PROCESS AUTOMATION

Freelance 2019

Reference Manual
DMS / API

PROCESS AUTOMATION

Freelance 2019

Reference Manual
DMS / API

Document Number: 3BDD012508-111
Revision: A
Release: January 2019

Notice

This document contains information about one or more ABB products and may include a description of or a reference to one or more standards that may be generally relevant to the ABB products. The presence of any such description of a standard or reference to a standard is not a representation that all of the ABB products referenced in this document support all of the features of the described or referenced standard. In order to determine the specific features supported by a particular ABB product, the reader should consult the product specifications for the particular ABB product.

ABB may have one or more patents or pending patent applications protecting the intellectual property in the ABB products described in this document.

The information in this document is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this document.

Products described or referenced in this document are designed to be connected, and to communicate information and data via a secure network. It is the sole responsibility of the system/product owner to provide and continuously ensure a secure connection between the product and the system network and/or any other networks that may be connected.

The system/product owners must establish and maintain appropriate measures, including, but not limited to, the installation of firewalls, application of authentication measures, encryption of data, installation of antivirus programs, and so on, to protect the system, its products and networks, against security breaches, unauthorized access, interference, intrusion, leakage, and/or theft of data or information.

ABB verifies the function of released products and updates. However system/product owners are ultimately responsible to ensure that any system update (including but not limited to code changes, configuration file changes, third-party software updates or patches, hardware change out, and so on) is compatible with the security measures implemented. The system/product owners must verify that the system and associated products function as expected in the environment they are deployed.

In no event shall ABB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license. This product meets the requirements specified in EMC Directive 2014/30/EU and in Low Voltage Directive 2014/35/EU.

Trademarks

All rights to copyrights, registered trademarks, and trademarks reside with their respective owners.

Table of Contents

4 - Name management

Appendix C - DMSAPI files

Index

About this book

Use of warning, caution, information, and tip icons

This publication includes **Warning**, **Caution**, and **Information** where appropriate to point out safety related or other important information. It also includes **Tip** to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

Electrical warning icon indicates the presence of a hazard which could result in *electrical shock.*

Warning icon indicates the presence of a hazard which could result in *personal injury.*

Caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard which could result in *corruption of software or damage to equipment/property.*

Information icon alerts the reader to pertinent facts and conditions.

Tip icon indicates advice on, for example, how to design your project or how to use a certain function

Although **Warning** hazards are related to personal injury, and **Caution** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all **Warning** and **Caution** notices.

Terminology

The Glossary contains terms and abbreviations that are unique to ABB or have a usage or definition that is different from standard industry usage. Please make yourself familiar to that.

You will find the glossary at the end of the ***Engineering Manual System Configuration.***

Document conventions

The following conventions are used for the presentation of material:

- The words in names of screen elements (for example, the title in the title bar of a window, the label for a field of a dialog box) are initially capitalized.
- Capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, press the ENTER key.
- Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the **space bar**, **comma key**, and so on.
- Press CTRL+C indicates that you must hold down the CTRL key while pressing the C key (to copy a selected object in this case).
- Press **ESC, E, C** indicates that you press and release each key in sequence (to copy a selected object in this case).
- The names of push and toggle buttons are boldfaced. For example, click **OK**.
- The names of menus and menu items are boldfaced. For example, the **File** menu.
  - The following convention is used for menu operations: MenuName > MenuItem > CascadedMenuItem. For example: select **File** > **New** > **Type**.
  - The **Start** menu name always refers to the **Start** menu on the Windows Task Bar.

- System prompts/messages are shown in the Courier font, and user responses/input are in the boldfaced Courier font. For example, if you enter a value out of range, the following message is displayed:

  ```
  Entered value is not valid. The value must be 0 to 30.
  ```

  You may be told to enter the string TIC132 in a field. The string is shown as follows in the procedure:

  TIC132

  Variables are shown using lowercase letters.

  sequence name

1 Application interface to Freelance for Windows

1.1 Overview

DMS/API

stands for Digimatik Message Specification and Application Programmable Interface.

DMS is a subset of Manufacturing Message Specification, as per ISO 9506 (MMS).

The computer running the application is connected to the Freelance Control Net (Ethernet) and, under the DMS/API, uses communication facilities in the same way as Freelance does internally, with all the possibilities that Freelance provides.

DMS/API

- Is the application interface for communicating directly with Freelance from a user program on an external computer (host);
- Is a library of functions programmed in “C” that runs under Windows
- When linked to a programmed application, provides all necessary higher-level-language commands for simple and rapid data exchange between the application and the Freelance system;
- Is implemented on the client-server model. The application station appears as a gateway in the Freelance Engineering, which makes data from the configured measuring points available for use by the application via the download function.
- The DMS/API setup program is distributed on two diskettes or a CD; its installation is menu-driven in the same way as that of all Freelance products.

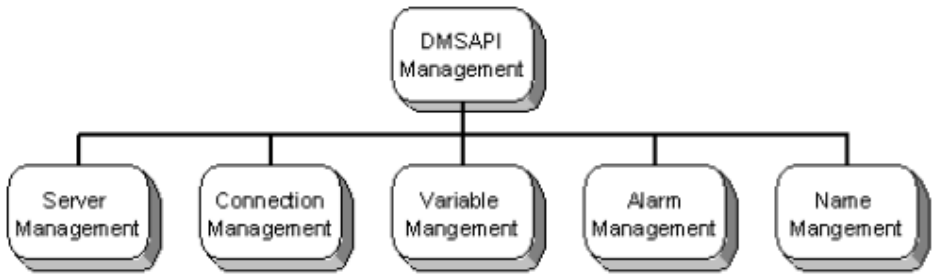
DMS is used in the following Freelance applications:

- Freelance Engineering
- Freelance Operations

- Freelance process stations
- Freelance CSO gateway
- Freelance OPC gateway

External API connections are configured and loaded as gateways in Freelance Engineering. Thereafter, all Freelance measuring point addresses are available to all gateway stations.

The DMS/API can be subdivided into the following function areas:



Flow chart.bmp

1.2 Manufacturing message specification ISO 9506 (MMS)

MMS is a standard, designed to eliminate communication difficulties among computers in industrial automation. MMS was approved as an international norm in 1988. It grew out of an initiative called Manufacturing Automation Protocol (MAP) started by General Motors in the early 1980s.

MMS fits in the top layer of the OSI 7-layer communications model, the application layer.

Application layer (MMS)
Presentation layer
Session layer
Transport layer
Network layer
Control layer
Physical layer

Applications:

- Process control technology
- Memory-programmable control applications
- Numeric control applications
- Robots

MMS attempts to formulate the messages and requests that must be exchanged between the different system components in a heterogeneous computer network in a standardized vocabulary.

MMS accomplishes this on a client-server model. A client presents a request to a server. The server processes the request and sends a response back to the client.

The MMS specification describes all requests which a server must understand and process. For this purpose objects are defined which can be operated. Operations on

these objects de-scribe what these objects are meant for. There are a total of 16 different types of objects and 79 operations defined under MMS.

1.3 Digimatik message specification (DMS)

DMS is only a partial implementation of MMS. A pragmatic approach was taken in the creation of DMS:

- What are the communications requirements?
- Which MMS services are required to fulfill them?

Freelance is a classical client-server system. The PC (as engineering station or operator station) functions as client and sends requests (measurement, open- and closed- loop control) to the process modules.

A logical communications channel is open between client and server. Both sides recognize when the connection is broken and when it is reestablished after a breakdown.

Programs are written on the engineering PC which are translated into executable code whose contents is to be loaded into cards at the process station.

These downloaded programs are run on the process stations, meaning that they can be started and stopped from the PC.

The downloaded programs perform measurements, open- and closed- loop control tasks. The data that this process produces is displayed or archived on the PC.

The user can actively access and change process values.

Limits which are exceeded as the process proceeds will be immediately displayed on the PC in address form.

1.4 DMS/MMS function areas

Environment and general management services offer services relating to the administration and handling of connections. In the Freelance system, connection management has been optimized to the requirements of a distributed control system.

Domain management services are services relating to loading and managing of program and data areas. The following services are implemented in DMS:

- Download (InitiateDL, DownloadSegment, TerminateDL)
- Upload (InitiateUL, UpLoadSegment, TerminateUpLoad)
- DeleteDomain

Services for domain management are not implemented in the DMS API.

Program Invocation Management Services are services relating to the creation, starting, stopping, resetting and deleting of programs.

The DMS implementation has been spared the services CreatePI and DeletePI. These func-tions are handled automatically through Download and Domain Delete (TaskDomains).

- StartPI
- StopPI
- ResetPI

Variable access services allow reading and writing variables from the process currently running. DMS has implemented the following variable access services:

- Read
- Write
- Define Named Variable List
- Information report; the server sends this report “without being asked” and without requiring a DMS-layer acknowledgment. In Freelance the reports are used for long-term archiving data curves, for disturbance course logs and short-term updating of value windows/trend curves and all Freelance Operations graphics.

Event management services offer event-driven services such as alarms and acknowledgements:

- GetAlarmSummary
- EventNotification
- Acknowledge EventNotification

Journal management is concerned with storing and consulting stored information. Freelance has the following configurable function blocks for this purpose:

- Trend function block
- Signal sequence log

The information in these function blocks can be read using the variable access services.

Freelance Name management provides coordinated access to valid variable and tag names in Freelance as well as transformation to Freelance addresses.

1.5 Freelance addressable objects

1.5.1Variables

- Predefined variables (project independent)
- User-defined variables
- User-defined structure variables
- Curves and disturbance course logs
- Tag variables

All variables are addressed with the following values:

- Station number
- Object number
- Component number
- Type of variable

The transformation of variable names Freelance addresses is handled by the name management functions.

1.5.2 Allocated tags

- Function blocks
- Sequence control

- MSR tasks
- MSR program lists
- MSR IPC programs

Allocated tags are addressed as follows:

  – Station number
  – Object number
  – Class number

1.5.3 System objects

- MSR resources

1.6 Freelance layered communications model

Application layer (DMS)
Layer 6 (not used)
Basis Transport
TCP/UDP
IP-Protocol
CSMA/CD procedure
Physical layer

In Freelance the communications layer is divided into a DMS (Freelance message specification) layer and a Basis transport (BTR) layer. Communication from a client application over the network to a server can be depicted as follows:

Client applications -> DMS ->BTR ->TCP/IP ->cable ->TCP/IP ->BTR-> DMS -> Server

While the DMS layer is programmed in operating-system-independent C code, the BTR layer is operating system dependent and available for the following platforms:

- PSOS
- WINDOWS

The various communications tasks are managed in the operating-system-dependent communications section:

Under PSOS/NT, for example, these are the following tasks:

- ListenTask (only used if the station functions as a server)
- SendTasks (waits on MailBox for send requests)
- ReceiveTasks (waits on Socket for incoming data)
- UDP tasks:
  - Send/receive cyclic variable lists
  - Time synchronization
  - Connection establishment

The operating-system-dependent communications layer is unaware of the structure of the communications packets, the station numbers and of other information.

To port the system to another operating system, only this layer, together with memory and se-maphore management functions, need be re-implemented.

1.7 DMS/API installation

The DMS/API is distributed as a setup program on two diskettes; its installation on the application computer is menu driven, as is that of all Freelance products.

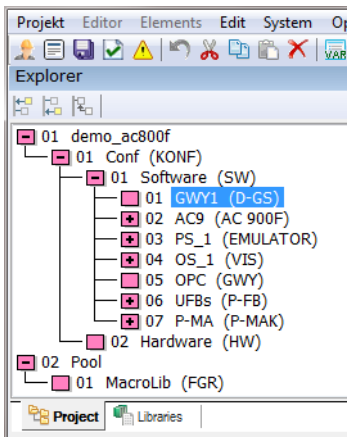
If the DMS/API is to be installed on the same machine as Freelance Engineering, then the DMS/API must be loaded in the Freelance standard directory.

The DMS/API DLLs must be in the same directory as Freelance.

1.8 Configuring the DMS/API gateway in Freelance Engineering

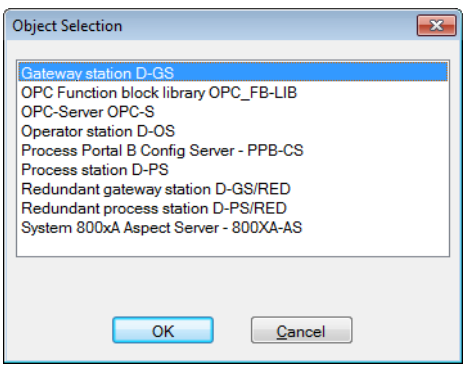
Every gateway in the project tree must be entered and configured in Freelance Engineering so that address information will be available.

See ***Engineering Manual System Configuration***.



project_tree_us.png

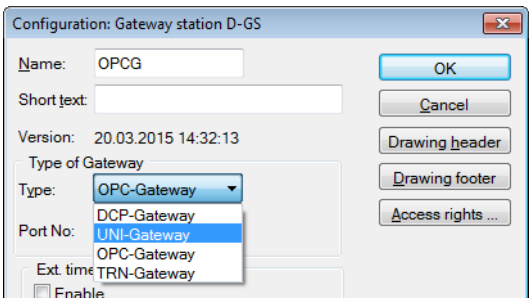
If under Configuration a new DMS/API resource is to be configured, then “Gateway station” must be chosen as the station type.



opject_selection_us.png

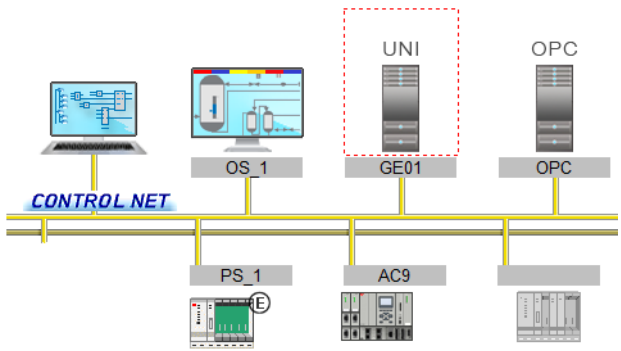
The type of gateway can be set by editing fields in the gateway header dialog box:

- DCP gateway
- UNI gateway (for own applications using the API)
- OPC gateway
- TRN gateway
-



gateway_type_us.png

In the Hardware structure, the gateway is shown as a PC, because, from the point of view of the Process station, it looks like an Operator station.

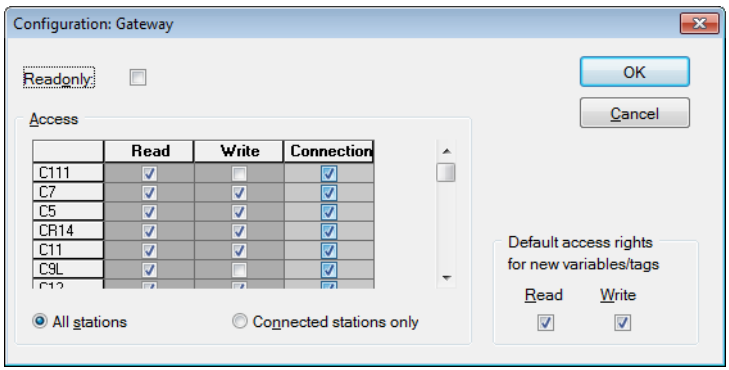


hw_struc_us.png

With which Process stations the gateway is to perform read/write services is configured after the gateway is entered.

