identification, the sub-system and the error number. The seriousness of the errors is specified in error classes. Five different error classes have been established:

1 F_KLAS_LOKAL_LEICHT_KUC

Errors of this class have an effect only on the sub-system that transferred the error message. The action causing the warning is continued without acknowledgment.

2 F_KLAS_LOKAL_SCHWER_KUC

Errors of this class have an effect only on the sub-system that transferred the error message. The action causing the error message is aborted.

3 F_KLAS_GLOBAL_LEICHT_KUC

Errors of this class have an effect on the overall system. A program probably in progress is interrupted. The controller passes into the error state and waits for the acknowledgment of the error by the HMI application.

4 F_KLAS_GLOBAL_SCHWER_KUC

Errors of this class have an effect on the overall system. A program probably in progress is aborted. The controller passes into the error state and waits for the acknowledgment of the error by the HMI application.

5 F_KLAS_FATAL_KUC

Errors of this class have an effect on the overall system. A program probably in progress is aborted. After this error, the controller is to be reset and restarted.

For further information about the single elements of the error message and the acknowledgment message, an extract of the message structure of section 2.4.2 follows:

```
/*-------------------------------------------------------------------------*/
typedef swapped struct MSG_TR {  /* Nachrichtenstruktur für alle Nachrichten des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             sb0_uc;                    /* Steuerblock 0 */
   uchar             sb2_uc;                    /* Steuerblock 2 bei Bedarf */
   uchar             sb1_uc;                    /* Steuerblock 1 */
   uchar             index_uc;                  /* Wird bei Anfragen durchgeschleift */
   uchar             modul_uc;                  /* Nachrichtensender */
   uchar             handle_uc;                 /* Wird bei Anfragen durchgeschleift */
   ushort            len_us;                    /* Länge */
   swapped union {
      ...
      ERR_MELD_TR       err_meld_r;             /* Fehlermeldung */
      ERR_QUITT_TR      err_quitt_r;            /* Fehlerquittung */
      ...
   }                 n;
}  MSG_TR;
```

```
/*-------------------------------------------------------------------------*/
typedef swapped struct ERR_MELD_TR {  /* Struktur für Fehlermeldungen innerhalb des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             task_uc;               /* fehlermeldende Funktion */
   uchar             klasse_uc;             /* Fehlergewichtung */
   short             nummer_s;              /* Fehlernummer */
   char              info_format_ac [32];   /* Nur Format (vgl.printf, "%d%l"), keine Texte */
   char              info_data_ac   [80];   /* Nutzdaten der Zusatzinformation */
} ERR_MELD_TR;

/*-------------------------------------------------------------------------*/
typedef struct ERR_QUITT_TR { /* Struktur für Fehlermeldungsquittung innerhalb des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             task_uc;          /* fehlermeldende Funktion */
   uchar             klasse_uc;        /* Fehlergewichtung */
} ERR_QUITT_TR;
```

Explanation

sb0_uc includes the value SB0_EXCEPTION_KUC in case of error messages and error acknowledgments

sb1_uc includes the value SB1_FEHLERNUMMER_KUC in case of error messages and SB1_FEHLERQUITTUNG_KUC or SB1_FEHLERQUITTUNG_ALLE_KUC in case of error acknowledgments

modul_uc in case of error messages of the controller, includes the sub-system that signaled the error and is in this case always unequal 0. In case of error messages generated by the operating computer, *modul_uc* is always 0 and is, therefore, to be used as computer identification

len_us is 116 (size of ERR_MELD_TR) in case of error messages and 2 (size of ERR_QUITT_TR) in case of acknowledgment messages

task_uc includes the number of the sub-system in which the error occurred. Even in case of error messages of the controller, it needs not to be identical with *modul_uc*, since error recognition and error message are not always made in the same sub-system.

klasse_uc includes the error class, see above.

nummer_s includes the error number and is used together with *task_uc* for an assignment of the error text to the error message.

info_format_ac includes a printf-compatible format string. This string is normally used together with *info_data_ac* for the formatting of the error-specific additional information for the display. In case of error messages from the CNC55, the PNC55 and the ENC55, the additional information is already provided as formatted string. In this case, *info_format_ac* and *info_data_ac* form a block string.

info_data_ac includes a printf-compatible list of arguments for the error-specific additional information.

All other elements of the message structure are irrelevant for error messages and error acknowledgments.

Possible error-signaling modules in *task_uc* for error messages of the HMI

0	IPC	- IPCOM.DLL
1	GTW	- MMIGTWAY.EXE
2	DLL	- MMICTRL.DLL
11	WIN	- Error messages of Windows-API functions
15	HMI	- HMI application
17	COM	- CNCDPR55.EXE or ET-PC01

Possible error-signaling modules in *task_uc* for error messages of the NCR

1	AXE	- Axis computer
3	INT	- Interpreter
4	KOP	- Link or Ethernet coupling
5	PLC	- PLC runtime system
6	POS	- Interpolator
7	SPV	- Memory management
9	ZST	- Central control
10	SPS	- PLC program
11	SYS	- Operating system
12	CAN	- CAN bus communication
13	UTI	- Auxiliary functions
18	DRV	- DS402 drive control device
19	LNZ	- Lenze drive control device
20	NOV	- Novotron drive control device
21	SDL	- Seidel drive control device
22	IDR	- Indramat drive control device

In the case of error, some functions of the MMICTRL.DLL provide a return code that includes an error number; in case of other functions, the error number is to be determined by means of the Windows™ function **GetLastError**. The following two cases are distinguished:

1. Windows™ error codes

In case of Windows™ error codes, bit 29 is not set. A complete list of Windows™ error codes is included in Windows™ Help. Example:

14L ERROR_OUTOFMEMORY

110L ERROR_OPEN_FAILED

112L ERROR_DISK_FULL

233L ERROR_PIPE_NOT_CONNECTED

2. Application error codes

In case of these error code, bit 29 is always set. This includes all error codes that are generated in IPCOM.DLL, MMICTRL.DLL, MMIGTWAY.EXE or CNCDPR55.EXE. The following coding is used with all application error codes:

Bit 23-16 Includes the sub-system that has recognized the error:

- 0 IPCOM.DLL
- 1 MMIGTWAY.EXE
- 2 MMICTRL.DLL
- 15 HMI application
- 17 CNCDPR55.EXE or ET-PC01

Bit 15-0 Includes the error number of the respective sub-system.

Assignment of error texts to error messages

Error messages are most reasonably structured into the following ranges: controller (NCR), PLC and control panel (HMI). Therefore, the structuring has been made in the following files:

NCR_FEHL.DB includes the error texts of all error messages generated in the controller firmware and is prepared by Eckelmann.

SPS_FEHL.DB includes the error texts of all error messages generated in the PLC and in the DIN program (G253) and is prepared by the PLC programmer.

MMI_FEHL.DB includes the error texts of all error messages generated in the operating computer and is partly prepared by Eckelmann. Additional error texts can be added by the control panel programmer.

Each line of these files includes three strings, each separated by a comma. The first string includes the computer identification, the sub-system number and the error number, each separated by a point. The second string includes the respective error text and the third string is reserved. Below is an example of an error message from the controller, in this case from the sub-system 3, the interpreter:

"2.3.1502","G2/G3: Keine Ebene programmiert",""

In these files, the error number is always represented as decimal value without sign. In case of negative field numbers, this value always results from the error number +65536. Example for error -98:

"2.11.65438","RAM-Disk: Batterie ist leer, bitte wechseln",""

2.3.1 Error messages of the IPCOM.DLL (1.0.x)

	Lfd. Nr.	Fehlertext deutsch	Fault text english
1.0	1	IPCOM.DLL: nicht genügend ipc Speicher	IPCOM.DLL: not enough ipc memory
	2	IPCOM.DLL: kein gültiger ipc Speicher	IPCOM.DLL: no valid ipc memory
	3	IPCOM.DLL: kein freier ipc Block	IPCOM.DLL: no free ipc block
	4	IPCOM.DLL: ipc Block defekt	IPCOM.DLL: damaged ipc memory block
	5	IPCOM.DLL: Semaphore Zugriff fehlgeschlagen	IPCOM.DLL: access to semophore failed
	6	IPCOM.DLL: Semaphore Freigabe fehlgeschlagen	IPCOM.DLL: release of Semaphore failed
	7		
	8		
	9		
	10	IPCOM.DLL: keine solche Instanz	IPCOM.DLL: no such instance
	11	IPCOM.DLL: kein solches Handle	IPCOM.DLL: no such handle
	12	IPCOM.DLL: kein freies Handle	IPCOM.DLL: no free handle
	13	IPCOM.DLL: Spion bereits aktiviert	IPCOM.DLL: an spy is already active at this Instance
	14	IPCOM.DLL: zu viele Schreibzugriffe offen	IPCOM.DLL: to many write pending
	15	IPCOM.DLL: ungültiger Zeiger	IPCOM.DLL: invalid pointer
	16	IPCOM.DLL: ungültige Speichergröße	IPCOM.DLL: invalid memory size
	17	IPCOM.DLL: ungültiger IntHandle	IPCOM.DLL: invalid inthandle

2.3.2 Error messages of the MMIGTWAY.EXE (1.1.x)

	Lfd. Nr.	Fehlertext deutsch	Fault text english
	1		
	2	Download: Fehler beim Öffnen einer Datei	Download: File Open Error
	3	Download: Fehler beim Lesen einer Datei	Download: Read File Error
	4		
	5		
	6	Download: Kommunikationsfehler	Download: CommunicationError
	7	Download: Timeout im DPR-Kommunikationsbereich	Download: Timeout in DPR-Communication
	8	Download: Falscher Steuerblock 0 in Downloadnachricht von NCR	Download: Wrong sb0 in Downloadmessage from NCR
	9	Download: Ladefehler beim Laden der SPS, möglicher Dateifehler	Download: PLC StoringError, maybe file damaged
	10		
	11	Download: Ein nicht konfiguriertes Gerät wurde gefunden	Download: found not configured device
	12	Download: Ein konfiguriertes Gerät wurde nicht gefunden	Download: configured device not found