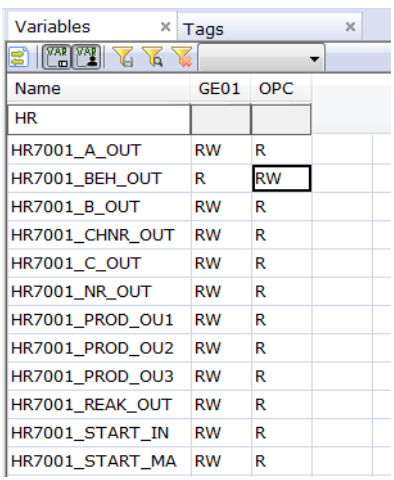
In addition, the variables and tags for which the gateway is to receive address information must be laid down in the configuration. Beside the normal view, there

is, associated with the variables and tags, a gateway view is set to read and/or write flags for each variable and tag.



conf_gateway_us.png

Variable List



conf_gateway_us.png

Tag List

Variables | Tags

Name	GE01	OS_1	OPC
HR			
HR7001_A	RW	X	RW
HR7001_B	RW	X	RW
HR7001_C	RW	X	RW
HR7001_CHNR	RW	X	RW
HR7001_NR	RW	X	RW
HR7001_TEMP	RW	X	RW
HR7002_A	RW	X	RW
HR7002_B	RW	X	RW
HR7002_C	RW	X	RW
HR7002_CHNR	RW	X	RW
HR7002_NR	RW	X	RW
HR7002_TEMP	RW	X	RW

msr_list_us.png

1.9 Loading the DMS/API gateway

1.9.1Initial configuration

The Process stations should always be loaded before the gateways. Once the Process stations have been loaded, the Gateway stations are then able to access the Freelance address information.

1.9.2 Re-configuration

When the process stations are reconfigured, the changed objects must be loaded on Process stations and gateways. The Process station does not know either the variable names or the tag names, but only the DMS addressing with the object number and component number.

When re-configuration is performed in an ‘inexpert’ manner it is two possible that two objects may exchange object numbers (when objects are deleted and re-inserted). In such cases, write access through the gateway to the process station (after the process station has been loaded and before the gateway is loaded) can produce undesirable results.

If objects are only being added to a process station or changed, and no objects are being deleted or moved, then the addresses of the old objects remain unchanged. In this case the gateway only receives information on new objects. There is no danger of mal-operation by the gateway.

This should be taken into account when creating a custom DMS/API application

Examples of possible solutions are as follows:

- The DMS/API gateway can be put into configuration mode through user intervention before the process stations are loaded
- If the DMS/API gateway is re-initialized from within Freelance Engineering before the process station is loaded, the application that is waiting to perform a write operation may respond to that.
- Version control can be activated:
  - Read accesses are always allowed
  - Write accesses are permitted only when the versions are the same.
- After the gateway has been re-configured, the custom database should be checked and, if necessary, read accesses and Alarm Summary should be re-applied.
- The DMS/API application is informed by Freelance Engineering through Callback functions each time the gateway is reconfigured.

1.10 DMS/API function overview

Client management, Environment and general management services

Function	Description
DMSAPI_Init	Initialize a DMS session
DMSAPI_Exit	Terminate a DMS session
DMSAPI_ConnectByName	Establish connection to a DMS server
DMSAPI_ConnectByAddr	Establish connection to a DMS server
DMSAPI_ConnectByNo	Establish connection to a DMS server
DMSAPI_Disconnect	Terminate connection to a DMS server
DMSAPI_GetConnectionData	Check connection to a DMS server
DMSAPI_SetSystemTime	Set time in Freelance system
DMSAPI_SetSystemTimeBy-DmsType	Set time in Freelance system
DMSAPI+_SetSystemTimeBy-String	Set time in Freelance system
DMSAPI_RestartResource	Warm start, cold start, or toggle a Freelance station

Variable access services

Function	Description
DMSAPI_VLCreate	Create variable list
DMSAPI_VLAddWriteVarByName	Add write variable
DMSAPI_VLAddReadVarByName	Add read variable
DMSAPI_VLAddWriteVarByAddr	Add write variable
DMSAPI_VLAddReadVarByAddr	Add read variable
DMSAPI_VLChangeValue	Change value within variable list
DMSAPI_VLDelVar	Delete variable from variable list

Function	Description
DMSAPI_VLCreate	Create variable list
DMSAPI_VLClear	Delete all variables from a variable list
DMSAPI_VLRead	One-time read of variable list
DMSAPI_VLReadCycle	Cyclical read of variable list
DMSAPI_VLWrite	One-time write to a variable list
DMSAPI_VLStopCycleVar	Stop cyclical variable list
DMSAPI_VLDelete	Delete variable list

Event Management Services

Function	Description
DMSAPI_GetAlarmSummary	Request alarm summary of a DMS server. All future alarms that occur will automatically be sent by this server.
DMSAPI_AckAlarmByList	Perform acknowledgment with fully filled out list

Client receive

Function	Description
DMSAPI_RegisterClientCB	Register user-programmed callback function which will be called when DMS messages for the client are received.
API_CallbackReceive	Callback receive function will be called asynchronously when DMS messages are received

Freelance name management

Function	Description
DMSAPI_SetProjectDir	Set project path for loading/saving configuration information
DMSAPI_ChangeProject	Switch to another project
DMSAPI_LockOV	Lock Name Management against re-configuration by Freelance Engineering
DMSAPI_UnlockOV	Remove the lock
DMSAPI_GetProjectInfo	Get current project version
DMSAPI_GetFirstResourceInfo	Get configuration information for first station
DMSAPI_GetNextResourceInfo	Get configuration information for all remaining stations
DMSAPI_GetFirstVarInfo	Get configuration information for first variable
DMSAPI_GetNextVarInfo	Get configuration information for all remaining variables
DMSAPI_GetFirstTagInfo	Get configuration information for first tag
DMSAPI_GetNextTagInfo	Get configuration information for all remaining tags
DMSAPI_GetTagByAddr	Get configuration information for a particular tag
DMSAPI_GetFirstCmpOfObjCls	Get configuration information for the first component of an object class
DMSAPI_GetNextCmpOfObjCls	Get configuration information for all remaining components of an object class
DMSAPI_GetVarnameByOPath	Transform variable name into Freelance address information
DMSAPI_GetVarInfoByName	Transform Freelance address information into variable name

DMS utilities

Function	Description
DMSAPI_SetVarCode	Set variable transformation formats
DMSAPI_GetValueByString	Transform: string DMS value
DMSAPI_GetStringByValue	Transform: DMS value string
DMSAPI_GetVarLen	Length required by a variable within a variable list
DMSAPI_DumpRecData	Outputs the receive data structure to STDOUT

2 Basic transport application interface (BTR)

The basic transport layer is protocol-independent. It is possible to set up both with the P protocol (for AC 870P / Melody) and with the Freelance protocol.

This section can be skipped if the DMS/API is running under an operating system in which the BTR layer is already implemented (PSOS, Windows). On other operating systems the BTR layer must be implemented from scratch.

The BTR layer must be initialized by the calling applications. (In this case it is the DMS/API using the application rather than the application using the DMS/API).

The initialization routine is named

BTR_Init

Before it shuts down, the application should call routine:

BTR_Exit

The BTR layer links a client application with a server application Within the BTR layer, each connection is identified by a unique ConnectionHandle.

The applications provide the BTR layer with three “Callback functions”:

AbortProc, is called up when a connection is interrupted. The ConnectionHandle and the reason for the interruption to the connection are passed to the function as parameters.

KeepAliveProc is called up when no send instruction is received within a timeout. Along with the ConnectionHandle, the parameters for passing take the form of a buffer into which the (log-specific) packet for monitoring the connection must be coded.

ReceiveProc, is called up when data arrive on a connection. Along with the ConnectionHandle, the parameters for passing take the form of a buffer holding the data.

2.1 Server functionality (TCPIP)

The server application calls a procedure called “BTR_OpenServer”.

When it does this, the server application passes the three Callback functions to the BTR layer.

Calling the OpenServer procedure starts a task (PSOS) or thread (Windows) that waits for a client to try to establish a connection.

Each time a connection is established, two further tasks/threads are started:

- Send
- Receive

3 DMS client management

All functions which initiate an action on the Ethernet or are waiting for an event on the Ethernet have a SyncFlag and a TimeOut. The procedure returns when the timeout expires if not before. The SyncFlag can take the following values:

- Synchronous with Receive (if the response does not arrive within the time interval, then it must be collected using Receive)
- Synchronous with Callback (if the response does not arrive within the time interval, then the Callback function is called)
- Asynchronous with Receive (the timeout only applies for sending the message)
- Asynchronous with Callback (the timeout only applies for sending the message)

All procedures with a response are identified by the symbol:

The responses are described in the section “Receiving/decoding data”.

Many procedures have the following error:

E_DMSAPI_INTERNAL_ERROR

This means that the DMS has encountered an unrecognized error, for example:

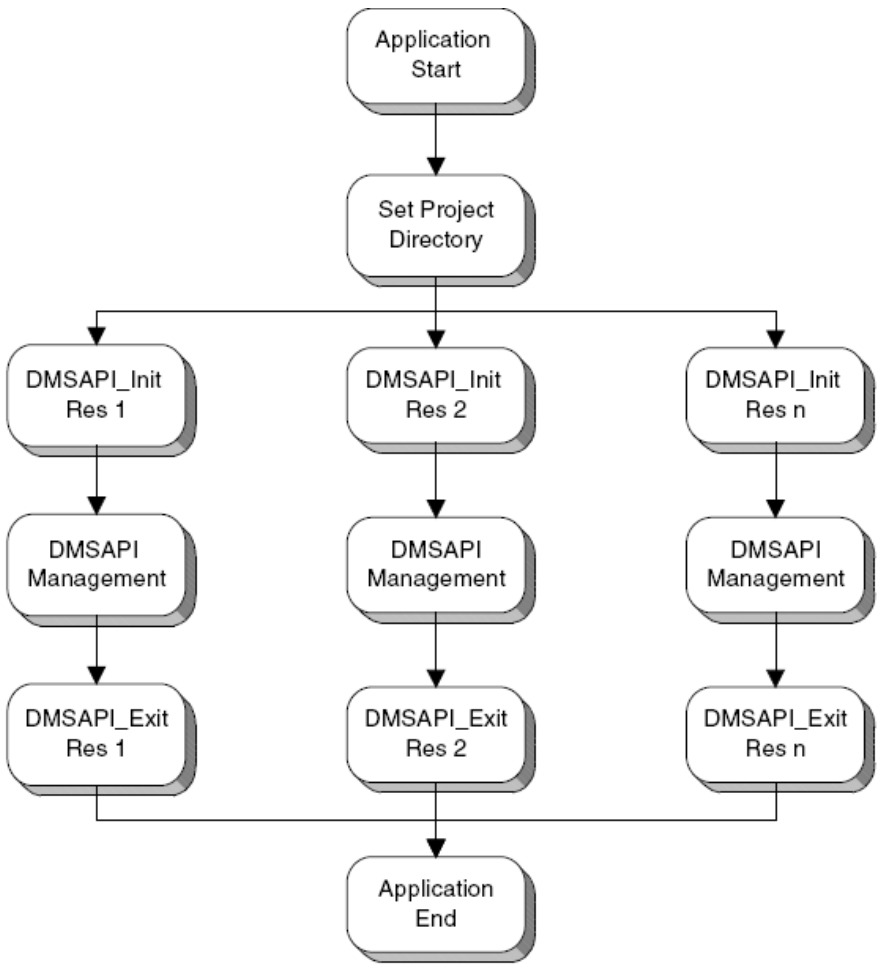
- The application has written to data in the DMS with an uninitiated pointer
- For inexplicable reasons the TCPIP is no longer operational.

The application should be exited and re-started in a way that will affect the user data as little as possible.

3.1 Environment and general management services

3.1.1 Initializing and terminating a DMS session

The DMSAPI is capable of “multi-projecting”, that is a DMSAPI application can be entered as a gateway with different resource numbers in more than one Freelance project. Each Freelance Engineering can download the configuration for “its” gateway. The DMSAPI application can access the various resources and objects from the different projects through those projects.



ap010us.bmp

DMSAPI_Init

SYNTAX

```
DMS_RC DMSAPI_Init (
    DMS_RES_NOOwnResNo/* Own Resource No */,
    DMS_RES_TYPEOwnResType/* Own Resource Type */,
    DMS_INT16          NoOfSrvConn/* Number of ServerConnection */,
    DMS_BOOLEANbStandardServer/* use of StandardServer */
)
```

Initializes the DMS application layer. The parameters passed are the function's own resource number and the number of server connections which exist simultaneously. If the application to be written by Freelance Engineering is to be provided with the address information, then the same resource number should be chosen as was assigned to the gateway in Freelance Engineering. The number of server connections should be set to 1.

Each resource can only manage one project at a time. If the name management system of several projects is accessed through the DMSAPI at the same time, then the initialization routine should be called several times with different resource numbers. In the various different projects the gateway must then also be configured with these different resource numbers.

If a custom DMSAPI server application is to be written, the parameter bStandardServer should be set to FALSE. In this case the procedures from Section 6, DMS utilities must be used.

If there is no requirement for server functionality, NoOfServerConn is set to 0.

In the Environment and general management services section of DMSDEF.H the following definitions can be found:

- DMSAPI_MAX_APPLICATION
- DMSAPI_MAX_CONNECTION

Parameters:

- OwnResNo Resource number of its own station within the Freelance system. Logical DMS connections always exist between two resources (the value lies

between 1 and 255). More than one logical resource can be initialized on a computer. They must be assigned different resource numbers.

- OwnResType own resource type
  - DMS_OS_DIGIVIS
  - DMS_OS_DIGITOOL
  - DMS_OS_EPROM
  - DMS_OS_MSR
  - DMS_OS_DDE_GWY
  - DMS_OS_P_GWY
  - DMS_OS_GWY (this type is generally used for DMSAPI applications)
- NoOfSrvConn Number of possible server connections
- bStandardServer:
  - TRUE: Application can be employed as a server for Freelance Engineering => Freelance Engineering can download the address information onto the server. Name management is activated by loading.
  - FALSE: Application can implement its own server functions

Possible return values:

Function	**Description**
E_DMSAPI_ALREADY_INIT	The function was called although the DMS layer for this resource number was already initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INTERNAL_ERROR	Error during initialization of DMS layer.
E_DMSAPI_MAX_CONNECTION	The number of server connections exceeds the number of possible connections.
E_DMSAPI_MAX_APPLICATION	The function cannot be called by more than the maximum number of applications.

DMSAPI_Exit

SYNTAX

```
DMS_RC  DMSAPI_Exit (
    DMS_RES_NO OwnResNo  /* Own Resource No */
)
```

Resource exits the DMS application layer. All DMS objects attached to this resource (connections, variables,...) are cleared.

Parameters:

- OwnResNo: Resource number of its own station within the Freelance system. Logical DMS connections always exist between two resources (Value lies between 1 and 255).

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INTERNAL_ERROR	Error on exiting the DMS layer.

3.1.2 Connection management

Overview of procedure to establish a connection between client and server station

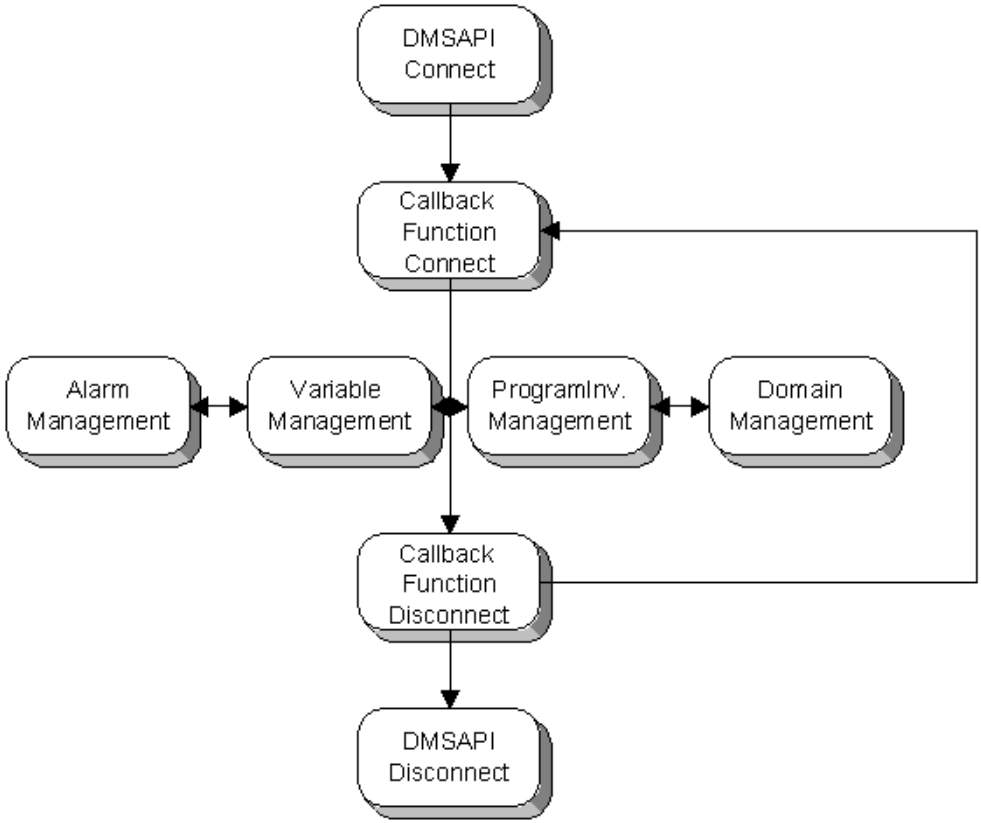
- Through UDP the client station sends a DMSNameServerRequest to the two IP addresses passed to it, and a fixed UDP port.
- The two server stations send back a DMSNameServerResponse. The information contained in this response is as follows:
  - Station - primary or secondary
  - Station - ControlPortNumber, on which the TCPIP connection can be established

- Client station executes a Connect to the transmitting ControlPort for the (first-to-respond) primary.
- The server station which has waited at this ControlPort with a “List” causes the connection to be established.
- The client station sends a DmsInit command on the opened connection, in which it conveys the following details:
  – Port number of the UDP port to which the cyclical variable lists are to be sent.
  – Timeout that is to be used to monitor the connection
  – Current DMS version number
  – Own resource number
- In response to this DMSInit command the server station sends a DMSInit-Response
  – Own resource number
  – Own resource type
  – Version number
- The client station checks the returned values and assigns the status to the application’s Callback functions

Viewing a DMSAPI application on the process station

The Callback functions are called automatically by the DMSAPI layer when a connection is established or released.

The functional areas ProgramInv and Domainmanagement are only used by Freelance Engineering.



ap014us.bmp

DMSAPI_ConnectByAddr

Icon.bmp

SYNTAX

```
DMS_RC DMSAPI_ConnectByAddr(
    DMS_RES_NO      OwnResNo    /* Own Res No */,
    DMS_INT16       nBTRLnk     /* BasicTranspLayer */,
    DMS_UINT32      ulIPAddr1   /* 1.IPAddr. Res */,
```

```
DMS_UINT32          ulIPAddr2      /* 2.IPAddr. Res */,
DMS_RES_NO          ResNo          /* Resource No */,
DMS_RES_TYPE        ResType        /* Resource Type */,
DMS_UINT16          uKeepAliveT    /* KeepAliveTimeout */,
DMS_CONN_HANDLE*lpConnHandle /* ConnectionHandle */,
DMS_INT16           nSyncFlag      /* Synchronicity Flag */,
DMS_UINT32          ulProcT        /* ProcedureTimeout */,
DMS_UINT32          ulRecConnLen   /* Size of storage area referenced to
                                                          the pointer */,
DMS_REC_CONN_DATA   *RecConn       /* RecStruct of Conn. */
)
```

Establishing a connection to a DMS server station: The return value is a connection handle that can be used later to identify the connected resource. This enables more than one connection to be established to a resource. Thus, for instance, one connection can be used for alarm messages, and another for updating or operation. It is also possible to perform all the different services on a single connection. If the server station has been configured with redundancy, then the connection to the active station will be opened automatically. The connection between client and server station is monitored. The connection is monitored with the parameter “KeepAliveTimeout”. If the synchronisation flag is set to DMSAPI_SNYC and if the connection is established within the declared procedure timeout, then the connection structure is filled with values.

After an interruption to a connection the connection is re-established automatically. If the application no longer needs this connection the procedure DMSAPI_Disconnect should be called.

Parameters

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- nBTRLnk: Shows the BTR layer via which the DMS services are being transferred.

  DMS_BTR_TCPIP (

DMS_BTR_REDLNK (Redundancy link on process station)

- ulIPAddr1: IP address of server station
- ulIPAddr2: 2. IP address of server station if configured with redundancy.

If the server station is not configured with redundancy, 0 is entered here.

- ResNo: Resource number of server station. If more than one resource number is installed on the server station, the connection is established to the correct server.
- ResType: Resource type of server station. The following values are valid:
  - DMS_OS_DIGIVIS
  - DMS_OS_DIGITOOL
  - DMS_OS_EPROM
  - DMS_OS_MSR (usually connected to this station type)
  - DMS_OS_DDE_GWY
  - DMS_OS_P_GW
  - DMS_OS_GWY
- uKeepAliveT: Connection monitoring in seconds/milliseconds between the two resources, that is an interruption to the connection (e.g. cable problem, failure of the connected resource etc.) is detected when the timeout expires at the latest. If the two resources are not sending any data, then within half the timeout they exchange a KeepAlive packet.
- lpConnHandle: After the connection has been established, data exchange on this connection is addressed via this ConnectionHandle.
- nSyncFlag

DMSAPI_SYNCHRON: The procedure waits for the length of time specified by “Procedure-Timeout” for the connection to be established. If the connection is established successfully, RecStruct returns valid values. The process of establishing a connection also continues after the timeout has expired.
DMSAPI_ASYNCHRON: When a connection is established, this is indicated through the Callback function, or it may be read via the function DMSAPI_GetConnectionStatus. The procedure timeout is not evaluated.

- ulProcT:

DMSAPI_NO_TIMEOUT no timeout value in milliseconds.

DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.

- ulRecConnLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used. Then it has the same length as the structure DMS_REC_CONN_DATA
- RecConn:

```
typedef struct DMS_REC_CONN_DATA
{DMS_RES_NO        OwnResNo;        /* Own resource ID */
 DMS_RES_NO        ResNo;           /* Resource ID of station */
 DMS_RES_TYPE      ResType;         /* Resource type of server station */
DMS_CONN_STATUS    ConnStatus;      /* Connection status of station */
DMS_UINT32         ulIPAddr;        /* IP address of conn. station */
DMS_UINT32         ulBoardType;     /*  BoardType */
DMS_UINT32         ulConnFlag;      /*  */
} DMS_REC_CONN_DATA;
```

The ConnStatus can take the following values:

```
DMS_CONN_OK,                          /* All OK */
DMS_CONN_ABORT,                       /* No connection */
DMS_CONN_INVALID_RES_TYPE, /* Incorrect resource type */
DMS_CONN_INVALID_RES_NO, /* Incorrect resource number */
DMS_CONN_NO_OS,                /* No operating system */
DMS_CONN_SECONDARY,       /* Only connected to secondary
                             => invalid configuration */
DMS_CONN_INVALID_VERSION /* Invalid DMS version */
```

The ulBoardType can take the following values:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

The ulConnFlag can take the following values:

```
DMS_RES_PRIMARY     /* Connection to a primary server */
DMS_RES_SECONDARY   /* Connection to a secondary server */
DMS_RES_CLIENT      /* Connection to a client */
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialised.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_TIMEOUT	It was not possible to establish a connection within the specified timeout.
E_DMSAPI_INTERNAL_ERROR	Internal error on establishing connection
E_DMSAPI_MAX_CONNECTION	The DMS layer does not allow more than the maximum number of connections.

DMSAPI_ConnectByName

SYNTAX

```
DMS_RC DMSAPI_ConnectByName (
DMS_RES_NO          OwnResNo        /* Own Resource No */,
DMS_CHAR            *ResName        /* Name of resource  */,
DMS_CONN_HANDLE     *lpConnHandle   /* ConnectionHandle */,
DMS_INT16           nSyncFlag       /* Synchronisation flag */,
DMS_UINT32          ulProcT         /* Procedure timeout */,
```

```
DMS_UINT32              ulRecConnLen        /* Size of storage area
                                               referenced to the pointer */,
DMS_REC_CONN_DATA   *RecConn                /* RecStruct of Conn. */
)
```

The procedure establishes the connection to a DMS resource. The project loaded by Freelance Engineering must be accessible to its own resource. Through the name management functions the resource name is converted to:

- BTRLnk
- IPAddr1
- IPAddr2
- KeepAliveTimeout
- ResourceNo
- ResourceType

Icon.bmp

Establishing a connection to a DMS server station: The return value is a connection handle that can be used later to identify the connected resource. This enables more than one connection to be established to a resource. Thus, for instance, one connection can be used for alarm messages, and another for updating or operation. It is also possible to perform all the different services on a single connection. If the server station has been configured with redundancy, then the connection to the active station will be opened automatically. The connection between client and server station is monitored. The connection is monitored with the parameter KeepAliveTimeout. If the synchronisation flag is set to DMSAPI_SNYC and if the connection is established within the declared procedure timeout, then the connection structure is filled with values.

After an interruption to a connection the connection is re-established automatically. If the application no longer needs this connection the procedure DMSAPI_Disconnect should be called.

Parameters

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- ResName: Name of the target resource as configured in Freelance Engineering.
- lpConnHandle: After the connection has been established, data exchange on this connection is addressed through this ConnectionHandle.
- nSyncFlag

DMSAPI_SYNCHRON: The procedure waits for the length of time specified by "ProcedureTimeout" for the connection to be established. If the connection is established successfully, RecStruct returns valid values. The process of establishing a connection also continues after the timeout has expired.

DMSAPI_ASYNCHRON: When a connection is established, this is indicated through the Callback function, or it may be read through the function DMSAPI_GetConnectionStatus. The procedure timeout has no significance.

- **ulProcT:**
  DMSAPI_NO_TIMEOUT no timeout value in milliseconds.

DMSAPI_WAIT_FOREVER:

Procedure will not return until the task is executed, or proves impossible to execute.

- **ulRecConnLen**: only used when the synchronisation flag DMSAPI_SYNCHRON is being used. Then it has the same length as the structure DMS_REC_CONN_DATA
- RecConn:

```
typedef struct DMS_REC_CONN_DATA {
        DMS_RES_NO      OwnResNo;     /* Own station number */
        DMS_RES_NO      ResNo;        /* Station number */
        DMS_RES_TYPE    ResType;      /* Resource type of server station
        DMS_CONN_STATUS ConnStatus;   /* Connection status of station
        DMS_UINT32      ulIPAddr;     /* IP address of conn. station
        DMS_UINT32      ulBoardType;  /*  Board type
        DMS_UINT32      ulConnFlag;   /*  */
```

} DMS_REC_CONN_DATA;

The ConnStatus can take the following values:

```
DMS_CONN_OK,                    /* All OK */
DMS_CONN_ABORT,                 /* No connection */
DMS_CONN_INVALID_RES_TYPE,      /* Incorrect resource type */
DMS_CONN_INVALID_RES_NO,        /* Incorrect resource  */
DMS_CONN_NO_OS,                 /* No operating system */
DMS_CONN_SECONDARY,             /* Only connected to
                                secondary => invalid
                                        configuration */
DMS_CONN_INVALID_VERSION        /* Invalid DMS version */
```

The ulBoardType can take the following values:

```
DMS_CPU_UNKNOWN
DMS_CPU_DCP02
DMS_CPU_DCP10
DMS_CPU_PC
```

The ulConnFlag can take the following values:

```
DMS_RES_PRIMARY         /* Connection to a primary server */
DMS_RES_SECONDARY       /* Connection to a secondary server */
DMS_RES_CLIENT          /* Connection to a client */
```

Possible return values:

Function	**Description**
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_NO_CONF	No project available

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_CONF	No information available about the specified station
E_DMSAPI_NO_RESOURCE	The station cannot be connected at present as there are still stations with the state ‘disconnecting’.
E_DMSAPI_TIMEOUT	It was not possible to establish a connection within the specified timeout.
E_DMSAPI_INTERNAL_ERROR	Internal error on establishing connection
E_DMSAPI_MAX_CONNECTION	The DMS layer does not allow more than the maximum number of connections.

DMSAPI_ConnectByNo


Icon.bmp

SYNTAX

```
DMS_RC DMSAPI_ConnectByNo(
DMS_RES_NO          OwnResNo        /* Own Resource No */,
DMS_RES_NO          ResNo           /* Resource No */,
DMS_CONN_HANDLE    *lpConnHandle    /* ConnectionHandle */,
DMS_INT16           nSyncFlag       /* Synchronisation flag */,
DMS_UINT32          ulProcT         /* Procedure timeout */,
DMS_UINT32          ulRecConnLen    /* Size of storage area
                                       referenced to the pointer */,
DMS_REC_CONN_DATA   *RecConn        /* RecStruct of Conn. */
)
```

The procedure establishes the connection to a DMS resource. The project loaded by Freelance Engineering must be accessible in its own resource. Through the name management functions the resource number is converted to:

- BTRLnk
- IPAddr1
- IPAddr2
- KeepAliveTimeout
- ResourceNo
- ResourceType

Establishing a connection to a DMS server station: The return value is a connection handle that can be used later to identify the connected resource. This enables more than one connection to be established to a resource. Thus, for instance, one connection can be used for alarm messages, and another for updating or operation. It is also possible to perform all the different services on a single connection. If the server station has been configured with redundancy, then the connection to the active station will be opened automatically. The connection between client and server station is monitored. The connection is monitored with the parameter KeepAliveTimeout. If the synchronisation flag is set to DMSAPI_SNYC and if the connection is established within the declared procedure timeout, then the connection structure is filled with values.

After an interruption to a connection the connection is re-established automatically. If the application no longer needs this connection the procedure DMSAPI_Disconnect should be called.

Parameters

- OwnResNo: Own station’s resource number. The DMSAPI_Init must be called.
- ResNo: Number of the target resource as configured in Freelance Engineering.
- lpConnHandle: After the connection has been established, data exchange on this connection is addressed through this ConnectionHandle.
- nSyncFlag

DMSAPI_SYNCHRON: The procedure waits for the length of time specified by "ProcedureTimeout" for the connection to be established. If the connection is established successfully, RecStruct returns valid values. The process of establishing a connection also continues after the timeout has expired.

DMSAPI_ASYNCHRON: When a connection is established, this is indicated through the Callback function, or it may be read through the function DMSAPI_GetConnectionStatus. The procedure timeout has no significance.

- ulProcT:

DMSAPI_NO_TIMEOUT no timeout

Value in milliseconds

DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.

- ulRecConnLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used. Then it has the same length as the structure DMS_REC_CONN_DATA
- RecConn:

```
typedef struct DMS_REC_CONN_DATA {
DMS_RES_NO          OwnResNo;          /* Own station number */
DMS_RES_NO          ResNo;             /* Station number of station */
DMS_RES_TYPE        ResType;           /* Resource type of server station */
DMS_CONN_STATUS  ConnStatus;           /* Connection status of station */
DMS_UINT32          ulIPAddr;          /* IP address of conn. station */
DMS_UINT32       ulBoardType;          /*  BoardType */
DMS_UINT32       ulConnFlag;    /*  */
} DMS_REC_CONN_DATA;
```

The ConnStatus can take the following values:

```
DMS_CONN_OK,                           /* All OK */
DMS_CONN_ABORT,                        /* No connection */
```

```
DMS_CONN_INVALID_RES_TYPE,      /* Incorrect resource type */
DMS_CONN_INVALID_RES_NO,        /* Incorrect resource number */
DMS_CONN_NO_OS,                 /* No operating system */
DMS_CONN_SECONDARY,             /* Only connected to secondary =>
                                   invalid configuration */
DMS_CONN_INVALID_VERSION        /* Invalid DMS version */
```

The ulBoardType can take the following values:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

The ulConnFlag can take the following values:

```
DMS_RES_PRIMARY/* Connection to a primary server */
DMS_RES_SECONDARY/* Connection to a secondary server */
DMS_RES_CLIENT/* Connection to a client */
```

Possible return values:

Function	**Description**
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_NO_CONF	No project available
E_DMSAPI_INVALID_CONF	No information available about the specified station
E_DMSAPI_NO_RESOURCE	The station cannot be connected at present as there are still stations with the state ‘disconnecting’.

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_TIMEOUT	It was not possible to establish a connection within the specified timeout.
E_DMSAPI_INTERNAL_ERROR	Internal error on establishing connection
E_DMSAPI_MAX_CONNECTION	The DMS layer does not allow more than the maximum number of connections.

DMSAPI_Disconnect

SYNTAX

DMS_RC DMSAPI_Disconnect(

DMS_CONN_HANDLE ConnHandle /* ConnectionHandle */

)

The function performs a controlled disconnection from the specified resource. Before a DMS session is terminated, all resources should be re-enabled. The Connhandle is not enabled immediately after the procedure is completed; this only takes place after the controlled disconnection. The specified callback functions are called first, that is the handles are not released until the callback functions have been called.

Parameters:

ConnHandle: ConnectionHandle which was returned when procedure DMSAPI_Connect was called.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error on disconnection

DMSAPI_GetConnectionData

SYNTAX

```
DMS_RC DMSAPI_GetConnectionData(
        DMS_CONN_HANDLE ConnHandle/* ConnectionHandle */,
        DMS_UINT32      ulRecConnLen /* Size of storage area
                         referenced to the pointer */
        DMS_REC_CONN_DATA*RecConn/* ReceiveStructure of Conn.
*/
)
```

The function returns the connection structure for a valid ConnectionHandle. If the DMS is installed without any Callback function, this function must be used to check the connection status of the individual resources.

Parameters:

- ConnHandle: ConnectionHandle which was returned when procedure DMSAPI_Connect was called.
- ulRecConnLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used. Then it has the same length as the structure DMS_REC_CONN_DATA
- RecConn:

typedef struct DMS_REC_CONN_DATA {

```
DMS_RES_NO                  OwnResNo;          /* Own station number */
DMS_RES_NO                  ResNo;             /* Station number of station */
DMS_RES_TYPE                ResType;           /* Resource type of server station */
DMS_CONN_STATUSConn         Status;            /* Connection status of station */
DMS_UINT32                  ulIPAddr;          /* IP address of connected station */
DMS_UINT32                  ulBoardType;       /*  BoardType */
DMS_UINT32                  ulConnFlag;        /*  */
} DMS_REC_CONN_DATA;
```

The ConnStatus can take the following values:

```
DMS_CONN_OK,                                   /* All OK */
DMS_CONN_ABORT,                                /* No connection */
DMS_CONN_INVALID_RES_TYPE,                     /* Incorrect resource type */
DMS_CONN_INVALID_RES_NO,                       /* Incorrect resource number */
DMS_CONN_NO_OS,                                /* No operating system */
DMS_CONN_SECONDARY,                            /* Only connected to secondary =>
                                                  invalid configuration */
DMS_CONN_INVALID_VERSION                       /* Invalid DMS version */
```

The ulBoardType can take the following values:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

The ulConnFlag can take the following values:

```
DMS_RES_PRIMARY          /* Connection to a primary server */
DMS_RES_SECONDARY        /* Connection to a secondary server */
DMS_RES_CLIENT           /* Connection to a client */
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error on establishing connection

DMSAPI_SetSystemTime

SYNTAX

DMS_RC DMSAPI_SetSystemTime(

SYSTEMTIME Time /* Time */[Pointer to GMT])

This function sends the time as displayed to all connected Freelance stations as a broadcast on the Ethernet. There is no acknowledgement to confirm whether or not this time packet has arrived at any station.

Under Windows the current time can be queried using the function GetLocal.

Parameter:

- Time: Type is Windows type SYSTEMTIME, which contains the time to be set.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid. No times prior to 1984 are sent (beginning of MMS time).
E_DMSAPI_INTERNAL_ERROR	Internal error on sending time

DMSAPI_RestartResource

SYNTAX

```
DMSAPI_RestartResource(
                DMS_CONN_HANDLE     ConnHandle    /* ConnectionHandle */,
                DMS_RESTART_REASONRestartRea/* RestartReason */
)
```

This procedure performs a cold or warm start on the Freelance process station.

After startup the process station is rebooted, that is the connection is broken and then re-established.

Parameters:

- Connhandle: ConnectionHandle for this resource
  - RestartReason

    DMSAPI_RESTART_WARM: Process station warm start

    DMSAPI_RESTART_COLD: Process station cold start

    DMSAPI_RESTART_TOGGLE: Switches process station over from primary to secondary. This command can only be sent to a redundant process station.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_CONNECTION	No connection to this station.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.2 Variable access services

Using the variable access services, data from a DMS server can be:

- read once
- read cyclically
- written once.

Through the DMSAPI complete variables lists are read and written. For the various services (read once, write once, read cyclically) empty lists must be set up into which variables can then be inserted. All variables within a variables list are read or written at the same time. The effect of these operations being performed at the same time is that the calculation of loop functions and tasks is interrupted for the duration of the variables list operation.

Once the read or write operation has been completed, the variables list can be further used as follows:

==> Deleting existing variables from the variables list

==> Adding new variables to the variables list

==> Changing values within the variables list

==> Deleting all variables from the variables list

==> Deleting the variables list

After this the read/write function can be performed again.

Any variables lists that are no longer required must always be deleted explicitly. The variables list must also be deleted when the connection to a station is interrupted. Once a variables list has been deleted, nothing else is received for that list. If the

deletion of variables lists is “forgotten” by the application, once a certain number of variables lists have been lost the application cannot create any more new lists.

Cyclical variables lists must be stopped before any changes are made, and can be re-started afterwards. Likewise, after a variables list has been stopped, nothing more is received for that list.

Variables lists that are to be read or written once only can only be altered after the response has been received. (There is little point in deleting variables lists before the response has been received).

A variables list can only contain variables from the same station.

3.3 Caution!

The reading/writing of variables exerts a load on the DMS server. If the read/write routines are called cyclically from the API, possibly as quickly as possible, this results at the process station in a CPU engagement factor of up to 80%. For this reason the following rules should be observed:

- For cyclical read tasks the function ReadCycleVarList should be used, rather than using ReadVarList cyclically.
- For reading and writing, as many tasks as possible should be performed in one variables list rather than perform each variable function in a separate variables list. Requests for variables should first be collected together and then executed.

During routine operation the process station responds after a time of 20 to 100 milliseconds.

The structure of the variables list is described on Page B-71, Receiving/decoding data, The procedures for accessing variables lists can be classified as follows:

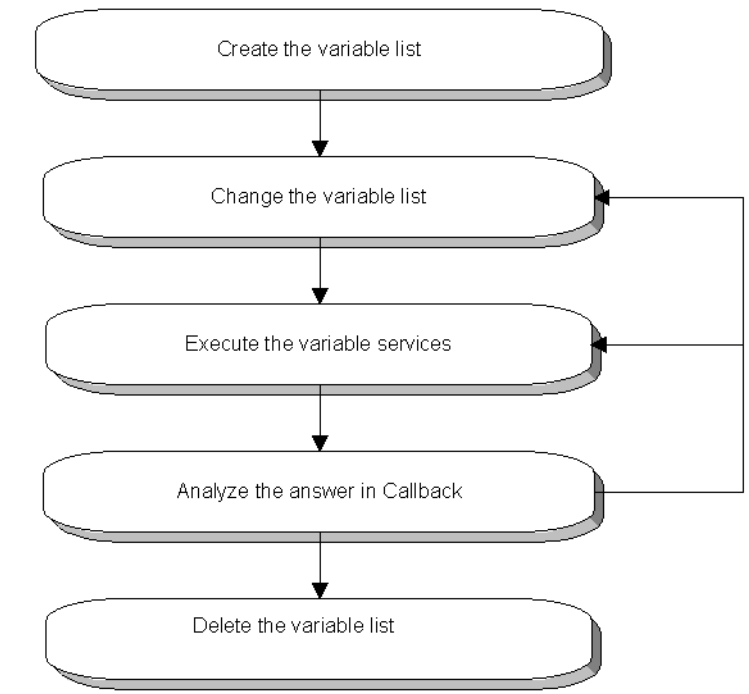
Creating a variables list DMSAPI_VLCreate

Modifying the variables list: DMSAPI_VLAddWriteVarByName(only for writing)

DMSAPI_VLAddReadVarByName(only for reading)
DMSAPI_VLAddWriteVarByAddr(only for writing)
DMSAPI_VLAddReadVarByAddr(only for reading)
DMSAPI_VLChangeValue (only for writing)

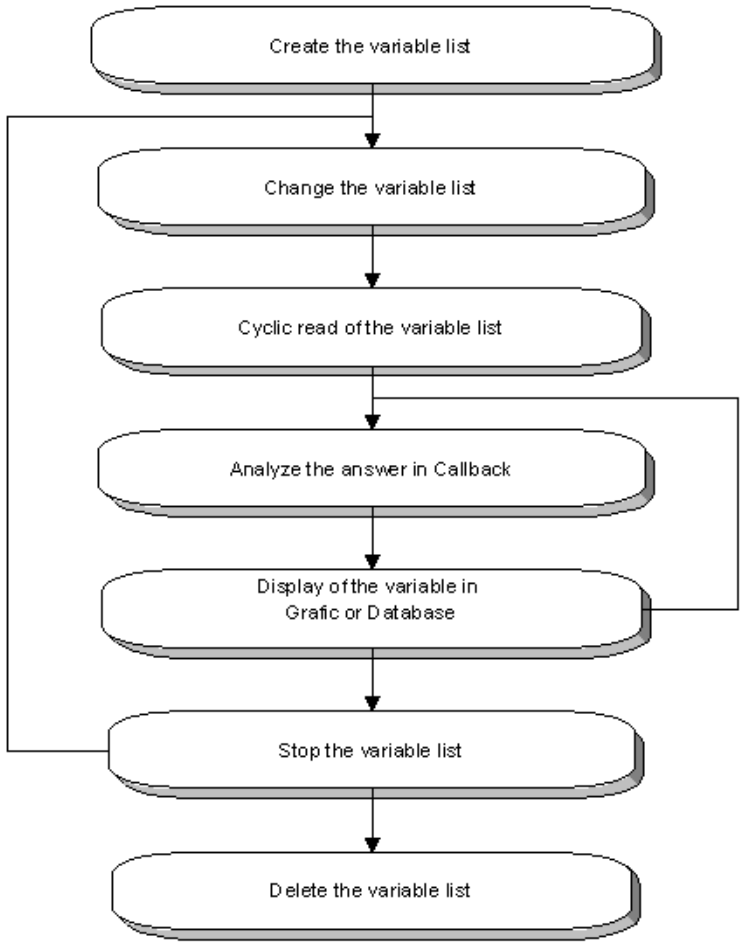
	DMSAPI_VLDelVar DMSAPI_VLClear
Simple variable functions:	DMSAPI_VLRead DMSAPI_VLWrite
Cyclical variable functions:	DMSAPI_VLReadCycle
Cycl. stop Variables lists	DMSAPI_VLStopCycle
Deleting variables lists:	DMSAPI_VLDelete

Lifespan of a variables list for read and write functions



ap015us.bmp

Lifespan of a variables list for cyclical read functions



ap016us.bmp

3.3.1 DMSAPI_VLCreate

SYNTAX

```
DMS_RC DMSAPI_VLCreate(
        DMS_CONN_HANDLE  ConnHandle /*  ConnectionHandle */,
        DMS_INT16        nVLService /*  Type of VarList Service */,
        DMS_HANDLE       *lpDmsHandle/*  Identifier for Varlist */
)
```

Through this procedure the storage is set up and a unique DMS-Handle created for a DMS variables list. Once a variables list has been created, variables may be inserted into it.

Populated variables lists can be utilised via the Read/ Readcycle/ Write functions.

The storage and DMS-Handle are deleted only via the function DMSAPI_DeleteVarList.

Parameters:

- Connhandle: ConnectionHandle for this resource
- nVLService:

DMSAPI_VL_SINGLE_READ: reading this variables list once only

DMSAPI_VL_CYCLE_READ: reading this variables list cyclically

DMSAPI_VL_SINGLE_WRITE: writing this variables list once only

- lpDmsHandle Handle for this variables list, through which all further operations on this variables list are controlled

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	No resources (storage/DMSHandles) in order to create this variables list.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_VLAddReadVarByName

SYNTAX

DMS_RC DMSAPI_VLAddReadVarByName(

```
   DMS_HANDLE              DmsHandle          /*  VarListHandle */,
   DMS_CHAR               *lpszVarname        /*  Variable name */,
   DMS_REC_VARLIST_DATA  **lplpRecVar     /*  Pointer to RecVarStruct */,
   DMS_INT16              *lpnIndex         /*  Index in RecVarStruct */
)
```

This procedure adds one element to an existing variables list. The element is addressed via the variable name. The procedure converts variable names into DMS addresses. As a result of this, the procedure requires more calculating time than the routine

DMSAPI_VLAddVarReadByAddr.

Once a variables list has been filled, the required function can be performed. The variables list cannot be changed through the add and delete functions until any read function has been completed (or halted).

The number of bytes that can be accommodated in a variables list is limited. This number of bytes determines the number of variables that can be communicated. As variables of different data types require differing amounts of storage space, it is not possible to define a constant DMSAPI_MAX_VAR_IN VARLIST. The procedure DMSAPI_GetVarLen returns the storage requirement of the different variable types within a variables list. The constant DMSAPI_VL_MAX_BYTES shows the maximum storage occupied by a variables list. The structure DMS_REC_VARLIST_DATA always indicates the amount of storage available within the variables list.