	13	Download: Ladefehler beim Laden der Firmware, möglicher Dateifehler	Download: Firmware LoadError, maybe file damaged
	14	Download: Fehler beim DPR-Test, Dual-Port-RAM ist defekt	Download: DPR-Test Error, Dual-Port-RAM is damaged
	15	Download: Fehler beim Laden oder Starten der Firmware	Download: Firmware Load or Start Error
	16	Download: Gerät meldet Reset, Download muss erneut gestartet werden	Download: Device sends reset, new Download
	17	Download: Kommunikationsfehler	Download: CommunicationError
	18	Download: Speichermangel beim Einlesen der Konfigurationsdatei	Download: not enough memory for reading configuration file
	19	Download: zu viele Geräte an einem Transputer-Link	Download: too many devices for Transputer-Link
	20	Download: Ungültige Primäridentifikationskennung empfangen	Download: received unknown primary ID
	21	Download: Unbekannten Fehlercode über DPR-Fehlerpuffer empfangen	Download: received unknown Errorcode from DPR-Errorbuffer
	22	Download: Syntax-Fehler in der Konfigurationsdatei	Download: Syntax-Error in Configurationfile
	23	Download: Fehler beim Beschreiben des Dual-Port-RAM	Download: Error writing Dual-Port-RAMs
	24	Download: Baugruppe ist bereits geladen	Download: Device already active
	25	ncrConnect: Firmware-Download erforderlich	ncrConnect: Firmware-Download neccessary
	26	ncrConnect: Steuerung antwortet nicht	ncrConnect: no answer from NC

	Lfd. Nr.	**Fehlertext deutsch**	**Fault text english**
1.1	101	Steuerung noch nicht initialisiert, Nachricht nicht gesendet	NC not initialized, message not sent
	102	MMI->NC Nachrichtenpuffer-Timeout	MMI->NC messagebuffer-Timeout
	103	Die Steuerung wurde resetet	NC reseted

x.y	Lfd. Nr.	Fehlertext deutsch	Fault text english
1.1	201	Timeout bei Remote-Dateizugriff	Timeout Remote-Fileaccess
	202	Ungültiges IO-Kommando bei Remote-Dateizugriff empfangen	received unknown IO-Command at Remote-Fileaccess
	203	Unerlaubter Zugriffsmode bei FileOpen	FileOpen: unkown accessmode
	204	Event für FileOpen konnte nicht erzeugt werden	FileOpen: Event not created
	205	Timeout bei Remote-Dateizugriff	FileOpen: no more File-Handle
	206	Datei wurde nicht gefunden	unknown File
	207	Datei konnte nicht angelegt werden	File not created
	208	IO-Fehler beim Lesen aus der Datei	IO-Error: File read
	209	IO-Fehler beim Schreiben in die Datei	IO-Error: File write
	210	IO-Fehler beim Positionieren des Schreib-/Lesezeigers in der Datei	IO-Error: File position
	211	IO-Fehler beim Schließen der Datei	IO-Error: File close
	212	Ungültiges File-Handle in Funktion FileClose	unknown File-Handle FileClose
	213	Ungültiges File-Handle in Funktion FileRead	unknown File-Handle FileRead
	214	Ungültiges File-Handle in Funktion FileWrite	unknown File-Handle FileWrite
	215	Ungültiges File-Handle in Funktion FileSeek	unknown File-Handle FileSeek
	216	Ungültiges File-Handle in Funktion FilePushString	unknown File-Handle FilePushString
	217	Ungültiges File-Handle in Funktion FileGetPos	unknown File-Handle FileGetPos
	218	Ungültiges File-Handle in Funktion FileGetSize	unknown File-Handle FileGetSize
	219	Datei zum Schreiben geöffnet bei FilePushString	FilePushString: File opened for writing
	220	String länger als Schreib-/Lesepuffer bei FilePushString	FilePushString: String too long

	Lfd. Nr.	Fehlertext deutsch	Fault text english
1.1	301	Nachrichtenpuffer-Timeout bei Dateiübertragung	File-Transfer: Timeout
	302	File-Transfer: Fehler bei Schließen der Datei	File-Transfer: Close Error
	303	File-Transfer: Fehler bei Lesen aus der Datei	File-Transfer: Read Error
	304	File-Transfer: Fehler beim Schreiben in die Datei	File-Transfer: Write Error
	305	File-Transfer: Fehler beim Öffnen der Datei	File-Transfer: Open Error

	Lfd. Nr.	Fehlertext deutsch	Fault text english
1.1	401	Fehler bei Pufferallocierung	Buffer allocation error
	402	Fehler: falsche Anforderung	Invalid request
	403	Fehler: falscher Index	Invalid connection index
	404	Fehler: Verbindung gelöscht	Connection already removed
	405	Fehler: falsche Verbindung	Invalid connection index
	406	PCI - Parameter kann nicht geändert werden	Can not change PCI connection parameters
	407	Fehler beim Öffnen einer neuen Verbindung	Can not add new connection
	408	PCI - Verbindung kann nicht gelöscht werden	Can not remove a PCI connection
	409	Kein freies UDP-Handle	No free UDP handles
	410	Fehler beim UDP FileMap	Udp FileMap error
	411	Fehler beim UDP MapView	Udp MapView error
	412	Fehler: Udp SockVer	Udp SockVer error
	413	Fehler beim Udp Socket	Udp Socket error
	414	Fehler beim SetSockOpt	Udp SetSockOpt error
	415	Fehler beim Starten des UDP Threads	Udp Thread error

2.3.3 Error messages of the MMICTRL.DLL (1.2.x)

	Lfd. Nr.	Fehlertext deutsch	Fault text english
	101	Ungültiges Handle für eine Steuerung	unknown Handle for NC
	102	Die angegebene Steuerung konnte nicht geöffnet werden	can not open unknown NC
	103	Unbekannter DPR-Treiber bei ncrGetDprAddress	ncrGetDprAddress: unknown DPR-Driver
	104	ncrGetDprAddress: DPR-Zugriff wird bei Remote-Steuerungen nicht unterstützt	ncrGetDprAddress: no DPR-Access with Remote-NC
	105	Win32GetDprPointer: Konnte DPR-Beschreibung nicht aus der Treiberinfo extrahieren	Win32GetDprPointer: unkowm DPR-Description
	201	Zu viele Dateiübertragungen gleichzeitig angestoßen	too many Filetransfers
	202	Ungültiges Handle für Dateitransfer	unknown Handle for Filetransfer
	203	Timeout bei Dateiübertragung (ncrWaitTransfer)	Timeout Filetransfer (ncrWaitTransfer)
	301	Zu viele Statusanfragen gleichzeitig angestoßen	too many Statusrequests
	302	Ungültiges Handle für Statusanfrage	unknown Handle for Statusrequest
	401	Timeout bei ncrWriteParamList	Timeout ncrWriteParamList
	402	Fehlerhaftes Argument bei ncrWriteParamList	wrong Argument ncrWriteParamList
	403	Timeout bei ncrReadParamList	Timeout ncrReadParamList
	404	Fehlerhaftes Argument bei ncrReadParamList	wrong Argument ncrReadParamList

	501	Ungültige Dateinummer bei Sb0FileOpen	unknown Fileno. Sb0FileOpen
	502	Ungültige Dateinummer bei Sb0FileClose	unknown Fileno. Sb0FileClose
	503	Ungültige Dateinummer bei Sb0FileRead	unknown Fileno. Sb0FileRead
	504	Ungültige Dateinummer bei Sb0FileWrite	unknown Fileno. Sb0FileWrite
	505	Ungültige Dateinummer bei Sb0FileSeek	unknown Fileno. Sb0FileSeek
	601	Timeout beim Warten auf Antwort von MMIGTWAY.EXE	Timeout wait for MMIGTWAY.EXE

2.3.4 Error messages of the CNCDPR55.EXE (1.17.x)

	Lfd. Nr.	**Fehlertext deutsch**	**Fault text english**
1.17	1	Timeout beim Senden	Send Timeout
	2		
	3	Timeout beim Warten auf die Quittung	Wait for Accept Timeout
	4		
	5		
	6		
	7		
	8		
	9	Timeout bei Trace-Übergabe mittels Interrupt	trace interrupt transfer Timeout
	10		
	11	Keine Applikation geladen	No Application loaded
	12		
	13	Nicht genug Speicher für Download	not enough memory for download
	14		
	15		
	16		
	17		
	18		
	19		
	20	Fehler im dynamischen RAM	Error in dynamic RAM
	21	Startupfehler in der Firmware	Startuperror in Firmware
	22	Timeout bei der Nachrichtenübergabe über DPR-Nachrichtenpuffer	Transfer Timeout DPR-Messagebuffer
	23	CPU-Typ der Firmware stimmt nicht mit der Steuerung überein	NC and CPU-Typ of Firmware are different