Parameters:

- DmsHandle: Variables list handle for this variables list
- lpszVarname: Name of the variable to be read
- lplpRecVar: Structure of the populated variables list

```
typedef struct DMS_REC_VARLIST_DATA {
        DMS_HANDLEDmsHandle;
        DMS_INT16    ActVarNo;   /* Current number of variables */
        DMS_INT16   MaxVarNo;   /* Max. number of variables with
                                                blank entries */
        DMS_INT16   FreeBytes;    /* Number of free bytes in the VL */
        DMS_REC_VAR *lpVar;     /* Actual variables list */
} DMS_REC_VARLIST_DATA;
```

The structure DMS_REC_VAR:

```
typedef struct DMS_REC_VAR {
        DMS_VAR_STATUS   VarStatus;          /* Status of the variable */
        DMS_VAR_RC          VarRc;         /* ReturnCode after function */
        DMS_OBJ_PATH      ObjPath;        /* Object path on server */
        DMS_CHAR           VarName;         /* Variable name or NULL */
        DMS_UINT32        ValueSize;        /* Size of the value buffer */
```

```
DMS_VAR_TYPE      VarType;        /* Type of value */
DMS_VALUE         *VarValue;      /* Value of the variables or
                                     NULL */
} DMS_REC_VAR;
```

VarStatus can take the following values:

DMS_VAR_NOT_VALID: After the value has been added and before the function is executed, or where an error occurred when the function was executed.

DMS_VAR_CHANGED: After a function has been executed

DMS_VAR_DELETED: Variable has been deleted via DMSAPI_VLDelVar but the entry still exists

VarRc can accept various errors from the server. See DMS error codes on page 150.

The ObjPath has the structure DMS_OBJ_PATH and contains the addressing on the server.

```
typedef struct {
                    DMS_OBJNO ObjNo;
                    DMS_CMPNOCmpNo;
                } DMS_OBJ_PATH;
```

VarType can take on various values. (See Appendix A, Variable types and error codes)

VarValue is a pointer that points to the value of the variable after the read function has been executed (See DMS variable types on page 147).

- lpnIndex within the lplpRecVar

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	Variables list is full A new variables list must be created.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_NO_CONF	No project available
E_DMSAPI_INVALID_CONF	No information available about the specified variable
E_DMSAPI_INVALID_DMS_HANDLE	A valid variables list handle has not been passed.
E_DMSAPI_INVALID_CONN_HANDLE	The specified variable is not present on the specified resource
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_VLAddWriteVarByName

SYNTAX

```
DMS_RC DMSAPI_VLAddWriteVarByName(
        DMS_HANDLE          DmsHandle          /*  VarListHandle */,
        DMS_CHAR            *lpszVarname       /*  Variable name */,
        DMS_VAR_TYPE        VarType;           /* Type of value */
        DMS_VALUE           *lpVarValue;       /* Value of the variable or
                                                  NULL */
        DMS_REC_VARLIST_DATA **lplpRecVar      /*  Pointer to
                                                  RecVarStruct */,
        DMS_INT16           *lpnIndex          /*  Index in RecVarStruct */
)
```

This procedure adds one element to an existing variables list. The element is addressed through the variable name. The procedure converts variable names into DMS addresses. As a result of this, the procedure requires more calculating time than the routine.

DMSAPI_VLAddVarWriteByAddr.

Once a variables list has been filled, the write function can be performed. After a write function has been completed the variables list can be edited through the “Add, Change, Delete” functions.

A variables list can only accommodate a limited number of bytes. This number of bytes determines the number of variables that are communicated. As variables of different data types require differing amounts of storage, it is not possible to define a DMSAPI_MAX_VAR_IN VARLIST constant. The procedure DMSAPI_GetVarLen returns the storage requirement of the various variable types within a variables list. The constant DMSAPI_VL_MAX_BYTES shows the maximum amount of storage occupied by a variables list.

The structure DMS_REC_VARLIST_DATA always shows the amount of storage available within the variables list.

Parameters:

- DmsHandle: Variables list handle for this variables list
- lpszVarname: Name of the variable to be read
- VarType can take the following values (See DMS variable types on page 147).
- VarValue is a reference to the value of the variable for executing the write function (See DMS variable types on page 147).
- lplpRecVar: Structure of the populated variables list
- lpnIndex: lpnIndex within the lplpRecVar

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	Variables list is full A new variables list must be created.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_VARTYP	The variable passed is of the wrong type.
E_DMSAPI_INVALID_NO_CONF	No project available
E_DMSAPI_INVALID_CONF	No information available about the specified variable
E_DMSAPI_INVALID_DMS_HANDLE	A valid variables list handle has not been passed.
E_DMSAPI_INVALID_CONN_HANDLE	The specified variable is not present on the specified resource
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_VLAddReadVarByAddr

SYNTAX

```
DMS_RC DMSAPI_VLAddReadVarByName (
                DMS_HANDLE          DmsHandle           /*  VarListHandle */,
                DMS_OBJ_PATH        lpOPath;        /* Object path on server */
                DMS_VAR_TYPE        VarType;            /* Type of value */
                DMS_REC_VARLIST_DATA**lplpRecVar    /*  Pointer to
                                                        RecVarStruct */,
                DMS_INT16           *lpnIndex           /*  Index in
                                                        RecVarStruct*/
)
```

This procedure adds one element to an existing variables list. The element is addressed through the object path and the variable type. This procedure offers a time advantage over the procedure DMSAPI_VLAddReadVarByName as the variable name does not need to be converted into a DMS address.

Once a variables list has been filled, the required function can be performed. Once a read function has been completed (or halted) the variables list can be changed through the Add and Delete functions.

A variables list can only accommodate a limited number of bytes. This number of bytes determines the number of variables that are communicated. As variables of different data types require differing amounts of storage, it is not possible to define a DMSAPI_MAX_VAR_IN VARLIST constant. The procedure DMSAPI_GetVarLen returns the storage requirement of the various variable types within a variables list. The constant DMSAPI_VL_MAX_BYTES shows the maximum amount of storage occupied by a variables list. The structure DMS_REC_VARLIST_DATA always shows the amount of storage available within the variables list.

Parameters:

- DmsHandle: Variables list handle for this variables list
- ObjPath has the structure DMS_OBJ_PATH and contains the addressing on the server.

```
typedef struct {
        DMS_OBJNO ObjNo;
        DMS_CMPNOCmpNo;
} DMS_OBJ_PATH;
```

- VarType can take the following values (See DMS variable types on page 147).
- lplpRecVar: Structure of the populated variables list
- lpnIndex: lpnIndex within the lplpRecVar

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	Variables list is full A new variables list must be created.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variables list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_AddWriteVarByAddr

SYNTAX

```
DMS_RC DMSAPI_VLAddWriteVarByName(
        DMS_HANDLE          DmsHandle          /*  VarListHandle */,
        DMS_OBJ_PATH        lpOPath;           /* Object path on
                                                  server */
        DMS_VAR_TYPE        VarType;           /* Type of value */
        DMS_VALUE           *lpVarValue;       /* Value of the variable or
                                                  NULL */
      DMS_REC_VARLIST_DATA**lplpRecVar/*  Pointer to RecVarStruct */,
      DMS_INT16             *lpnIndex          /*  Index in RecVarStruct */
)
```

This procedure adds one element to an existing variables list. The element is addressed through the object path and the variable type. This procedure offers a time advantage over the procedure DMSAPI_VLAddWriteVarByName as the variable name does not need to be converted into a DMS address.

Once a variables list has been filled, the required function can be performed. After a write function has been completed the variables list can be edited through the “Add, Change, Delete” functions.

A variables list can only accommodate a limited number of bytes. This number of bytes determines the number of variables that are communicated. As variables of different data types require differing amounts of storage, it is not possible to define a DMSAPI_MAX_VAR_IN VARLIST constant. The procedure DMSAPI_GetVarLen returns the storage requirement of the various variable types within a variables list. The constant DMSAPI_VL_MAX_BYTES shows the maximum amount of storage occupied by a variables list. The structure DMS_REC_VARLIST_DATA always shows the amount of storage available within the variables list.

Parameters:

- DmsHandle: Variables list handle for this variables list
- ObjPath has the structure DMS_OBJ_PATH and contains the addressing on the server.

```
typedef struct {        DMS_OBJNO ObjNo;
                        DMS_CMPNOCmpNo;
} DMS_OBJ_PATH;
```

- VarType can take the following values (See DMS variable types on page 147).
- VarValue is a reference to the value of the variable for executing the write function (See DMS variable types on page 147).
- lplpRecVar: Structure of the populated variables list
- lpnIndex: Index within the lplpRecVar

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	Variables list is full A new variables list must be created.
E_DMSAPI_INVALID_VARTYP	The variable passed is of the wrong type.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variables list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_VLChangeValue

SYNTAX

```
DMS_RC DMSAPI_VLChangeValue(
        DMS_HANDLE      DmsHandle           /*  VarListHandle */,
        DMS_INT16       nIndex          /*  Index in RecVarStruct */,
        DMS_VAR_TYPE    VarType;        /* Type of value */
        DMS_VALUE       *lpVarValue;    /* Value of the variable or
                                           NULL */
        DMS_REC_VARLIST_DATA**lplpRecVar    /*  Pointer to
                                           RecVarStruct */,
)
```

In an existing variables list this procedure modifies the value that is to be written. When the procedure is called the variables list is not allowed to wait for the response to the previous write access.

This procedure is needed if there is a requirement to write to the same variables several times in succession. There is no need to create the variables list anew each time.

Parameters:

- DmsHandle: Variables list handle for this variables list
- nIndex: Index within the lplpRecVar
- VarType can take the following values. (See Appendix A, Variable types and error codes)
- VarValue is a reference to the value of the variable for executing the write function (See DMS variable types on page 147).
- lplpRecVar: Structure of the populated variables list

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_INDEX	Invalid index in the variables list.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_VARTYP	The variable passed is of the wrong type.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.2 DMSAPI_VLDelVar

SYNTAX

```
DMS_RC DMSAPI_VLDelVar(
                DMS_HANDLE      DmsHandle           /*  VarListHandle */,
                DMS_INT16       nIndex          /*  Index in RecVarStruct */,
                DMS_REC_VARLIST_DATA**lplpRecVar/*  Pointer to
RecVarStruct */,
)
```

In an existing variables list this procedure deletes the variable that is addressed through the index. The indexes of the other variables remain unchanged. When new variables are added to the variables list these gaps are filled up.

This procedure is needed if faceplates can be opened and closed through user intervention in a graphic.

If a variable is being accessed cyclically for reading, it must be stopped before variables are deleted.

Parameters:

- DmsHandle: Variables list handle for this variables list
- nIndex: Index within the lplpRecVar
- lplpRecVar: Structure of the populated variables list

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_INDEX	Invalid index in the variables list.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.3 DMSAPI_VLClear

SYNTAX

```
DMS_RC DMSAPI_VLDelVar(
                DMS_HANDLE          DmsHandle   /*  VarListHandle */,
)
```

This procedure deletes all variables from an existing variables list. Afterwards, the variables list can be populated with new variables.

Parameter:

- DmsHandle: Variables list handle for this variables list

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.4 DMSAPI_VLRead


Icon.bmp

SYNTAX

```
DMS_RC DMSAPI_VLRead(
                DMS_HANDLE      DmsHandle          /*  VarListHandle */;
                DMS_INT16         nCBId            /* CallbackId */,
                DMS_INT16       nSyncFlag      /* Synchronisation flag */,
                DMS_UINT32        ulProcT        /* Procedure timeout */,
                DMS_UINT32     ulRecVarLen ^     /* Size of storage area
                                                        to the pointer */,
DMS_REC_VARLIST_DATA      *lpRecVar          /* RecStruct of VL */
```

This procedure executes the simple read function on a populated variables list. There is a response to this query. Once the response has been received and evaluated, the variables list can be modified and re-read or completely deleted using the Delete and Insert procedures.

Parameters:

- DmsHandle: Variables list handle for this variables list
- nCBId: CallbackId, or the variables list is retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the Read function.
- DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the read access again in cases where insufficient resources are available.
- ulProcT:
  DMSAPI_NO_TIMEOUT no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.
- ulRecVarLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used.
- lpRecVar: where the structure of the variables list being read is read simultaneously with the current values (null for asynchronous operation).

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_VARMODE	The called function does not correspond with the variables list type set up at the time of its creation.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.

Function	Description
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Read function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.5 DMSAPI_VLReadCycle

SYNTAX

```
DMS_RC DMSAPI_VLReadCycle(
                DMS_HANDLE      DmsHandle           /*  VarListHandle */;
                DMS_UINT32      ulCycleTime         /*  Cycle time in ms */,
                DMS_INT16       nCBId               /* CallbackId */,
                DMS_INT16       nSyncFlag          /* Synchronisation flag */,
                DMS_UINT32        ulProcT           /* Procedure timeout */,
                DMS_UINT32            ulRecVarLen   /* Size of storage area
                                                    referenced to the pointer */,
```

```
                    DMS_REC_VARLIST_DATA   *lpRecVar     /* RecStruct of VL */
)
```

This procedure executes the cyclical read function on a populated variables list. There is a response and cyclical variables list messages in response to this query. The cyclical Read function is stopped via the function DMSAPI_StopCycleVar. Once it has been stopped, the variables list can be modified and re-read or completely deleted using the Delete and Insert procedures.

If the variables list is stopped, the new cyclical Read access is not started before the response from the previous Stop has been received. If the variables list is stopped by the application before the cyclical function has been executed, then the function is cancelled.

Parameters:

- DmsHandle: Variables list handle for this variables list
- ulCycleTime: Cycle time in milliseconds; the specified cycle time is rounded to the nearest number divisible by 200.
- nCBId: CallbackId, or the variables list is retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the Read function.
- DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the read access again in cases where insufficient resources are available.
- ulProcT:
  DMSAPI_NO_TIMEOUT no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.
- ulRecVarLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used.
- lpRecVar: when reading is executed synchronously, the structure of the variables list that is read, along with the current values

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_VARMODE	The called function does not correspond with the variables list type set up at the time of its creation.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Read function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.6 DMSAPI_StopCycle


Icon.bmp

SYNTAX

DMS_RC DMSAPI_VLStopCycle(

DMS_HANDLEDmsHandle/* VarListHandle */;

)

This procedure stops a running cyclical variables list. Once a cyclical variables list has been stopped, nothing more is received for that variables list. This means that if this procedure is called before the application has received the first values for the variables list, then it will never be able to receive and evaluate the requested values. Variables can be removed from and added to a stopped variables list. Subsequently, this variables list can be read cyclically again.

Parameters:

- DmsHandle: Variables list handle for this variables list
- nSyncFlag
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the Stop function.
- DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the Stop access again in cases where insufficient resources are available.
- ulProcT:
  DMSAPI_NO_TIMEOUT no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_VARMODE	No cyclical Read function is started for this variables list.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation => list is stopped automatically.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Stop function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.7 DMSAPI_VLWrite

SYNTAX

```
DMS_RC DMSAPI_VLWrite(
DMS_HANDLE      DmsHandle          /*  VarListHandle */;
DMS_INT16       nCBId              /* CallbackId */,
```

```
DMS_INT16                nSyncFlag                /* Synchronisation flag */,
DMS_UINT32               ulProcT                  /* Procedure timeout */,
DMS_UINT32               ulRecVarLen              /* Size of storage area
                                                     referenced to the pointer */,
DMS_REC_VARLIST_DATA   *lpRecVar                  /* RecStruct of VL */
)
```

This procedure executes the Write function on a populated variables list. There is a response to this query. Once the response has been received, the variables list can be modified, re-written or completely deleted using the Modify Values, Delete and Add procedures.

Parameters:

- DmsHandle: Variables list handle for this variables list
- nCBId: CallbackId, or the variables list is retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the Write function.
  DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the write access again in cases where insufficient resources are available.
- ulProcT:
  DMSAPI_NO_TIMEOUT no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.
- ulRecVarLen: only used when the synchronisation flag DMSAPI_SYNCHRON is being used.
- lpRecVar: when writing is executed synchronously, the structure of the variables list that is read, along with the current values

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_VARMODE	The called function does not correspond with the variables list type set up at the time of its creation.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Write function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.3.8 DMSAPI_VLDelete

SYNTAX

```
DMS_RC DMSAPI_VLDelete(
DMS_HANDLE          DmsHandle    /*  VarListHandle */
)
```

This procedure deletes an existing variables list. Even if the variables list is being read cyclically at the time, it is automatically stopped and deleted. Once it has been deleted the Callback function is not called up for that variables list again.

Parameter:

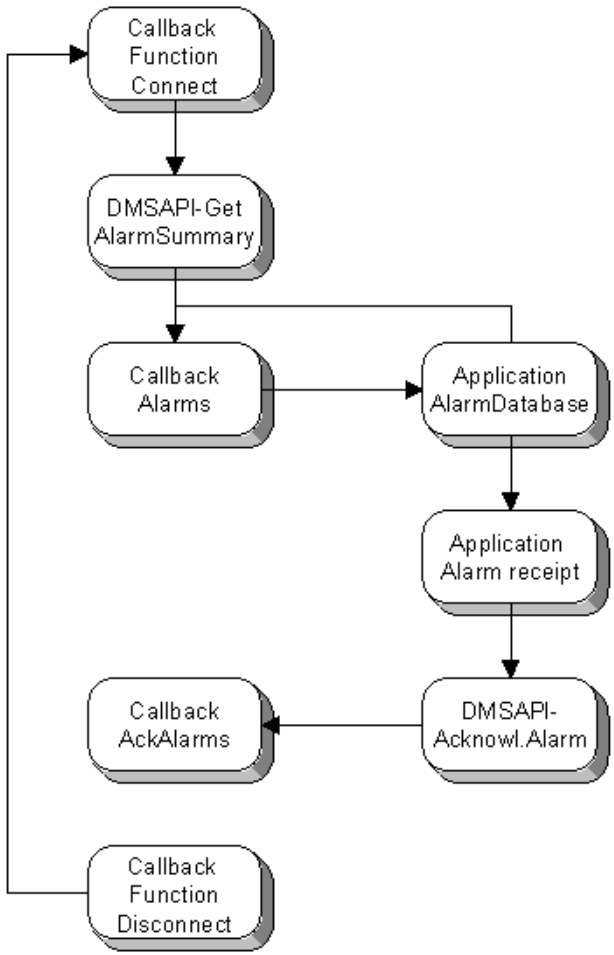
- DmsHandle: Variables list handle for this variables list

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_DMS_HANDLE	A valid variable list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4 Alarm management

Starting acquisition of alarms after a connection has been established.



ap017us.bmp

After the GetAlarmSummary function has been started, the Callback function for alarms is called automatically each time a new alarm is received. The application stores these alarms in an alarms database, and can acknowledge them if required. There is a response to each alarm acknowledgement. The Callback function for AcknowledgeAlarms is called.

3.4.1 DMSAPI_GetAlarmSummary

SYNTAX

```
DMS_RC DMSAPI_GetAlarmSummary(
DMS_CONN_HANDLE          ConnHandle     /*  ConnectionHandle */,
                DMS_INT16        nCBId          /*  Callbackid */,
                DMS_INT16        nSyncFlag      /*  Synchronisation flag */,
                DMS_UINT32       ulProcT        /*  Procedure timeout */,
                DMS_UINT32       ulRecAlaLen    /*  Size of storage area
                                                    referenced to the pointer */,
DMS_REC_ALARMLIST_DATA*lpAlarmRec    /*  Pointer to AlarmListStruct */
)
```

After a GetAlarmSummary all alarms at the process station as well as all alarms arriving from this point in time on are sent to the client. Following interruption of the connection this function must be requested again.

Parameters:

- Connhandle: ConnectionHandle for this resource
- nCBId: CallbackId, or the alarms are retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON:The procedure waits for the specified procedure timeout for the first response from the GetAlarmSummary function.
  DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the write access again in cases where insufficient resources are available.
- ulProcT:
  DMSAPI_NO_TIMEOUT no timeout

Value in milliseconds
DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.

- ulRecAlaLen: is only used when the synchronisation flag DMSAPI_SYNCHRON is used, and contains the length of the next storage area in sequence.
- lpAlarmRec: when the structure of the GetAlarmSummary is received synchronous with approximately the first 43 alarms

```
typedef struct DMS_REC_ALARMLIST_DATA {
        DMS_ALARM_LIST_TYPE  ListType;
        DMS_INT16          ActAlarmNo;  /* Current number of alarms */
        DMS_REC_ALARM      *lpAlarm;    /* Alarm list */
} DMS_REC_ALARMLIST_DATA;
```

The element ListType can take the following values:

DMS_ALARM_GAS old alarms which were requested through a GetAlarmSummary. There are more old alarms to follow.

DMS_ALARM_LAST_GAS: all the old alarms requested through GetAlarmSummary have arrived.

DMS_ALARM_EVENTS: list containing currently arising alarms

The element IpAlarm is the following alarm list with ActAlarmNo entries.

```
typedef struct DMS_REC_ALARM {
        DMS_DT            TransitionTime;          /* Alarm time */
        DMS_OBJNO         ObjectId;                /* ObjNumber */
        DMS_WORD16        AlarmIndex;              /* Alarm index */
        DMS_ALARM_TYPE    AlarmType;               /* Alarm type */
        DMS_OBJNO         ObjectClass;             /* Object class */
        DMS_ALARM_STATUS CurrAlarmStatus;    /* Curr. alarm stat. */
        DMS_ALARM_STATUS PrevAlarmStatus;    /* Old alarm st. */
```

```
    DMS_ALARM_PRIO   Priority;          /* Alarm prio */
    DMS_BOOLEAN      NotificationLost;  /* Alarm burst */
    DMS_RC           rc;                /* Alarm error */
    DMS_UINT32       ValueSize;         /* Size A-value */
    DMS_VAR_TYPE     AlarmValType;      /* Data type */
    DMS_VALUE        *AlarmValue;       /* Alarm value */
} DMS_REC_ALARM;
```

Alarm type can take values through which the text for the system messages can be identified.

CurrAlarmStatus and PrevAlarmStatus can take the following values:

Function	Description
DMS_ALARM_INACT_INACTNACKED	Inactive/NotAcknowledged
DMS_ALARM_ACT_ACTNACKED	Active/Active_NotAcknowledged
DMS_ALARM_INACT_INACTACKED	Inactive/Inactive_Acknowledged
DMS_ALARM_ACT_ACTACKED	Active/Acknowledged
DMS_ALARM_INACT_ACTNACKED	Inactive/Active_NotAcknowledged
DMS_ALARM_AP_DELETED	Alarm point has been deleted

AlarmValType can take the different values for the various different data types (See DMS variable types on page 147).

The alarm value is a reference to the value of the alarm point (See DMS variable types on page 147).

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the specified resource.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute GetAlarmSummary. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.2 DMSAPI_CreateAckAlarmList

DMSAPI_CreateAckAlarmList(ConnHandle, &DmsHandle);

Through this procedure the storage is set up and a unique DMS-Handle created for a DMS acknowledgement alarm list. Once this list has been created, alarm acknowledgements can be added to it.

Populated acknowledgement lists can be sent to the server through Acknowledge Alarm. After its receipt, the acknowledgement list can be cleared using the Clear function, and new acknowledgement messages can then be added.

The storage and DMS-Handle are deleted only through the function DMSAPI_DeleteAckAlarmList.

Parameters:

- Connhandle: ConnectionHandle for this resource
- DMS_Handle: Handle for this acknowledgement list, through which all further operations on the list are controlled

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_NO_RESOURCE	No resources (storage/DMSHandles) in order to create this acknowledgement list.
E_DMSAPI_INVALID_CONN_HANDLE	A valid ConnectionHandle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.3 DMSAPI_AddAckAlarmByAddr

DMSAPI_AddAckAlarmByAddr (DmsHandle, AlarmPoint, AlarmStatus, &AlarmIndex,)

This procedure adds one element to an existing acknowledgement list. The element is described by the alarm point and the alarm status.

Once an acknowledgement list has been populated, the acknowledgement function can be performed. Once the acknowledgements have been evaluated, the list can be cleared and used again.

Parameters:

- Dmshandle: Handle for the alarm acknowledgement list
- Alarm point of the alarm to be acknowledged
- Alarm status of the alarm to be acknowledged
- Alarm index within the RecStruct being returned

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_NO_RESOURCE	Acknowledgement list is full It must first be uploaded. Following this, the other alarms can be acknowledged.
E_DMSAPI_INVALID_ARG	The parameter passed is invalid.
E_DMSAPI_INVALID_DMS_HANDLE	A valid handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.4 DMSAPI_ClearAckAlarmList

DMSAPI_ClearAckAlarmList(DmsHandle)

This procedure clears all the acknowledgements from an existing alarm acknowledgement list. The procedure cannot be called if an acknowledgement for the list concerned is running at the time.

Parameter:

- Dmshandle: Handle for this alarm acknowledgement list.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_DMS_HANDLE	A valid handle has not been passed.
E_DMSAPI_SERVICE_IN_USE	The response to the acknowledgement for this list has not yet been received from the server.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.5 DMSAPI_AckAlarmList

DMSAPI_AckAlarmList(DmsHandle, CBId,SyncMode,Timeout,&RecStruct);

This procedure executes the Acknowledge function on a populated alarm acknowledgement list. There is a response to this query. Once the response has been received and evaluated, the list can be modified and re-acknowledged or completely deleted using the Delete and Insert procedures.

Parameters:

- Dmshandle: Handle for this acknowledgement list
- CBId: CallbackId, or the acknowledgement is retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- SyncFlag/ProcTimeOut
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the acknowledgement.
- DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the Write access again in cases where insufficient resources are available.
- ProcTimeOut:
  0 no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.
- RecStruct: when calling is executed synchronously, the structure of the acknowledgement list that is read, along with the current values.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Acknowledge function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.6 DMSAPI_DeleteAckAlarmList

DMSAPI_DeleteAckAlarmList(DmsHandle);

This procedure deletes an existing alarm acknowledgement list. If the response for the acknowledgement task has not yet been received, then when it does arrive it will not be forwarded to the application.

Parameter:

- Dmshandle: Handle for this alarm acknowledgement list

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_INVALID_DMS_HANDLE	A valid handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.4.7 DMSAPI_AckAlarmByList

SYNTAX

```
DMS_RC DMSAPI_ AckAlarmByList(
DMS_CONN_HANDLE     ConnHandle      /*  ConnectionHandle */,
DMS_HANDLE          *lpDmsHandle    /*  Identifier for Acklist */,
DMS_INT16             nCBId         /*  CallbackId */,
DMS_INT16           ActAlarmNo      /*  Current number of alarms to be
                                        acknowledged */,
```

```
DMS_REC_ACKALARM *lpAlarmAck        /*  Pointer to AlarmAckStruct */
DMS_INT16           nSyncFlag       /*  Synchronisation flag */,
DMS_UINT32          ulProcT         /*  Procedure timeout */,
);
```

This procedure executes the Acknowledge function on a transferred alarm acknowledgement list that has been filled by the application itself. There is a response to this query. Once the response has been received and evaluated, the list is automatically cleared, i.e. the DmsHandle is invalid. If the response to this request is synchronous, then the error codes for the individual acknowledgements are coded into the transferred list.

Parameters:

- ConnHandle: Connectionhandle for this resource
- lpDmsHandle: Handle for this acknowledgement list
- nCBId: CallbackId, or the acknowledgement is retrieved through the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- ActAlarmNo: Number of alarms to be acknowledged that have been transferred

lpAlarmAck: populated list containing alarms which are to be acknowledged

The element IpAlarm is the following alarm list with ActAlarmNo entries.

```
typedef struct DMS_REC_ACKALARM {
DMS_OBJNO           ObjectId;           /* ObjNumber */
DMS_WORD16          AlarmIndex;         / * AlarmIndex */
DMS_ALARM_STATUS    AlarmStatus;        /* Curr. AlarmSt. */
DMS_RC              rc;                 /* Alarm error */
} DMS_REC_ACKALARM;
```

The alarm status can take the following values:

DMS_ALARM_INACT_INACTNACKED: Inactive/not acknowledged

DMS_ALARM_ACT_ACTNACKED: Active/Active_Not acknowledged

DMS_ALARM_INACT_INACTACKED: Inactive/Inactive_Acknowledged

DMS_ALARM_ACT_ACTACKED: Active/Acknowledged

DMS_ALARM_INACT_ACTNACKED: Inactive/Active_NotAcknowledged

DMS_ALARM_AP_DELETED: Alarm point has been deleted

- nSyncFlag

DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from the alarm acknowledgement.

DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the alarm acknowledgement again in cases where insufficient resources are available.

- ulProcT:

DMSAPI_NO_TIMEOUT no timeout

Value in milliseconds

DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer was not initialized.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the resource specified at the time of creation.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.

Function	Description
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the Acknowledge function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INVALID_DMS_HANDLE	A valid list handle has not been passed.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.5 Domain management

Domain management is used for exchanging significant volumes of configuration data between DMS-client and DMS-server. These volumes of data are known as domains. A client can request the following domain functions on the server:

- Download domains onto the server
- Upload domains from the server
- Delete domains on the server

This domain management can only be performed by Freelance Engineering. The procedures are therefore not enabled and described for the DMSAPI applications.

3.6 Program invocation management

The management system for program invocation is currently only implemented and enabled for Freelance Engineering. Name management for the DMSAPI application does not at present hold any configuration information for implementing the task names on the process station for DMS address.

DMSAPI_StartPI

DMSAPI_StartPIByName (CBId, PIName, PILen, PI-Parameter, &DMS_Handle, SyncMode,Timeout,&PIMsg)

DMSAPI_StartPIByAddr (ConnHandle, CBId, ObjNo, PILen, PI-Parameter, &DMS_Handle, SyncMode,Timeout,&PIMsg)

This procedure executes a “StartProgramInvocation” function on a DMS server station. The “ProgramInvocation” is either identified by its name or by object number and connection.

Parameters:

- Connhandle: ConnectionHandle for this resource
- CBId: CallbackId, or the alarms are retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- PIName: Name of the PI object that is to be started. DMS name management must be activated.
- ObjNo: Object number of the PI object that is to be started
- PILen: Length of the PI parameters
- PI-Parameters: Parameters that are transferred to the PI
- Dmshandle: Handle for this acknowledgement list
- CBId: CallbackId, or the acknowledgement is retrieved via the DMSAPI-Receive function; CBId is assigned the value DMS_NO_CALLBACK
- SyncFlag / ProcTimeOut
  DMSAPI_SYNCHRON: The procedure waits for the specified procedure timeout for the response from PI Start.
- DMSAPI_ASYNCHRON: The response is not waited for. The timeout is used to automatically send the PI start access again in cases where insufficient resources are available.

- ProcTimeOut:
  0 no timeout
  Value in milliseconds
  DMSAPI_WAIT_FOREVER: Procedure will not return until the task is executed, or proves impossible to execute.
- RecStruct: When the procedure is called synchronously, the structure of the PI response is contained therein with the current values

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_TIMEOUT	The function called has been executed, but the response requested synchronously has not yet been received. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_NO_CONNECTION	There is currently no connection to the specified resource.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_INVALID_CONF	There is no information in name management on the specified PI object
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_SMALL_RCV_BUFF	The receive buffer advised is too small, only possible in synchronous operation.
E_DMSAPI_NO_RESOURCE	It was not possible at the time to execute the StartProgramInvocation function. More functions are being requested than the server is capable of processing within a time interval. This error cannot occur if DMSAPI_WAIT_FOREVER is supplied as the timeout.
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_StopPI

DMSAPI_StopPI (ConnHandle, PIName, PILen, PI-Parameter, &DMS_Handle, SyncMode,Timeout,&PIMsg)

DMSAPI_ResetPI

DMSAPI_ResetPI (ConnHandle, PIName, PILen, PI-Parameter, &DMS_Handle, SyncMode,Timeout,&PIMsg)

3.7 Receiving/decoding data

In the DMS-API applications have three options for receiving messages from the server station:

- In the procedure that issues the request it is possible to wait for the response synchronously. In this case the application provides the storage area into which the message is coded. The messages for the cyclical variables lists must continue to be received.
- The application can retrieve the responses actively via the DMSAPI_Receive function. In this case the application provides the storage area into which the message is coded.
- The application can obtain information on arriving messages through the Callback function. The storage location in which the message is coded belongs to the DMS-API. Once the Callback function is completed, the storage location ceases to be valid.