	Lfd. Nr.	Fehlertext deutsch	Fault text english
1.17	220		
	221		
	222		
	223		
	224		
	225		
	226		
	227	Fehler in der Dateistruktur, Entrypoint außerhalb des Codes	File Error, Entrypoint out of code
	228	Static-Bedarf zu groß	too much Static-memory needed
	229	Vectorspace nicht erlaubt	Vectorspace not allowed
	230	Workspace-Bedarf zu groß	too much Workspace needed
	231	Falscher Prozessortyp	wrong Processortype
	232	Falsche Compiler-Version	wrong Compiler-Version
	233	Fehler in der Dateistruktur	Error Filestructur
	234	Zu wenig Speicher zum Laden des Codes	not enough memory for loading
	235	Fehler in der Dateistruktur, mehr Code als erwartet	Error Filestructur, more Code then needed
	236	Fehlerhafte Code-Tabelle in komprimierter Firmware	wrong Code-table in zipped Firmware
	237		
	238		
	239		
	240		
	241		
	242		
	243	Checksummenfehler in der Firmware	Checksum Error in Firmware
	244	Firmware im Testmode geladen und kann daher nicht gestartet werden	Firmware load in Testmode, no start possible
	245	Firmware ungültig oder korrupt	unknown Firmware
	246	Firmware enthält keine Versionsinformation	no Version in Firmware
	247	Firmware nicht vollständig	Firmware damaged
	248	Firmware inkompatibel, Sektionen sind nicht aufsteigend sortiert	Firmware damaged: sections not sorted
	249	Firmware inkompatibel und verletzt geschützte Speicherbereiche	Firmware damaged: wrong access to memory
	250	Nicht genug Speicher zum Laden der Firmware	not enough memory for loading Firmware
	251	Firmware ist keine ausführbare Datei	Firmware is not executable
	252	Firmware-Dateiformat ist unbekannt	unknown FirmwareFile
	253	Firmware konnte nicht entpackt werden, Zielpuffer zu klein	Unzip Firmware not possible, buffer too small
	254	Fehlerhafte Code-Tabelle in komprimierter Firmware	wrong Code-table Firmware

2.4 Definitions and structures

2.4.1 com_1st.h

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ACH@-*/
/*
*/
/*    LastUpdate   COM_1ST.H   04-05-1992 09:14  R1.0   by Ebert   */
/*
**
**    PROJECT:   CNC21-Steuerung
**
**    VERSION:   V1.0                 $Revision:   1.5  $
**
**    MODULE DESCRIPTION:   com_1st.h    ( HEADER (DEF) )
**    ---------------------------------------------------------------
**
**    Diese Datei enthält projektunabhängige Definitionen und sollte vor
**    allen projektspezifischen Headerdateien eingebunden werden.
**
**
**
**
**
**    (C) ECKELMANN INDUSTRIEAUTOMATION, WIESBADEN
**    ---------------------------------------------------------------
**    CREATED: 01-08-91 20:02   AUTHOR: FISCHER    V: V1.0
**-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ECH@-*/


/* $Header:   P:\cnc21\include\vcs\com_1st.h_v   1.5   20 Apr 2001 11:10:02
ebert_0050046BD496  $ */


/*<f> hier: Dokumentation des Moduls   ..........................@ACC@../
/.............................................................@ECC@.*/


/*<f> History:
<--->*/



/*    TYPEN   */
/*-------------------------------------------------------*/
#ifndef uchar
#define  uchar       unsigned char           /* BYTE  8 */
#endif
#ifndef uint
#define  uint        unsigned int
#endif
#ifndef ushort
#define  ushort      unsigned short          /* WORD  16 */
#endif
#ifndef ulong
#define  ulong       unsigned long           /* DWORD 32 */
#endif
#ifndef ullong
#define  ullong      unsigned long long      /* DWORD 64 */
#endif
#ifndef llong
#define  llong       long long
#endif
#ifndef ldouble
#define  ldouble     long double
#endif
```

```
/*   BITS   fuer Bits: 0..7..15..31 */
/*-------------------------------------------------------*/
#define  BITVALUE(val, bit)   ( ((val) & (1 << (bit)))? 1:0 )
#define  ISBITSET(val, bit)   ( ((val) & (1 << (bit)))? 1:0 )
#define  ISBITRES(val, bit)   ( ((val) & (1 << (bit)))? 0:1 )

#define  SETBIT(val, bit)     ( (val) |= ( (1 << (bit)) ) )
#define  RESBIT(val, bit)     ( (val) &= (~(1 << (bit)) ) )
#define  XCHBIT(val, bit)     ( (val) ^= ( (1 << (bit)) ) )

/*   STRING/CHAR   */
/*-------------------------------------------------------*/
#define  b2a(b)           ( ((char)(b) + (char) '0') )
#define  a2b(c)           ( ((char)(c) - (char) '0') )

/*   MATH   */
/*-------------------------------------------------------*/
#define  SIGNREL(x,r)     ( (x)==(r) ?  0 : (x)>(r) ? 1:-1 )
#define  SIGN(x)          ( SIGNREL( (x), 0 ) )

#define  ISEVEN(val)      ( ISBITRES((val),0)? 1:0  )
#define  ISODD(val)       ( ISBITSET((val),0)? 1:0 )

#define  DIV2(val)        ( (val) >> 1 )
#define  DIV4(val)        ( (val) >> 2 )
#define  MUL2(val)        ( (val) << 1 )
#define  MUL4(val)        ( (val) << 2 )

#define  RANGE(l,x,u)     ( ((x)>=(l) && (x)<=(u)) ? 0 : ((x)>(u)? 1 : -1) )
#define  ISINRANGE(l,x,u) ( RANGE( (l),(x),(u) ) ? 0:1 )
#define  TORANGE(l,x,u)   ( ((x)<=(l))? (l) : \
                          ( ((x)>=(u))? (u) : (x) ) )
/* base_value, new_value, delta_max */
#define  TORANGEREL(b,x,d)( ((x)>=(b)+(d)) ? ((b)+(d)) : \
                          ( ((x)<=(b)-(d)) ? ((b)-(d)) : (x) ) )

/*   MISC   */
/*-------------------------------------------------------*/
#define  DIFFTIME(n,o)    ( (n) - (o) )
#define  FOREVER          for(;;)

/*   COMMON   */
/*-------------------------------------------------------*/
#ifndef TRUE
#define  TRUE             1
#endif
#ifndef FALSE
#define  FALSE            0
#endif
#ifndef YES
#define  YES              1
#endif
#ifndef NO
#define  NO               0
#endif
#ifndef JA
#define  JA               1
#endif
#ifndef NEIN
#define  NEIN             0
#endif
#ifndef OKAY
#define  OKAY             1
#endif
#ifndef NOTOKAY
#define  NOTOKAY          0
#endif
```

```
#ifndef GOOD
#define  GOOD              1
#endif
#ifndef BAD
#define  BAD               1
#endif
#ifndef ACTIVE
#define  ACTIVE            1
#endif
#ifndef INACTIVE
#define  INACTIVE          0
#endif
#ifndef ON
#define  ON                1
#endif
#ifndef OFF
#define  OFF               0
#endif
#ifndef AUF
#define  AUF               1
#endif
#ifndef ZU
#define  ZU                0
#endif
#ifndef VOLL
#define  VOLL              1
#endif
#ifndef LEER
#define  LEER              0
#endif
#ifndef HIGH
#define  HIGH              1
#endif
#ifndef LOW
#define  LOW               0
#endif
#ifndef POSITIV
#define  POSITIV           0
#endif
#ifndef NEGATIV
#define  NEGATIV           1
#endif

#ifndef RECHTS
#define  RECHTS            0
#endif
#ifndef LINKS
#define  LINKS             1
#endif

#ifndef ERROR
#define  ERROR             -1
#endif
#ifndef NOERROR
#define  NOERROR           0
#endif
#ifndef FAILURE
#define  FAILURE           -1
#endif
#ifndef SUCCESS
#define  SUCCESS           0
#endif

#ifndef EOS
#define  EOS               ( (char)0 )          /* EndOfString */
#endif
#ifndef NIL
#define  NIL               ( (void*)0 )         /* NotInList-pointer */
#endif
```

```
#ifndef   ACL                                        /* ACL defined? */
   #ifndef VOID_FUNC
   typedef void (*VOID_FUNC)( void );                    /* func returning nothing */
   #endif

   #ifndef INT_FUNC
   typedef int (*INT_FUNC)( void );                      /* func returning int */
   #endif
#endif

#ifndef NULL_PTR
#define  NULL_PTR          ( (void*)NULL )      /* ptr to nothing */
#endif


/*    USEFUL   */
/*---------------------------------------------------------*/
#define  MK_DWORD(h,l)     ( (long) (((long)(h) << 16) | (int)(l)) )
#define  MK_WORD( h,l)     ( (int)  (((int) (h) <<  8) | (int)(l)) )

#define  MK_FARPTR(seg,off)   ( (void far *) MK_DWORD((seg),(off)) )

/* Calculate the byte offset of a field in a structure of type type. */
#define  STRUCTOFF(type, field)  ( (int) &(((type *)0)->field) )

/* Extract high and low order parts of 32/16 bit quantity */
#ifndef  LOWORD
#define  LOWORD(l)         ( (ushort)(l) )
#endif
#ifndef  HIWORD
#define  HIWORD(l)         ( (ushort)((l)>>16) )
#endif
#ifndef  LOBYTE
#define  LOBYTE(w)         ( (uchar)(w) )
#endif
#ifndef  HIBYTE
#define  HIBYTE(w)         ( (uchar)((w)>>8) )
#endif

/* far: mem/SS:off   near: DS:off */
#ifndef  SEGOF
#define  SEGOF(p)          ( HIWORD( (long)(void far *)(p) ) )
#endif
#ifndef  OFFOF
#define  OFFOF(p)          ( LOWORD( (long)(void far *)(p) ) )
#endif

/* make ptr to loc 'p+n'; n = byte-offset; wrap not handled! */
#define  TOPTR(p,n)        ( (void*) (((char*)(p))+(n)) )


/* get value in type 't' from loc pointed to by 'p+n'; n = byte-offset ! */
/*      e.g. GETTYPE( float, &var, 5 ) -> (float) (&var+5byte) */
#define  GETTYPE(t,p,n)    ( (t) *( (t*)(TOPTR((p),(n))) ) )
/* get 'byte' pointed to by 'p+n'; n = byte-offs = 0..i */
#define  GETBYTE( p,n)     ( GETTYPE(char,(p),(n)) )

/* put value in type 't' from loc pointed to by 'p+n'; n = byte-offset ! */
#define  PUTTYPE(t,p,n,v)  ( *( (t*)(TOPTR((p),(n))) ) = (t)(v) )
/* put 'byte' pointed to by 'p+n'; n = byte-offs = 0..i */
#define  PUTBYTE( p,n,v)   ( PUTTYPE(char,(p),(n),(v)) )


/*---   eof: com_1st.h   ---*/
```

2.4.2 com_def.h

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ACH@-*/
/*
*/
/*    LastUpdate   COM_DEF.H   09-11-2001 12:00  R1.14   by Ebert,T.   */
/*
**
**    PROJECT:   CNC21-Steuerung
**
**    VERSION:   V1.0                 $Revision:   1.15  $
**
**    MODULE DESCRIPTION:   com_def.h    ( HEADER (DEF) )
**    ------------------------------------------------------------------
**
**    defines, die fuer das gesamte cnc-system (mmi & ncr) wichtig sind
**
**
**
**
**
**
**    (C) ECKELMANN INDUSTRIEAUTOMATION, WIESBADEN
**    ------------------------------------------------------------------
**    CREATED: 16-07-91 17:54   AUTHOR: TRAISER    V: V1.0
**-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ECH@-*/


/* $Header:   P:\cnc21\include\vcs\com_def.h_v   1.15   09 Nov 2001 12:06:18   ebert_0050046BD496  $ */

/*<f> History:
Eb 29-08-00 17:21  R1.2  NCK_MESS_KANAL_ANZAHL_KC von 8 auf 16 erhöht.
Eb 07-11-00 09:45  R1.3  NCK_SFELD_ANZAHL_KUS und NCK_TFELD_ANZAHL_KUS von 32 auf 100 erhöht.
Eb 20-07-01 13:45  R1.12  12 Achsen bei CNC55 und PNC55.
Eb 04-10-01 10:44  R1.13  CONIOMSG_TN: Attribut swapped hat gefehlt.
Eb 09-11-01 12:00  R1.14  MSG_TR: index_uc und hande_uc getauscht.
<--->*/


/* <f> */
/*****************************************************************************/
/*  Include *.h Gesamt-Applikation betreffend ****************************/
/*****************************************************************************/
#ifndef MMI
 #include "ncr_ver.h"                 /* für Versionsabh„ngige Einstellungen */
 #define swapped  __packed__(1,1,1)    /* prefix für little endian Datenstrukturen */
#else
 #define swapped                      /* prefix für little endian Datenstrukturen */
#endif

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Festlegung von Feldgroessen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#define  PFELD_ANZAHL_KUS              2048  /* Anzahl der Einträge (Elemente, keine Bytes) */
#define  QFELD_ANZAHL_KUS                64  /* Anzahl Q-Feld-Meldungen der sps */
#define  QTAB_ANZAHL_KUC                  8  /* Max. Anzahl von gleichzeitig aktiven Q-Bit-Abfragen */
#define  VTAB_ANZAHL_KUC                  8  /* Max. Anzahl von gleichzeitig aktiven V-Tab-Abfragen */
#ifndef __NCR_ENC55__     /* CPU05/PNC55 */
 #define  NCK_SFELD_ANZAHL_KUS          100  /* Anzahl von Werkstückkoordinatensystemen */
 #define  NCK_TFELD_ANZAHL_KUS          100  /* Anzahl von Werkzeugkoordinatensystemen */
#else                     /* ENC55 */
 #define  NCK_SFELD_ANZAHL_KUS           32  /* Anzahl von Werkstückkoordinatensystemen */
 #define  NCK_TFELD_ANZAHL_KUS           32  /* Anzahl von Werkzeugkoordinatensystemen */
#endif

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Achsen-Definitionen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#ifndef MMI
```

```
 #ifndef __NCR_ENC55__     /* CPU05/PNC55 */
  #define  NCK_PHYACHS_KC            12  /* Anzahl physikalischer Achsen */
  #define  NCK_APPLACHS_KC           12  /* Anzahl der vorhandenen Achsen (applikationsabhängig) */
  #define  NCK_LOGACHS_KC             6  /* logische Achsen, d.h. Achsen, die gleichzeitig mit einer */
  #define  NCK_SERCACHS_KC            8  /* max. Anzahl der SERCOS-Achsen */
  #define  NCK_SPLINEACHS_KC          6  /* Anzahl der gleichzeitigen Splineachsen */
  #define  NCK_SPINDELANZAHL_KC       3  /* Anzahl der verwaltbaren Spindeln */
  #define  NCK_HANDRADANZAHL_KC       2  /* Anzahl der verwaltbaren Handräder */
  #define  NCK_PWMANZAHL_KC           4  /* Anzahl der PWM-Kanäle */
  #define  NCK_KANAL_ANZAHL_KC        3  /* n*NB+1*UB-Kanaele, der letzte Kanal ist UB! */
 #else                      /* ENC55 */
  #define  NCK_PHYACHS_KC             8  /* Anzahl physikalischer Achsen */
  #define  NCK_APPLACHS_KC            8  /* Anzahl der vorhandenen Achsen (applikationsabhängig) */
  #define  NCK_LOGACHS_KC             5  /* logische Achsen, d.h. Achsen, die gleichzeitig mit einer   */
  #define  NCK_SERCACHS_KC            8  /* max. Anzahl der SERCOS-Achsen */
  #define  NCK_SPLINEACHS_KC          5  /* Anzahl der gleichzeitigen Splineachsen */
  #define  NCK_SPINDELANZAHL_KC       2  /* Anzahl der verwaltbaren Spindeln */
  #define  NCK_HANDRADANZAHL_KC       1  /* Anzahl der verwaltbaren Handräder */
  #define  NCK_PWMANZAHL_KC           0  /* Anzahl der PWM-Kanäle */
  #define  NCK_KANAL_ANZAHL_KC        2  /* n*NB+1*UB-Kanaele, der letzte Kanal ist UB! */
 #endif
 #define  NCK_WERKZEUGKORRANZ_KC     30  /* Anzahl der Werkzeugspezifischen Korrekturwerte */
 #define  NCK_MAX_FILTERBREITE      256  /* maximale Vist-Verzögerung in GIT's (Tiefpass) */
#else
 #define  NCK_PHYACHS_KC            12  /* Anzahl physikalischer Achsen */
 #define  NCK_APPLACHS_KC           12  /* Anzahl der vorhandenen Achsen (applikationsabhängig) */
 #define  NCK_KANAL_ANZAHL_KC        3   /* n*NB+1*UB-Kanaele, der letzte Kanal ist UB! */
#endif

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Kanal-Definitionen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#define  NCK_KANAL_EBENE_ANZAHL_KC        2  /* Ebenen in einem Kanal */
#define  NCK_UB_KANAL_INDEX_KC         NCK_KANAL_ANZAHL_KC -1 /* Unterbrechen-Kanal, logisches Define */
#define  NCK_MESS_KANAL_ANZAHL_KC        16  /* Anzahl Messkanälen */

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* defines fuer Fehlergewichtung */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#define  F_KLAS_LOKAL_LEICHT_KUC          1        /* lokal leicht */
#define  F_KLAS_LOKAL_SCHWER_KUC          2        /* lokal schwer */
#define  F_KLAS_GLOBAL_LEICHT_KUC         3        /* global leicht */
#define  F_KLAS_GLOBAL_SCHWER_KUC         4        /* global schwer */
#define  F_KLAS_FATAL_KUC                 5        /* fatal-error => shut down */

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Parameter fuer Quittung auf Blockuebertragung */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#define  DUE_QUITT_OK_KC                  0  /* Block akzeptiert, her mit dem naechsten Block */
#define  DUE_QUITT_NOT_OK_KC           -128  /* allgemein Blockübertragung abbrechen */
/* -1 bis -39 reserviert für Betriebssystem-Returncodes */
#define  DUE_QUITT_MK_FEHLER_1_KC      -40  /* Falscher MK-Bezeichner */
#define  DUE_QUITT_MK_FEHLER_2_KC      -41  /* Kommentarschachtelungsfehler */
#define  DUE_QUITT_MK_FEHLER_3_KC      -42  /* Falscher Steuerblock */
#define  DUE_QUITT_MK_FEHLER_4_KC      -43  /* ZST nicht im Idle-Zustand */
#define  DUE_QUITT_MK_FEHLER_5_KC      -44  /* Parameterende nicht gefunden, Semikolon fehlt */
#define  DUE_QUITT_WAIT_KC             -50  /* KEIN FEHLER: */
                                            /* vorläufige Block-Quittung, wird zyklisch bis */
                                            /* zur entgültigen Block-Quittung wiederholt */
#define  DUE_QUITT_BREAK_KC            -51  /* Abbruchanforderung der Übertragung, da weitere */
                                            /* Programmanlage sinnlos (Programmausführung gestoppt) */
#define  DUE_QUITT_DIN_FEHLER_1_KC     -60  /* DIN-Satz länger als erlaubt */
#define  DUE_QUITT_DIN_FEHLER_2_KC     -61  /* Fehler bei der Programmanlage */
#define  DUE_QUITT_DIN_FEHLER_3_KC     -62  /* Fehler im DIN-Programm */
#define  DUE_QUITT_DIN_FEHLER_4_KC     -63  /* Leeres DIN-Programm, keine Satzdaten */
#define  DUE_QUITT_DIN_FEHLER_5_KC     -64  /* Fehler beim Programmöffnen */
#define  DUE_QUITT_DIN_FEHLER_6_KC     -65  /* Keine Programmnummer gefunden */
#define  DUE_QUITT_DIN_FEHLER_7_KC     -66  /* Kein Abschlusszeichen am Satzende */
```

```
#define  DUE_QUITT_SDO_FEHLER_KC        -70  /* SDO Transfer abgebrochen */
#define  DUE_QUITT_SDO_TIMEOUT_KC       -71  /* SDO Timeout */
#define  DUE_QUITT_SDO_ABBRUCH_KC       -72  /* SDO Abbruch: Objekt nicht vorhanden */
/* -80 bis -127 reserviert für Betriebssystem-Returncodes */

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* File-IO über DLL-Fuktionen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Parameter für SB1_OPEN_FILE_KUC */
#define  DLL_IOMODE_RDONLY_KUL            0  /* Datei zum Lesen öffnen */
#define  DLL_IOMODE_WRONLY_KUL            1  /* Datei zum Schreiben öffnen */

/* Fehlercodes (Rückmeldung erfolgt über SB2) */
#define  DLL_IOERR_ZERO_KUC               0  /* kein Fehler */
#define  DLL_IOERR_NOFILE_KUC             1  /* File not found */
#define  DLL_IOERR_NOPATH_KUC             2  /* Path not found */
#define  DLL_IOERR_MFILE_KUC              3  /* Too many open files */
#define  DLL_IOERR_ACCES_KUC              4  /* Permission denied */
#define  DLL_IOERR_NOPEN_KUC              5  /* File not open */
#define  DLL_IOERR_OHTER_KUC              6  /* anderer Fehler aufgetreten */

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Achsflags aus ACHSINFO_TR */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#define  ACHSFLAG_ROTATIONSACHSE    0x0001   /* keine Linearachse */
#define  ACHSFLAG_OHNE_ENDSCHALTER  0x0002   /* Achse ohne Endschalter */
#define  ACHSFLAG_SPINDEL           0x0004   /* Achse ist eine Spindel */
#define  ACHSFLAG_MESSACHSE         0x0008   /* Achse ist eine Messachse */
#define  ACHSFLAG_MODULO360         0x0010   /* Modulo 360ø Achse */
#define  ACHSFLAG_KUERZERER_WEG     0x0020   /* bei Mod386ø-Achsen wird der kürzere Weg gefahren */
#define  ACHSFLAG_GANTRYACHSE       0x0040   /* Synchronachse ist eine Gantryachse */
#define  ACHSFLAG_HANDRAD           0x0080   /* an dieser Achse ist ein Handrad angeschlossen */
#define  ACHSFLAG_SYNCHRONACHSE     0x8000   /* diese Achse ist eine Synchronachse */

/* Modultypen für IOMODULINFO_TR */
#define  MOD_DIGITAL                1
#define  MOD_ANALOG                 2
#define  MOD_CAN_IO                 3
#define  MOD_CAN_DRIVE              4
#define  MOD_CAN_HMI                5

/* Definitionen für Nachrichtenstrukturen */
#define  MSG_HEADERLEN_KI           STRUCTOFF(MSG_TR,n)
#define  MMI_TRACEMAX               12


/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Fehlermeldungen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

/*-------------------------------------------------------------------------*/
typedef swapped struct ERR_MELD_TR {  /* Struktur für Fehlermeldungen innerhalb des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             task_uc;               /* fehlermeldende Funktion */
   uchar             klasse_uc;             /* Fehlergewichtung */
   short             nummer_s;              /* Fehlernummer */
   char              info_format_ac [32];   /* Nur Format (vgl.printf, "%d%l"), keine Texte */
   char              info_data_ac   [80];   /* Nutzdaten der Zusatzinformation */
} ERR_MELD_TR;

/*-------------------------------------------------------------------------*/
typedef struct ERR_QUITT_TR { /* Struktur für Fehlermeldungsquittung innerhalb des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             task_uc;          /* fehlermeldende Funktion */
   uchar             klasse_uc;        /* Fehlergewichtung */
} ERR_QUITT_TR;
```