3.7.1 Structure definitions

When messages are received from the server station, the application is informed. It receives details for the following incoming messages.

==>	Connecting/disconnecting: Connection structure	
==>	Variable functions:	Variable structure
==>	InformationReport:	Information Report structure, the further definition of structures application-dependent Freelance defines the structure for curves and disturbance course logs
==>	Alarm functions:	Alarm structure
==>	Alarm acknowledgement function	Alarm acknowledgement structure
==>	Download functions	Download structure
==>	ProgramInvocation functions	ProgramInvocation structure
==>	Version changes:	Version structure

In the section DMS-Utilities there is a global function for these structures which outputs their content for debug purposes:

DMSAPI_DumpRecData

Data type	Component	Range of value
DMS_RC	rc	
DMS_CONN_HANDLE	ConnHandle: Handle for the connection	
DMS_INT32	BuffLen: Length of the next buffer in sequence. In the case of synchronous calling or synchronous receiving of data with the DMSAPI-Receive function, this buffer must be large enough.	

Data type	Component	Range of value
DMS_INT32	BuffType	DMS_REC_CONN_STATION_TYPE DMS_REC_VARLIST_TYPE DMS_REC_INFO_REPORT_TYPE DMS_REC_ALARMLIST_TYPE DMS_REC_ALARMACK_LIST_TYPE DMS_REC_PI_TYPE DMS_REC_DOM_TYPE DMS_REC_VERS_TYPE
DMS_REC_UNION	lpRecBuff: Union through all receive structures	

Connection structure

When a connection is being established, the ReturnCode can take the following values:

- E_DMSAPI_OK: All OK
- E_DMSAPI_INVALID_STATION_TYPE: Invalid station type
- E_DMSAPI_INVALID_STATION_NO: Invalid station number
- E_DMSAPI_NO_OS: No operating system

When disconnecting, the ReturnCode can only take the following value:

- E_DMSAPI_ABORT: Connection aborted

Data type	Component	Range of values
DMS_INT32	lBoardType: Type of CPU board	DMS_CPU_UNKNOWN DMS_CPU_DCP02 DMS_CPU_DCP10 DMS_CPU_PC DMS_CPU_HK80
DMS_INT32	lOSType: Type of operating system	DMS_OSVERSION_EPROM DMS_OSVERSION_MSR DMS_OSVERSION_GWY
DMS_INT32	lGwySubType Subtype of gateway	DMS_SUBTYPE_UNKN_GWY DMS_SUBTYPE_P_GWY DMS_SUBTYPE_DDE_GWY
DMS_UINT32	ulIPAddress	
DMS_INT32	OwnStationNo: Own station number is required only if the station is operating as a server station.	
DMS_INT32	lRedFlag	DMS_STATION_PRIMARY DMS_STATION_SECONDARY

Variable structure

The variable structure is called with the following return codes:

- E_DMSAPI_OK: All OK
- E_DMSAPI_ABORT: Connection aborted

Data type	Component
DMS_HANDLE	D MSHandle
DMS_INT16	MaxNoOfVar: Maximum number of variables

Data type	Component
DMS_INT16	ActNoOfVar: Number of assigned variables
DMS_INT16	FreeBytes: Number of free bytes in list
DMS_VAR_ELEM	DMSVarElem: A table with MaxNoOfVar entries, of which ActNoOfVar are valid.

The structure of DMS_VAR_ELEM

Data type	Component	Range of values
DMS_OBJPATH	DMS-Addressing: ObjNumber Cmp-Number	
DMS_CHAR	VarName	Variable name or NULL
DMS_UINT32	ValueSize	
DMS_VAR_RC	VarRc: ReturnCode for this 1 variable	
DMS_VAR_TYPE	VarType: Type of variable	DMS_VAR_TYPE_WORD16 DMS_VAR_TYPE_WORD32
DMS_VAR_STATUS	VarStatus	DMS_NOT_VALID DMS_CHANGED DMS_NOT_CHANGED DMS_DE-LETED
DMS_VALUE	VarValue	If the variable is invalid or deleted, a NullPointer is passed here.

Information report structure

The variable structure is only called with the following return code:

Data type	Component	Range of values
DMS_INT32	IRId	MSR supports: - DMSAPI_CP_ID - DMSAPI_SAP_ID
DMS_INT32	Buffer (application-dependent)	

The structure for DMSAPI_CP_ID and DMSAPI_CP_ID

Data type	Component	Range of values
DMS_INT16	NoOfVar: Number of variable elements	
DMS_OBJNO	ObjNumber	Object number of block
DMS_VAR_TYPE	VarType[6]	Data types of variables
DMS_INT16	ContentLen	Length of data
DMS_DT	StartEventTime for archiving	
DMS_BYTE	Data: - NoOfVar Time stamp - Data	DMS_DT

Alarm structure

The alarm structure is only called with the following return code:

- E_DMSAPI_OK: All OK

Data type	Component
DMS_INT32	NoOfAlarm Number of alarm elements
DMS_ALARM_ELEM	DMSAlarmElem

The structure of **DMS_ALARM_ELEM**

Data type	Component
DMS_APATH	DMS-Addressing: • ObjNumber • AlarmNumber
DMS_DT	Alarm time
DMS_WORD16	Alarm type
DMS_ASTATCurr	AlarmStatus
DMS_ASTATPrev	AlarmStatus
DMS_BOOLEAN	NotificationLost
DMS_VALUE	lpMsgValue

Alarm acknowledgement structure

The alarm structure is called with the following return codes:

- E_DMSAPI_OK: All OK
- E_DMSAPI_ABORT: Connection aborted

Data type	Component
DMS_INT32	NoOfAckAlarm: Number of acknowledged alarms
DMS_ALARM_ACK_ELEM	DMSAlarmElem

The structure of **DMS_ALARM_ELEM**

Data type	Component
DMS_APATH	DMS-Addressing: ObjNumber and AlarmNumber
DMS_ASTAT	CurrAlarmStatus
DMS_RC	rc: Return code for individual acknowledgement

Download structure

The download structure is called with the following return codes:

- E_DMSAPI_OK: All OK
- W_DMSAPI_DL_IS_RUNNING: Download still running
- E_DMSAPI_ABORT: Connection is aborted
- E_DMSAPI_DOWNLOAD_ABORT: Server has aborted the download
- E_DMSAPI_INVALID_CONF: Server was unable to install/delete domain

Data type	Component	Range of values
DMS_INT32	DMSHandle	
DMS_INT32	Percent: Download status in percent	0 -100

Program invocation structure

The “program invocation” structure is called with the following return codes:

- E_DMSAPI_OK: All OK
- E_DMSAPI_ABORT: Connection is aborted
- E_DMSAPI_INVALID_CONF: Server could not find PI
- E_DMSAPI_PI_IN_USE: Server was unable to execute PI as it is just being executed

Data type	Component
DMS_INT32	DMSHandle

Version structure

The version structure is only called with the following return code:

- E_DMSAPI_OK: All OK

Each time a reconfiguration is carried out, every Callback function is informed for which resource a reconfiguration has been performed. After reconfiguration, name

management has new values. The situation can arise where the current variables lists are invalid, or refer to different variables.

Data type	Component
DMS_INT32	ResourceNo
DMS_CHAR	Project name
DMS_INT32	MajorVersionNo
DMS_INT32	MinorVersionNo
DMS_INT32	PatchVersionNo
DMS_INT32	MaxObjN

3.7.2 Synchronous functions

All functions which perform an action on the server and thus occasion the server to send a response can be called synchronously. The function only returns when the response is present. The response buffer is made available by the application. The response is returned in the transfer structure.

DMSAPI_Receive (ReceiveTimeOut,&RecStruct)

All functions which perform an action on the server and thus occasion the server to send a response can be called asynchronously. The function returns immediately after sending. If no Callback function is installed for the function (CBId set to DMS_NO_CALLBACK), then the response must be retrieved via the DMSAPI_Receive function. The response buffer is made available by the application. The response is returned in the transfer structure.

3.7.3 DMSAPI_RegisterCltCB

SYNTAX

```
DMS_RC DMSAPI_RegisterCltCB(
   DMS_INT16              nCBId          /* CallbackId */,
   DMS_REC_DATA_PROC   CallBackProc/* Callbackfunction */
);
```

This procedure registers a callback function with a specific CallbackId. The registered Callback function is called when data are received. On connecting and disconnecting all registered Callback functions are called.

Registration of the various Callback functions ought to/must be carried out before DMSAPI_Init is called. Immediately after initialisation the DMS is active and from that point on can be connected with client stations.

These connections are also displayed to the applications through the connection structure in the Callback functions.

Parameters:

- nCBId: CallbackId for which the following procedure is installed.
- CallBackProc: Callback function that is called up when data for the CBID are received. If NULL is passed as a parameter, then the Callback function is de-installed.

```
typedef DMS_RC (* DMS_REC_DATA_PROC)
(DMS_REC_DATA *DmsRec);
```

Possible return values:

Function	Description
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_DUPLICATE_CALLBACK	A function is already installed for the specified CallbackId.

Function	Description
E_DMSAPI_NO_RESOURCE	The maximum number of Callback functions is registered.
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_RegisterFreeCltCB (&CBID, (*DMSRC) (Fnc(&RecStruct)))

This procedure registers a Callback function and returns a free CallbackId. The registered Callback function is called when data are received. On connecting and disconnecting all registered Callback functions are called.

Registration of the various Callback functions ought to/must be carried out before DMSAPI_Init is called. Immediately after initialization the DMS is active and from that point on can be connected with client stations.

These connections are also displayed to the applications through the connection structure in the Callback functions.

Parameters:

- CBId: CallbackIdentification for which the following procedure is installed.
- Fnc: Callback function that is called up when data for the CBId are received. If NULL is passed as a parameter, then the Callback function is de-installed.

Possible return values:

Function	Description
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CALLBACK	The specified callback function is not installed.
E_DMSAPI_DUPLICATE_CALLBACK	A function is already installed for the specified CallbackId.
E_DMSAPI_NO_RESOURCE	The maximum number of Callback functions is registered.
E_DMSAPI_INTERNAL_ERROR	Internal error

3.7.4 Callback function (&RecStruct)

If the DMS facility is called for a Callback function, the response is passed to the Callback function. After the Callback function is completed, the data range is invalid. If the Callback function produces a return code other than null, then the connection is aborted and then re-established.

In multitasking (or multithreading) environments the Callback function can be called “simultaneously” from different task contexts. In this function it is also possible to await the arrival of events. However, the event in question must not be the receipt of further data on the same connection. Only the other connections continue to be used.

When connecting and disconnecting, all Callback functions are always called. The connection to Freelance Engineering is then also indicated.

Likewise, each time a change is made to the configuration, all Callback functions are called. Now the application itself must decide what needs to be done. For example, which functions have just failed to perform correctly because of the incorrect configuration, and can now be repeated. Or again, which variables need to be re-read because the internal Freelance address details have changed. Configuration changes can be prevented by locking the object library. In this case Freelance Engineering announces that the downloads cannot be carried out.

4 Name management

Name management is only available on stations that have been configured and loaded as gateways in Freelance Engineering. The receipt of the files (domains) to be loaded by Freelance Engineering takes place through the server procedures of domain management. The domains are coded in binary.

Name management applies only to one resource, i.e. if the DMSAPI is being run for several resources simultaneously, those resources will have different address spaces.

E.g.:

Resource 1 manages Project1

Resource 2 manages Project2

The re-configuration of Freelance Engineering can be prevented if the object library is locked. For this purpose the following functions are provided: DMSAPI_LockOV and DMSAPI_UnlockOV.

Name management has access to the following information:

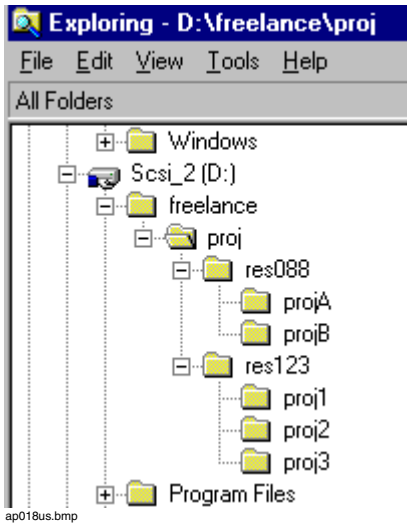
- Version information: Freelance Engineering checks whether or not the version of the connected gateway corresponds with the configured version.
- Station information - including, amongst other details, IP addresses and versions - for all the stations in a Freelance project.
- Variable information on all variables for which read or write access has been configured in Freelance Engineering.
- Tag information on all tags for which read or write access has been configured in Freelance Engineering.
- Information on Freelance object classes, with all the component names for the addressable variables in a block.

4.1 File directory

Before the DMSAPI is initialized the application can define a gateway directory, under which a directory tree is created. Under this directory a separate directory is set up for each resource. The directory is assigned the name Res<OwnResNo>. In this directory a file is stored with the name “Proj.dom”, containing the project used most recently for the resource. Under that directory a separate directory is set up for each project, called by the project name. The following files are downloaded from Freelance Engineering and stored in the directory:

- vers.dom: File containing details of its own project version
- ov.dom: File containing information on the size of the object library and the load status of the separate objects
- stat.dom: File containing details of the various resources
- version.dom: File containing version information on the various resources.
- tag.dom: File containing details of the tags
- var.dom: File containing details of the variables
- fb<Num>.dom: File containing details relating to the different object classes such as function blocks, SFC and user-defined structures.

In the example below, in Windows Explorer a gateway directory is set up under **<Freelance\proj>**. Under this directory 2 gateways have been set up with the resource numbers 88 and 123. Resource 88 has been loaded by Freelance Engineering with projects projA and projB, while resource 123 has been loaded by a different Freelance Engineering with projects proj1, proj2 and proj3.



4.1.1 DMSAPI_SetProjectDir

SYNTAX

DMS_RC DMSAPI_SetProjectDir(

DMS_CHAR * szProjectDir

);

If the DMSAPI is run on a computer with a hard disk and if the gateway configuration is downloaded on that computer, then the gateway directory can be set using the function **DMSAPI_SetProjectDir**. The result of this (setting) operation is that all Freelance domains are installed under this directory. If the project directory is changed through calling this function, connections to all client stations (generally Freelance Engineering) are automatically aborted. The project directory is set to the same directory for all resources. Freelance Engineering cannot load a configuration until the project directory is set.