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Parameterfeld */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

/*-------------------------------------------------------------------------*/
typedef swapped struct PFELD_LISTE_TR {  /* Übertragung von P-Feld-Inhalten als Liste */
/*-------------------------------------------------------------------------*/
   ushort            idx_us;        /* P-Feld-Index */
   double            wert_d;        /* P-Feld-Wert */
}  PFELD_LISTE_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct PFELD_BLOCK_TR {  /* Übertragung von P-Feld-Inhalten als Block */
/*-------------------------------------------------------------------------*/
   ushort            idx_us;        /* P-Feld-Index des ersten Parameters */
   ushort            anz_us;        /* Anzahl der nachfolgenden Parameter */
   double            wert_ad[63];   /* P-Feld-Werte (max. 508 Byte) */
}  PFELD_BLOCK_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct PFELD_ANZEIGE_TR { /* Beauftragung der zyklischen P-Feld-Anzeige */
/*-------------------------------------------------------------------------*/
   ushort            pos_us;        /* Position in der Anzeige */
   ushort            idx_us;        /* Index im P-Feld  */
}  PFELD_ANZEIGE_TR;

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Programmverwaltung */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

/*-------------------------------------------------------------------------*/
typedef swapped struct STARTPROG_TR { /* struct für SB1_PROGRAMM_START_KUC */
/*-------------------------------------------------------------------------*/
   ushort            prognr_us;        /* Programmnummer */
   ushort            satznr_us;        /* Satznummer */
   uchar             kanal_uc;         /* z.Z. unbenutzt, auf 0 initialisieren */
   signed char       errlevel_c;       /* wenn Programmstart nicht möglich ist: */
                                       /* 0=alle Fehlermeldungen erlauben, */
                                       /* 5=alle Fehlermeldung unterdrücken, */
   uchar             startmode_uc;     /* Belegung wie Starttaste in der virtuellen Tastatur */
                                       /* zusätzlich: 0x81=Prognr u. Satznr in P512 u. P513 und starten */
} STARTPROG_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct PROGENDE_TR {  /* struct fuer SB1_PROGRAMM_ENDE_KUC */
/*-------------------------------------------------------------------------*/
   short             quittung_s;       /*   -1 = kein Programm gestartet, Startmode war 0 */
                                       /*    0 = Programm wurde mit M30 beendet */
                                       /*    1 = Stop-Signal steht an, Programmstart nicht möglich */
                                       /*    2 = Steuerung ist nicht bereit, Programmstart nicht möglich */
                                       /*    3 = kein Programm aktiv, Start nächster Satz nicht möglich */
                                       /*    4 = Notaus-Signal steht an, Programmstart nicht möglich */
                                       /*    5 = Ungültiger Startmode, Programmstart nicht möglich */
                                       /* -128 = Programm wurde gestartet und vorzeitig abgebrochen */
} PROGENDE_TR;

/*-------------------------------------------------------------------------*/
typedef swapped union CONIOMSG_TN {  /* Ein-/Ausgabe über G-Fkt */
/*-------------------------------------------------------------------------*/
   char              out_ac[1];     /*-- Einfacher Text mit      SB1_GFKT_MELDUNG_KUC --*/
   swapped struct {                 /*-- Positionierter Text mit SB1_GFKT_MELDUNGXY_KUC --*/
      short             curx_s;     /* xCursor */
      short             cury_s;     /* yCursor */
      short             attrib_s;   /* Zeichenattribut */
      short             pad_s;      /* Füllwort */
      ulong             textidx_ul; /* Textindex */
      double            werta_d;    /* Parameter A */
      double            wertb_d;    /* Parameter B */
      double            wertc_d;    /* Parameter C */
```

```
      char              fmt_ac[1];  /* Formatanweisung für Ausgabe */
   }                 outxy_r;
   swapped struct {                 /*-- Eingabe mit SB1_EINGABEXY_KUC --*/
      short             curx_s;     /* xCursor */
      short             cury_s;     /* yCursor */
      short             attrib_s;   /* Zeichenattribut */
      short             idx_s;      /* Zielindex im PFeld */
      ulong             textidx_ul; /* Textindex */
      double            min_d;      /* min-Wert der Eingabe */
      double            max_d;      /* max-Wert der Eingabe */
      double            dflt_d;     /* aktueller Wert vom P-Feld[idx_s] */
      char              fmt_ac[1];  /* Formatanweisung für Eingabe */
   }                 inxy_r;
} CONIOMSG_TN;

/*---------------------------------------------------------------------------*/
typedef swapped struct ACHSINFO_TR { /* Informationen zu einer Achse abrufen */
/*---------------------------------------------------------------------------*/
   char              kenn_c;           /* -1, wenn Achse nicht konfiguriert */
   uchar             applachsidx_uc;
   ushort            achsflags_us;
   double            faktor_v_d;       /* [Einheiten / min] >> [Einheiten / delta_t] */
   double            faktor_b_d;       /* [Einheiten / sec^2 ] >> [Einheiten / delta_t^2] */
   float             deltat_f;         /* Grobinterpolationstakt */
   ulong             fit_pro_git_ul;
   float             impulse_f;
   float             weg_f;
   float             massstab_f;
   float             grundoffset_f;
   float             spindelumkehr_f;
   float             synchronabweichung_f;
   float             sw_ends_minus_f;
   float             sw_ends_plus_f;
   float             schleppgenauhalt_f;
   ushort            schleppabstand_us;
   uchar             schleppcnt_uc;
   uchar             regler_mode_uc;
   float             kp_f;
   float             kf_f;
   float             tv_f;
   float             tn_f;
   float             t2_f;
   ushort            achseingaenge_us;
   uchar             reftyp_uc;
   signed char       richtungundfolge_c;
   float             refvmax1_f;
   float             refbmax1_f;
   float             refvmax2_f;
   float             refbmax2_f;
   float             modvmax_f;
   float             vmax_f;
   float             beschl_f;
   float             brems_f;
   float             t_beschl_f;
   float             einheit_f;        /* Einheiten pro mm */
   float             kb_f;
} ACHSINFO_TR;

/*---------------------------------------------------------------------------*/
typedef swapped struct REGPARAM_TR {  /* Online Änderung von Reglerparametern */
/*---------------------------------------------------------------------------*/
   uchar             checksum_uc;   /* Summe über die Nutzdatenfracht muss 0 sein */
   uchar             achsnr_uc;     /* Für welche Applikationsachse sind die Parameter bestimmt */
   ushort            enable_us;     /* Bitcodierte Freigabe für die Änderung der nachfolgenden Daten */
   float             kp_f;          /* neuer Wert von MK_KP, falls Bit0 in enable_us gesetzt */
   float             kf_f;          /* neuer Wert von MK_KF, falls Bit1 in enable_us gesetzt */
   float             tv_f;          /* neuer Wert von MK_TV, falls Bit2 in enable_us gesetzt */
   float             tn_f;          /* neuer Wert von MK_TN, falls Bit3 in enable_us gesetzt */
   ulong             reglermode_ul; /* neuer Wert von MK_REGLERMODE, falls Bit4 in enable_us gesetzt */
```

```
   float            kb_f;         /* neuer Wert von MK_KB, falls Bit5 in enable_us gesetzt */
} REGPARAM_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct IOMODULINFO_TR {  /* Informationen zu den lokalen/externen IO-Modulen */
/*-------------------------------------------------------------------------*/
   uchar            modultyp_uc;
   uchar            dianzahl_uc;   /* Anzahl der digitalen Eingänge */
   uchar            doanzahl_uc;   /* Anzahl der digitalen Ausgänge */
   uchar            aianzahl_uc;   /* Anzahl der analogen Eingänge */
   uchar            aoanzahl_uc;   /* Anzahl der analogen Ausgänge */
   uchar            dioffset_uc;   /* Index in dig. Eing.-Array im dpr */
   uchar            dooffset_uc;   /* Index in dig. Ausg.-Array im dpr */
   uchar            aioffset_uc;   /* Index in anal. Eing.-Array im dpr */
   uchar            aooffset_uc;   /* Index in anal. Ausg.-Array im dpr */
   uchar            node_uc;       /* Knotennummer (CAN-Id) */
   uchar            reserve_auc[2];
} IOMODULINFO_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct SDOAUFTRAG_TR {  /* Aufträge zum Lesen/Schreiben von CANopen-Objekten */
/*-------------------------------------------------------------------------*/
   uchar    id_uc;
   ushort   objektidx_us;
   uchar    subidx_uc;
   long     len_l;
#ifndef __TURBOC__
   uchar    data_auc[0];
#endif
} SDOAUFTRAG_TR;

/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* File-IO über DLL-Fuktionen */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

/*-------------------------------------------------------------------------*/
typedef swapped struct DLL_DIRENTRY_TR {  /* Inhalt eines Directory Eintrags */
/*-------------------------------------------------------------------------*/
   char           filename_ac[12];  /* MS-DOS Dateiname (8.3) nullterminiert falls Name < 12 Zeichen */
   ulong          filesize_ul;      /* Dateigröße in Bytes */
}  DLL_DIRENTRY_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct DLL_FILEOPEN_TR {  /* Datei zum Lesen oder Schreiben öffnen */
/*-------------------------------------------------------------------------*/
   ulong          mode_ul;          /* Lesen oder Schreiben */
   char           filename_ac[16];  /* Dateiname */
}  DLL_FILEOPEN_TR;

/*-------------------------------------------------------------------------*/
typedef swapped struct DLL_FILEIO_TR { /* Lesen/Schreiben einer geöffneten Datei */
/*-------------------------------------------------------------------------*/
   ulong          iooffset_ul;      /* Schreib-/Lese-Offset in der Datei */
   uchar          iodata_auc[508];  /* zu Schreibenden bzw. gelesene Daten */
}  DLL_FILEIO_TR;
```

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
/* Nachricht inclusive Header */
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

/*-------------------------------------------------------------------------*/
typedef swapped struct MSG_TR {  /* Nachrichtenstruktur für alle Nachrichten des Gesamtsystems */
/*-------------------------------------------------------------------------*/
   uchar             sb0_uc;                    /* Steuerblock 0 */
   uchar             sb2_uc;                    /* Steuerblock 2 bei Bedarf */
   uchar             sb1_uc;                    /* Steuerblock 1 */
   uchar             index_uc;                  /* Wird bei Anfragen durchgeschleift */
   uchar             modul_uc;                  /* Nachrichtensender */
   uchar             handle_uc;                 /* Wird bei Anfragen durchgeschleift */
   ushort            len_us;                    /* Länge */
   swapped union {
      char              val_ac [512];           /* allgemein */
      uchar             val_auc[512];
      short             val_as [256];
      ushort            val_aus[256];
      long              val_al [128];
      ulong             val_aul[128];
      float             val_af [128];
      double            val_ad [64];
      void             *val_apv[128];

      ERR_MELD_TR       err_meld_r;             /* Fehlermeldung */
      ERR_QUITT_TR      err_quitt_r;            /* Fehlerquittung */
      PFELD_LISTE_TR    pfeld_liste_ar[51];     /* Liste mit P-Feldwerten (max. 510 Byte) */
      PFELD_BLOCK_TR    pfeld_block_r;          /* zusammenhängender block mit P-Feldwerten */
      PFELD_ANZEIGE_TR  pfeld_anz_ar[32];       /* Parametisierung für zyklische P-Feld-Anzeige */
      STARTPROG_TR      startprog_r;            /* Programmstart */
      PROGENDE_TR       progende_r;             /* Quittung am Programmende */
      CONIOMSG_TN       coniomsg_n;             /* Ein-/Ausgabe über G-Fkt */
      ACHSINFO_TR       achsinfo_r;             /* Achsinformationen */
      REGPARAM_TR       regparam_r;             /* Online-Änderung von Reglerparametern */
      IOMODULINFO_TR    iomodulinfo_ar[42];     /* Informationen zu den IO-Modulen (max. 504 Byte) */
      SDOAUFTRAG_TR     sdoauftrag_ar[64];      /* Aufträge zum Lesen/Schreiben von CANopen-Objekten */
      DLL_FILEOPEN_TR   fileopen_r;             /* Datei zum Lesen oder Schreiben öffnen */
      DLL_FILEIO_TR     fileio_r;               /* Lesen/Schreiben einer geöffneten Datei */
      DLL_DIRENTRY_TR   dir_ar[32];             /* bis zu 32 Verzeichniseinträge */
   }                 n;
}  MSG_TR;

/*---   eof: com_def.h   ---*/
```

2.4.3 com_sbx.h

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ACH@-*/
/*
*/
/*    LastUpdate   COM_SBX.H   09-07-2001 14:36  R1.18   by Ebert,T.   */
/*
**
**    PROJECT:   CNC21-Steuerung
**
**    VERSION:   V1.0                 $Revision:   1.21  $
**
**    MODULE DESCRIPTION:   com_sbx.h    ( HEADER (DEF) )
**    ---------------------------------------------------------------
**
**    Definitionen der Hauptsteuerblockgruppen zur Inter-Task-Kommunikation
**    des NCR Teilsystems, sowie zur Nachrichtenkommunikation zwischen MMI
**    und NCR.
**    Definition der Untergruppen zur Inter-Task-Kommunikation, sowie zur
**    Nachrichtenkommunikation zwischen MMI und NCR.
**
**
**    (C) ECKELMANN INDUSTRIEAUTOMATION, WIESBADEN
**    ---------------------------------------------------------------
**    CREATED: 24-01-91 12:51   AUTHOR: Lauter    V: V1.0
**-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ECH@-*/


/* $Header:   P:\cnc21\include\vcs\com_sbx.h_v   1.21   13 May 2002 13:24:02   ebert_0050046BD496  $ */

/*<f> History:
Eb 26-02-01 10:03  R1.10  SB1_CAN1_OBJECT_KUC, SB1_CAN2_OBJECT_KUC.
Eb 26-02-01 10:35  R1.11  TASK_COM_KC -> TASK_CAN_KC.
Eb 26-02-01 14:14  R1.12  TASK_NOV_KC.
Eb 08-03-01 09:16  R1.13  SB2_CAN_OPERATIONAL_KUC, SB2_CAN_PREOPERATIONAL_KUC.
Eb 09-07-01 14:36  R1.18  SB0_SPSAUFTRAG_KUC.
<--->*/

/* ------------------------------------------------------------------------- */

#define  TASK_AXE_KC       1
#define  TASK_INI_KC       2
#define  TASK_INT_KC       3
#define  TASK_KOP_KC       4
#define  TASK_PLC_KC       5
#define  TASK_POS_KC       6
#define  TASK_SPV_KC       7
#define  TASK_WRK_KC       8
#define  TASK_ZST_KC       9
#define  TASK_SPS_KC       10
#define  TASK_SYS_KC       11
#define  TASK_CAN_KC       12
#define  TASK_UTI_KC       13

#define  SYS_NCR_KC        14       /* Kommunikation NCR -> MMI */
#define  SYS_MMI_KC        15       /* Kommunikation MMI -> NCR */

#define  TASK_XT3_KC       16       /* Kommunikation XT3 -> NCR */
#define  TASK_TPR_KC       17       /* Koppeltransputer generell */
#define  TASK_DRV_KC       18       /* CAN Drive (2. CAN-Controller) */
#define  TASK_LNZ_KC       19       /* Lenze-Fehlermeldungen */
#define  TASK_NOV_KC       20       /* Novotron-Fehlermeldungen */
#define  TASK_SDL_KC       21       /* Seidel-Fehlermeldungen */
```

```
/*-----------------------------------------------------------------------------
 SB0 Hauptgruppen gültig für NCR Inter-Task-Kommunikation, sowie
     für Nachrichtenübertragung von NCR -> MMI und MMI -> NCR
-----------------------------------------------------------------------------*/
#define  SB0_INIT_KUC                    1     /* interne Initialisierungsnachrichten */
#define  SB0_DISPLAY_KUC                 2     /* Anzeigenachrichten */
#define  SB0_AUFTRAG_KUC                 3     /* Beauftragung einer Unterfunktion */
#define  SB0_DATENANFORDERUNG_KUC        4     /* allgemeine Datenanforderungen */
#define  SB0_DUE_FIRST_BLOCK_KUC         5     /* Datenübertragung erster Block */
#define  SB0_DUE_NEXT_BLOCK_KUC          6     /* Datenübertragung nächster Block */
#define  SB0_DUE_LAST_BLOCK_KUC          7     /* Datenübertragung letzter Block */
#define  SB0_DUE_FIRST_QUITT_KUC         8     /* Quittung auf ersten Datenblock */
#define  SB0_DUE_NEXT_QUITT_KUC          9     /* Quittung auf nächsten Datenblock */
#define  SB0_DUE_LAST_QUITT_KUC          10    /* Quittung auf letzten Datenblock */
#define  SB0_POLL_KUC                    11    /* Poll-Nachricht */
#define  SB0_DUE_BREAK_BLOCK_KUC         12    /* Vorzeitiger Abbruch der Blockübertragung */
#define  SB0_DLL_SERVICE_KUC             13    /* IO-Befehl an MMI */
#define  SB0_SPSAUFTRAG_KUC              14    /* Auftrag von SPS an MMI oder umgekehrt */
#define  SB0_CODESYS_KUC                 15    /* Auftrag zwischen Codesys und PLC-Laufzeitsystem */
#define  SB0_RESET_KUC                   85    /* Auftrag zum Resetten der Steuerung */
#define  SB0_DEBUG_KUC                   128   /* Debug-Nachricht (interne Verwendung) */
#define  SB0_KEYBOARD_KUC                129   /* Debug-Nachricht (interne Verwendung) */
#define  SB0_EINGABE_KUC                 130   /* Debug-Nachricht (interne Verwendung) */

#define  SB0_EXCEPTION_KUC               255   /*--- Ausnahmemeldung (z.B. Systemabbruch) ---*/


/*-----------------------------------------------------------------------------
 SB1 Untergruppen gültig fuer die jeweils angegebene SB0-Hauptgruppe
-----------------------------------------------------------------------------*/
/*--- SB1  bei SB0 = SB0_POLL_KUC ---*/
#define  SB1_POLL_KUC                    1     /* Poll-Nachricht */


/*--- SB1  bei SB0 = SB0_DISPLAY_KUC ---*/
#define  SB1_PLC_MELDUNG_KUC             1     /* PLC -> MMI Textausgabe über Textnummer  */
#define  SB1_PLC_LED_KUC                 2     /* PLC -> MMI LED ein/aus */
#define  SB1_GFKT_MELDUNG_KUC            3     /* INT -> MMI Textausgabe über G-Funktion */
#define  SB1_GFKT_MELDUNGXY_KUC          4     /* INT -> MMI Textausgabe über G-Funktion an Stelle XY */
#define  SB1_GFKT_EINGABEXY_KUC          5     /* INT -> MMI Eingabeanforderung an MMI an der Stelle XY */


/*--- SB1  bei SB0 = SB0_AUFTRAG_KUC ---*/
#define  SB1_ASCII_EINZELFUNKTION_KUC    1     /* MMI -> ZST Einzelsatzbearbeitung */
#define  SB1_EINZELFKT_MIT_QUITTUNG_KUC  2     /* MMI -> ZST Einzelsatzbearbeitung mit Quittungsmeldung */
#define  SB1_CLEAR_MEM_KUC               3     /* MMI -> ZST -> SPV löschen des gesamten SPV-Speichers */
#define  SB1_CLEAR_USERMEM_KUC           4     /* MMI -> ZST -> SPV löschen des Anwenderspeichers */
#define  SB1_CLEAR_ZYKMEM_KUC            5     /* MMI -> ZST -> SPV löschen des Zyklenspeichers */
#define  SB1_CLEAR_PROG_KUC              7     /* MMI -> ZST -> SPV löschen eines Programms */
#define  SB1_ANF_PFELD_LISTE_KUC         8     /* MMI -> ZST P-Feld-Inhalte anfordern */
#define  SB1_PFELD_LISTE_KUC             9     /* MMI <-> ZST P-Feld-Inhalte */
#define  SB1_PFELD_ANZEIGE_KUC           10    /* MMI -> ZST P-Feld-Indices für Permanentanzeige */
#define  SB1_EINGABE_ENDE_KUC            11    /* MMI -> INT Eingabeanforderung an MMI abgeschlossen */
#define  SB1_VORLAUF_ENDE_KUC            12    /* INT -> MMI Satzvorlaufgrenze erreicht */
#define  SB1_VORLAUF_AM_ZIEL_KUC         13    /* INT -> MMI Satzvorlaufzielposition erreicht */
#define  SB1_QUITTUNG_AUF_EINZELFKT_KUC  14    /* ZST -> MMI Quittungsmeldung auf ASCII-Einzelfunktion */
#define  SB1_GET_ACHSINFO_KUC            15    /* MMI -> ZST Achsinformationen anfordern */
#define  SB1_ACHSINFO_KUC                16    /* ZST -> MMI Informationen zu einer Achse */
#define  SB1_CHANGE_REGPARAM_KUC         17    /* MMI -> ZST Online Änderung von Reglerparametern */
#define  SB1_PROGRAMM_START_KUC          18    /* MMI -> ZST Starten eines DIN-Programms */
#define  SB1_PROGRAMM_ENDE_KUC           19    /* ZST -> MMI Quittung auf SB1_PROGRAMM_START_KUC */
#define  SB1_PROGRAMM_ABBRUCH_KUC        20    /* MMI -> ZST laufendes Programm abbrechen */
#define  SB1_GET_PROGNAME_KUC            21    /* MMI -> ZST symbolischen Programmnamen ermitteln */
#define  SB1_PROGNAME_KUC                22    /* ZST -> MMI Quittung auf SB1_GET_PROGNAME_KUC */
```

```
/*--- SB1  bei Datenanforderung, Datentransfer oder Quittung auf Datentransfer ---*/
#define  SB1_NC_PROG_KUC                  1     /* MMI -> ZST -> SPV NC-Programm */
#define  SB1_ONLINE_PROG_KUC              2     /* MMI -> ZST -> SPV Onlineprogramme */
#define  SB1_WERKZEUG_KORR_KUC            3     /* MMI -> ZST Werkzeugkoordinaten */
#define  SB1_WERKSTUECK_KORR_KUC          4     /* MMI -> ZST Werkstückkoordinaten */
#define  SB1_ACHS_KORR_KUC                5     /* MMI -> ZST Achskoordinaten */
#define  SB1_TECHNO_KUC                   6     /* MMI -> ZST Technologie */
#define  SB1_MASCHINENKONSTANTEN_KUC      7     /* MMI -> ZST Maschinenkonstanten */
#define  SB1_KONFIG_KUC                   8     /* MMI -> ZST Konfigurationsdaten */
#define  SB1_PFELD_BLOCK_KUC              9     /* MMI <-> ZST P-Feld-Inhalte */
#define  SB1_IOMODULOFFSET_KUC            10    /* MMI <-> ZST Offsets der IO-Module */
#define  SB1_3D_KORREKTUR_KUC             11    /* MMI -> ZST 3D-Achskorrekturen */
#define  SB1_CAN1_OBJECT_KUC              12    /* MMI <-> ZST SDO Schreiben/Lesen über IO-CAN-Bus */
#define  SB1_CAN2_OBJECT_KUC              13    /* MMI <-> ZST SDO Schreiben/Lesen über Drive-CAN-Bus */


/*--- SB1  bei SB0 = SB0_CODESYS_KUC ---*/
#define  SB1_LZS_REQUEST_KUC              1     /* Codesys <-> LZS */
#define  SB1_LZS_ACKN_KUC                 2     /* Codesys <-> LZS */


/*--- SB1  bei SB0 = SB0_RESET_KUC ---*/
#define  SB1_RESET_KUC                    170   /* Auftrag zum Reset der Steuerung */


/*--- SB1  bei SB0 = SB0_EXCEPTION_KUC ---*/
#define  SB1_FEHLERNUMMER_KUC             1     /* Fehlermeldung an MMI */
#define  SB1_FEHLERQUITTUNG_KUC           2     /* Fehlerquittung vom MMI */
#define  SB1_FEHLERQUITTUNG_ALLE_KUC      3     /* alle Fehler auf einmal quittieren */


/*--- SB1  bei SB0 = SB0_DLL_SERVICE_KUC (bisher nicht implementiert)---*/
#define  SB1_EXECUTE_COMMAND_KUC          1     /* DOS-Befehl ausführen */
#define  SB1_OPEN_FILE_KUC                2     /* Datei öffnen und Handle zurückliefern */
#define  SB1_CLOSE_FILE_KUC               3     /* Datei schließen */
#define  SB1_READ_FILE_KUC                4     /* aus geöffneter Datei lesen */
#define  SB1_WRITE_FILE_KUC               5     /* in geöffnete Datei schreiben */
#define  SB1_DELETE_FILE_KUC              6     /* Datei löschen */
#define  SB1_FIRST_FILE_KUC               7     /* Erste n Dateien suchen */
#define  SB1_NEXT_FILE_KUC                8     /* Nächste n Dateien suchen */


/*--- SB1  bei SB0 = SB0_DEBUG_KUC ---*/
#define  SB1_TRACE_FLAGS_KUC              128   /* MMI -> NCR Setzen von Traceflags */
#define  SB1_TRACE_MELDUNG_KUC            129   /* NCR -> MMI Tracemeldung zum Debuggen des NCR */
#define  SB1_TRACE_SELECTIONS_KUC         130   /* MMI -> NCR auswaehlen von Tracelevels */
#define  SB1_DB_READ_KUC                  136   /* MMI <-> NCR Anforderung zum Lesen eines SPS-DB */
#define  SB1_DB_WRITE_KUC                 137   /* MMI -> NCR Anforderung zum Beschreiben eines DB */
#define  SB1_TRACE_ACHSDATEN_KUC          138   /* NCR -> MMI Achsdaten-Mitschrieb */


/*--- SB1  bei SB0 = SB0_EINGABE_KUC ---*/
#define  SB1_EINGABE_ABGEBROCHEN_KUC      128   /* MMI -> NCR G252-Eingabe abgebrochen */
#define  SB1_EINGABE_DEFAULTWERT_KUC      129   /* MMI -> NCR G252-Eingabe bestätigt */
#define  SB1_EINGABE_WERTGEAENDERT_KUC    130   /* MMI -> NCR G252-Eingabe Wert geändert*/


/*-----------------------------------------------------------------------------
 SB2 Optionale Steuerinformation gültig fuer die jeweils angegebene
     SB1-Untergruppe. Muss wenn nicht benutzt, auf 0 gesetzt werden.
-----------------------------------------------------------------------------*/

/*--- SB2  bei SB1 = SB1_CAN1_OBJECT_KUC, SB1_CAN2_OBJECT_KUC ---*/
#define  SB2_CAN_OPERATIONAL_KUC           0    /* MMI -> NCR Übertragung im Zustand Operational */
#define  SB2_CAN_PREOPERATIONAL_KUC        1    /* MMI -> NCR Übertragung im Zustand PreOperational */


/*---   eof: com_sbx.h   ---*/
```

2.4.4 mmictrl.h

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ACH@-*/
/*
*/
/*    LastUpdate  ( Created  16-05-02 14:31 )  by ebert        */
/*
**
**    PROJECT:   CNC21-Steuerung
**
**    VERSION:   V1.0                 $Revision:   1.3  $
**
**    MODULE DESCRIPTION:   mmictrl.h    ( HEADER (DEF) )
**    ---------------------------------------------------------------
**
**    Prototypes f r MMICTRL.DLL
**
**
**
**
**
**
**    (C) ECKELMANN INDUSTRIEAUTOMATION, WIESBADEN
**    ---------------------------------------------------------------
**    CREATED: 16-05-02 14:31   AUTHOR: ebert      V: V1.0
**-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-@ECH@-*/


/* $Header:   P:\cnc21\hmi\include\vcs\mmictrl.h_v   1.3   13 Feb 2003 13:45:06
ebert_0050046BD496  $ */

/*<f> History:
<--->*/



/*<f> hier: #define fuer den Process -----------------------------------*/

/* Modulnummern aller fehlermeldenden Module im PC */
#define  MOD_GTW_KC     1        /* MMIGTWAY.EXE */
#define  MOD_DLL_KC     2        /* MMICTRL.DLL */
#define  MOD_DDP_KC     17       /* CNCDPR55.EXE oder ET-PC-01 */


/* Parameter sMsg für Callback-Funktion (f r neue Applikationen) */
#define  VK_MMI_DOWNLOAD_STATE      1200  /* Firmware-Download-Statusmeldungen */
#define  VK_MMI_DOWNLOAD_PART       1201
#define  VK_MMI_DOWNLOAD_COMPLETE   1202
#define  VK_MMI_DOWNLOAD_ERROR      1203
#define  VK_MMI_TRANSFER_STATE      1204  /* Transferstatusmeldungen */
#define  VK_MMI_TRANSFER_OK         1205
#define  VK_MMI_TRANSFER_ERROR      1206
#define  VK_MMI_TRANSFER_BREAK      1207
#define  VK_MMI_NCMSG_SENT          1208  /* Quittung auf ncrPostMessage */
#define  VK_MMI_NCMSG_RECEIVED      1209  /* Nachricht vom NC */
#define  VK_MMI_ERROR_MSG           1210  /* Fehlermeldung */
#define  VK_MMI_NCMSG_NOT_SENT      1211  /* Quittung auf ncrPostMessage */
#define  VK_MMI_CYCLIC_CALL         1220  /* zyklischer Callback-Aufruf aus blockierenden
                                             Funktionen */

/* Transferstaties f r Dateitransfer (Ergebnis von ncrGetTransferState) */
#define  TRANSFER_OK                0     /* Transfer erfolgreich abgeschlossen */
#define  TRANSFER_WAIT             -1     /* Online Wait */
#define  TRANSFER_STATUS           -2     /* nur Statusmeldung */
#define  TRANSFER_BREAK             1     /* Transfer vom MMI oder NC abgebrochen */
#define  TRANSFER_ERROR             2     /* Zugriffsfehler aufgetreten, Abfrage mit
                                             WindowsAPI/GetLastError() */
#define  TRANSFER_TIMEOUT           3     /* Timeout */
#define  TRANSFER_NOINIT            4     /* Firmware-Download wurde noch nicht gestartet */
```

```
#define  TRANSFER_QFULL             5     /* Überlauf der Transferqueue */

/* Transferstaties f r Firmwaredownload (Ergebnis von ncrGetTransferState) */
#define  FWDL_OK                    0     /* Firmwaredownload erfolgreich durchgeführt */
#define  FWDL_WARMSTART            -1     /* NC-Rechner war bereis geladen */
#define  FWDL_STATUS               -2     /* nur Statusmeldung */


/*<f> hier: typedefs struct/union fuer den Process ----------------------*/

/*-------------------------------------------------------------------------*/
typedef void WINAPI (*MSGCALLBACK)(ULONG ulType, ULONG ulPara, void *pvContext);
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
typedef struct TRANSFER_STATE_TR { /* Transfer-Status für Neu-Applikationen */
/*-------------------------------------------------------------------------*/
   DWORD             dwFilePosLow;        // aktuelle Schreib-/Lese Position
   DWORD             dwFilePosHigh;
   DWORD             dwFileSizeLow;       // Dateigröße
   DWORD             dwFileSizeHigh;
   TCHAR             azName[MAX_PATH];    // vollständiger Name der zu übertragenden Datei
   TCHAR             azMeld[256];         // Statusinformation für Firmware-Download
}  TRANSFER_STATE_TR;


/*<f> hier: Exportierte Funktionen der DLL ----------------------------*/

HANDLE WINAPI _export ncrOpenControl ( char *pzName, MSGCALLBACK pCallback, void *pContext );
HANDLE WINAPI _export ncrOpenDefaultControl ( MSGCALLBACK pCallback, void *pContext );
void * WINAPI _export ncrGetDprAddress ( HANDLE hCnc, BOOL bShowErrors );
void WINAPI _export ncrCloseControl ( HANDLE hCnc );
void WINAPI _export ncrCloseAllControls ( void );

long WINAPI _export ncrGetInitState ( HANDLE hCnc );
short WINAPI _export ncrGetTransferType ( HANDLE hTrans );
long WINAPI _export ncrGetTransferState ( HANDLE hTrans, TRANSFER_STATE_TR *prState );
HANDLE WINAPI _export ncrLoadFirmware( HANDLE hCnc, char *pzConfig );
long WINAPI _export ncrLoadFirmwareBlocked( HANDLE hCnc, char *pzConfig );

HANDLE WINAPI _export ncrSendFile( HANDLE hCnc, char *pzFilename,
                                   char *pzHeader, int iSb1 );
long WINAPI _export ncrSendFileBlocked( HANDLE hCnc, char *pzFilename,
                                        char *pzHeader, int iSb1 );
HANDLE WINAPI _export ncrSendFileEx( HANDLE hCnc, char *pzFilename, DWORD dwOffsetLow,
                                     DWORD dwOffsetHigh, char *pzHeader, int iSb1 );
HANDLE WINAPI _export ncrReceiveFile( HANDLE hCnc, char *pzFilename, int iSb1 );
long WINAPI _export ncrReceiveFileBlocked( HANDLE hCnc, char *pzFilename, int iSb1 );
long WINAPI _export ncrWaitTransfer ( HANDLE hTrans );
BOOL WINAPI _export ncrCloseTransfer ( HANDLE hTrans );

BOOL WINAPI _export ncrPostMessage ( HANDLE hCnc, MSG_TR *prMsg, int iRespKey );
BOOL WINAPI _export ncrSendMessage ( HANDLE hCnc, MSG_TR *prMsg );
BOOL WINAPI _export ncrReadParamArray ( HANDLE hCnc, WORD awIdx[], double adVal[], WORD wCount
);
BOOL WINAPI _export ncrWriteParamArray ( HANDLE hCnc, WORD awIdx[], double adVal[], WORD
wCount );


/*---   eof: mmictrl.h   ---*/
```