Parameter:

- szProjectDir: validdirectory

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialised.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INVALID_DIR	The directory cannot be accessed
E_DMSAPI_CONFIGURING	Freelance Engineering is currently loading the configuration
E_DMSAPI_INTERNAL_ERROR	Internal error

4.1.2 DMSAPI_ChangeProject

SYNTAX

```
DMSAPI_ChangeProject(
    DMS_RES_NO          OwnResNo          /* Own resource number */,
    DMS_CHAR *szProjectName          /* Project name */
)
```

If the DMSAPI is run on a computer with a hard disk and if the gateway configuration is downloaded on that computer, then the gateway directory can be set using the function DMSAPI_ChangeProject. The result of this (setting) operation is that all Freelance domains are installed under this directory. If the project directory is changed through calling this function, connections to all client stations (generally Freelance Engineering) are automatically aborted. If the specified project is not available, then the name management functions do not return any values. During initialization and Load Entire Station Freelance Engineering automatically sets the project directory to the Freelance Engineering project name.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- szProjectname: The project name must be a valid file name

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialised.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_CONFIGURING	Freelance Engineering is currently loading the configuration.
E_DMSAPI_INVALID_DIR	The directory cannot be accessed
E_DMSAPI_INTERNAL_ERROR	Internal error

4.2 Project information

4.2.1 DMSAPI_GetProjectInfo

SYNTAX

```
DMS_RC DMSAPI_GetProjectInfo(
        DMS_RES_NO          OwnResNo          /* Own resources */,
        DMS_VERSION_DATA* lpVersionData       /* Version data */
);
```

The following information is provided if the DMSAPI was initialised using the function DMSAPI_Init. Each time there is a download from Freelance Engineering all registered Callback functions automatically receive these details.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called. lpVersionData:

```
typedef struct DMS_VERSION_DATA {
        DMS_CHAR        *ProjName;          /* Project name */
        DMS_WORD16  wMajorVersion;          /* Major version number */
        DMS_WORD16  wMinorVersion;          /* Minor version number */
```

```
        DMS_WORD16  wPatchVersion;          /* Patch version number */
} DMS_VERSION_DATA;
```

The project name can be changed using the function DMSAPI_ChangeProject. If the current project name is different from the project that Freelance Engineering is processing and if the gateway station is loaded from Freelance Engineering, the gateway station is automatically allocated a project name and subsequent version numbers from Freelance Engineering.

The major version number changes each time “Load - entire station” is run from Freelance Engineering. The old value is incremented.

The minor version number changes with each object that is downloaded by Freelance Engineering. The old value is incremented.

The patch version number does not change at the gateway. It remains constant at 0.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_INTERNAL_ERROR	Internal error

4.3 Locking “Name management”

4.3.1 DMSAPI_LockOV

SYNTAX

```
DMS_RC DMSAPI_LockOV(
        DMS_RES_NO      OwnResNo    /* Own resources */
);
```

If the aim is to prevent Freelance Engineering being reconfigured over a specific period of time, this can be achieved by locking the object library. Freelance Engineering then indicates that a configuration cannot be loaded to this gateway.

Parameter:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_ALREADY_DONE	Object library is already locked
E_DMSAPI_INTERNAL_ERROR	Internal error

4.3.2 DMSAPI_UnlockOV

SYNTAX

```
DMSAPI_UnlockOV
                DMS_RES_NO      OwnResNo    /* Own resources */
 );
```

If reconfiguration by Freelance Engineering is to be allowed again after the object library has been locked, this procedure should be called up.

Parameter:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_ALREADY_DONE	Locking of OL has already been cancelled
E_DMSAPI_INTERNAL_ERROR	Internal error

4.4 Station information

The following 2 binary domains are received via the server procedures of domain management:

Stationname	IP-Address 1	IP-Address 2	StatNo	StatType	TimeOut
DPS1	172.16.1.2	172.16.1.3	2	RED_MSR	45
DPS2	172.16.1.4	0.0.0.0	3	MSR	120
GWY1	172.16.1.5	0.0.0.0	4	GWY	15
....					

StationNo	MajorVersion	MinorVersion	PatchVersion
2	142	340	0
3	10	223	2
....			

Access to this station information is controlled through the following two procedures:

- DMSAPI_GetFirstResoureInfo
- DMSAPI_GetNextResourceInfo

!!!

When these functions are used on multitasking operating systems, care should be taken in the application to ensure that suitable locking mechanisms are provided. This means that if the GetNext function is called from two tasks alternately, then both tasks do not receive all the elements from the station domain, but rather they receive the next elements from the domain alternately.

!!!

After this station domain is reconfigured by Freelance Engineering, the GetFirst procedure must always be called again. Otherwise the GetNext routine will return an error.

4.4.1 DMSAPI_GetFirstResourceInfo

SYNTAX

```
DMS_RC DMSAPI_GetFirstResourceInfo(
        DMS_RES_NO              OwnResNo   /* Own ResNum */,
        DMS_UINT32    *lpulNoOfRes         /* Number of resources */,
        DMS_UINT32     ResNameLen          /* Max. length resname */,
        DMS_CHAR               *lpResName  /* Res name */,
        DMS_NAME_RESOURCE_DATA      *lpResInfo     /* Res info */
);
```

This procedure returns the details of the first resource in the resource domain.

Parameters:

- OwnResNo: Own station’s resource number. The DMSAPI_Init must be called.
- lpulNoOfRes: The number of available resources is returned
- ResNameLen: Length of the next buffer in sequence (DMS_MAX_RESNAME_LEN)
- lpResName: Buffer for resource name

- lpResInfo: Resource information

```
typedef struct DMS_NAME_RESOURCE_DATA {
        DMS_WORD32     dwIPAddr1;
        DMS_WORD32      dwIPAddr2;
        DMS_RES_NO      ResNo;
        DMS_RES_TYPE     ResType;
        DMS_UINT16      wTimeOut; /* in sec. */
        DMS_UINT16      wMajorVersionNo;
        DMS_UINT16      wMinorVersionNo;
        DMS_UINT16      wPatchVersionNo;
} DMS_NAME_RESOURCE_DATA;
```

The ResType can take the following values:

```
DMS_OS_DIGIVIS
DMS_OS_DIGITOOL
DMS_OS_EPROM
DMS_OS_MSR
DMS_OS_DDE_GWY
DMS_OS_P_GWY
DMS_OS_GWY
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available

Function	Description
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	No station available in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

4.4.2 DMSAPI_GetNextResourceInfo

SYNTAX

```
DMS_RC DMSAPI_GetNextResourceInfo(
        DMS_RES_NO          OwnResNo       /* Own res number */,
        DMS_UINT32          ResNameLen     /*Max. length resname */,
        DMS_CHAR            *lpResName     /* Res name */,
DMS_NAME_RESOURCE_DATA      *lpResInfo     /* Res info */
);
```

This procedure returns the details of the other resources in the resource domain.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- ResNameLen: Length of the next buffer in sequence
- lpResName: Buffer for resource name
- lpResInfo: Resource information

```
typedef struct DMS_NAME_RESOURCE_DATA {
        DMS_WORD32      dwIPAddr1;
        DMS_WORD32      dwIPAddr2;
        DMS_RES_NO      ResNo;
        DMS_RES_TYPE    ResType;
        DMS_UINT16      wTimeOut; /* in sek. */
```

```
        DMS_UINT16      wMajorVersionNo;
        DMS_UINT16      wMinorVersionNo;
        DMS_UINT16      wPatchVersionNo;
} DMS_NAME_RESOURCE_DATA;
```

The ResType can take the following values:

```
DMS_OS_DIGIVIS
DMS_OS_DIGITOOL
DMS_OS_EPROM
DMS_OS_MSR
DMS_OS_DDE_GWY
DMS_OS_P_GWY
DMS_OS_GWY
```

Parameters:

- OwnResourceNo: Own station's resource number. The DMSAPI_Init must be called.
- StationNameLen: Length of the next buffer in sequence
- lpStationName: Buffer for station name
- lpStationInfo: Station information

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_NO_ACCESS	Function: GetFirst was not called
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.

Function	Description
E_DMSAPI_EMPTY_CONF	No other stations available in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

4.5 Variable information

The following binary domain is received via the server procedures of domain management:

Variable name	Data type	Access	StatNo	ObjNo	CompNo
ANA_E	DIGI_FLOAT3 2	R	1	131	1
BIN_A1	DIGI_BOOLEAN	R/W	2	131	2
...					

Access to this variable information is controlled through the following two procedures:

- DMSAPI_GetFirstVarInfo
- DMSAPI_GetNextVarInfo

!!!

When these functions are used on multitasking operating systems, care should be taken in the application to ensure that suitable locking mechanisms are provided. This means that if the GetNext function is called from two tasks alternately, then both tasks do not receive all the elements from the variable domain, but rather they receive the next elements from the domain alternately.

!!!

After this station domain is reconfigured by Freelance Engineering, the GetFirst procedure must always be called again. Otherwise the GetNext routine will return an error.

4.5.1 DMSAPI_GetFirstVarInfo

SYNTAX

```
DMS_RC DMSAPI_GetFirstVarInfo(
        DMS_RES_NO        OwnResNo    /* Own resource number */,
        DMS_UINT32        *lpulNoOfVar  /* Number of var */,
        DMS_UINT32        VarNameLen        /*Max. length var name */,
        DMS_CHAR          *lpVarName        /* Variable name */,
        DMS_NAME_VAR_DATA*lpVarInfo     /* Variable Info */);
```

This procedure returns the details of the first variable in the variable domain.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- lpulNoOfVar: The number of available variables is returned
- VarNameLen: Length of the next buffer in sequence (DMS_MAX_VARNAME_LEN)
- lpVarInfo: Variable information

```
typedef struct DMS_NAME_VAR_DATA {
        DMS_WORD32              dwAccessRights;
        DMS_VAR_TYPE      VarType;
        DMS_RES_NO        ResNo;
        DMS_OBJ_PATH      Opath;
} DMS_NAME_VAR_DATA;
```

dwAccessRights can take the following values:

```
        DMS_READ_ONLY
        DMS_READ_WRITE
```

VarType can take various different values. (see Appendix DMS Variable Types)

Opath

typedef struct {

DMS_OBJNOObjNo;

DMS_CMPNOCmpNo}

DMS_OBJ_PATH;

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	No variables available in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

4.5.2 DMSAPI_GetNextVarInfo

SYNTAX

```
DMS_RC DMSAPI_GetNextVarInfo(
        DMS_RES_NO      OwnResNo    /* Own resource number */,
        DMS_UINT32      VarNameLen  /* Max. length var name */,
        DMS_CHAR            *lpVarName    /* Variable name */,
        DMS_NAME_VAR_DATA*lpVarInfo     /* Variable info */);
```

This procedure returns the details of the other variables in the variable domain.

Parameters:

- OwnResNo: Own station’s resource number. The DMSAPI_Init must be called.

- VarNameLen: Length of the next buffer in sequence (DMS_MAX_VARNAME_LEN)
- lpVarName: Buffer for variable name
- lpVarInfo: Variable information

typedef struct DMS_NAME_VAR_DATA {

DMS_WORD32dwAccessRights;

DMS_VAR_TYPE VarType;

DMS_RES_NO ResNo;

DMS_OBJ_PATH Opath;

} DMS_NAME_VAR_DATA;

dwAccessRights can take the following values:

DMS_READ_ONLY

DMS_READ_WRITE

VarType can take various different values (See Appendix A, Variable types and error codes).

Opath is the DMS addressing on the server:

typedef struct {

DMS_OBJNOObjNo;

DMS_CMPNOCmpNo}

DMS_OBJ_PATH;

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available

Function	Description
E_DMSAPI_NO_ACCESS	Function: GetFirst was not called
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	No other variables available in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

4.6 Tag information

The following binary domain is received through the server procedures of domain management:

MSR name	ResNo	Access	Object class	ObjNo	CmpNo
ANA_Z1	1	R	DIGI_ANA_Z(267)	2689	0
BinOver	1	RW	DIGI_BINOV(279)	2788	0
StructTst	2	RW	Structured var (520)	131	12
.....					

Access to this tag information is controlled through the following two procedures:

- DMSAPI_GetFirstTagInfo
- DMSAPI_GetNextTagInfo

!!!

When these functions are used on multitasking operating systems, care should be taken in the application to ensure that suitable locking mechanisms are provided. This means that if the GetNext function is called from two tasks alternately, then both tasks do not receive all the elements from the tag domain, but rather they receive the next elements from the domain alternately.

!!!

After this tag domain is reconfigured by Freelance Engineering, the GetFirst procedure must always be called again. Otherwise the GetNext routine will return an error.

DMSAPI_GetFirstTagInfo

SYNTAX

```
DMSAPI_GetFirstTagInfo(
        DMS_RES_NO          OwnResNo    /* Own resource number  */,
        DMS_UINT32              *lpulNoOfTag/* Number of tags */,
        DMS_UINT32          TagNameLen/* Max. length tag name */,
        DMS_CHAR                *lpTagName   /* Tag name */,
        DMS_NAME_TAG_DATA*lpTagInfo     /* Tag info*/
);
```

This procedure returns the details of the first tag in the tag domain.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- lpulNoOfTag: The number of available tags is returned
- TagNameLen: Length of the next buffer in sequence (DMS_MAX_TAGNAME_LEN)
- lpTagName: Buffer for tag name
- lpTagInfo: Tag information

```
typedef struct DMS_NAME_TAG_DATA {
        DMS_WORD32              dwAccessRights;
        DMS_RES_NO              ResNo;
        DMS_OBJNO       ObjClass;
```

```
        DMS_OBJNO       ObjNo;
        DMS_CMPNO       CmpNo;
} DMS_NAME_TAG_DATA;
```

dwAccessRights can take the following values:

```
        DMS_READ_ONLY
        DMS_READ_WRITE
```

Possible return values

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	No tags available in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_GetNextTagInfo

SYNTAX

```
DMSAPI_GetNextTagInfo(
        DMS_RES_NO        OwnResNo    /* Own resource number */,
        DMS_UINT32        TagNameLen/* Max. length tag name */,
        DMS_CHAR                  *lpTagName   /* Tag name */,
        DMS_NAME_TAG_DATA*lpTagInfo     /* Tag info*/
);
```

This procedure returns the details of all other tags in the tag domain.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- TagNameLen: Length of the next buffer in sequence (DMS_MAX_TAGNAME_LEN)
- lpTagName: Buffer for tag name
- lpTagInfo: Tag information

```
typedef struct DMS_NAME_TAG_DATA {
        DMS_WORD32      dwAccessRights;
        DMS_RES_NO      ResNo;
        DMS_OBJNO       ObjClass;
        DMS_OBJNO       ObjNo;
        DMS_CMPNO       CmpNo;
} DMS_NAME_TAG_DATA;
```

dwAccessRights can take the following values:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_EMPTY_CONF	No other tags available in the project
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_NO_ACCESS	Function: GetFirst was not called
E_DMSAPI_INTERNAL_ERROR	Internal error

DMSAPI_GetTagByAddr

```
CGEXPORT DMS_RC DMSAPI_GetTagByAddr (
                DMS_RES_NO          OwnResNo     /* Own resource number */,
                DMS_RES_NO          ResNo        /* Res number */,
                DMS_OBJNO           ObjNo        /* Object path */,
                DMS_UINT32          TagNameLen   /* Max. length tag name */,
                DMS_CHAR           *lpTagName /* Tag name */,
                DMS_NAME_TAG_DATA*lpTagInfo     /* Tag info*/
);
```

The procedure returns the details of a “tag”, which is addressed through resource number and object number.

Parameters:

- OwnResNo: Own station’s resource number. The DMSAPI_Init must be called.
- ResNo: Resource number of server station.
- ObjNo: Object number of object being searched for
- TagNameLen: Length of the next buffer in sequence (DMS_MAX_TAGNAME_LEN)
- lpTagName: Buffer for tag name
- lpTagInfo: Tag information

```
typedef struct DMS_NAME_TAG_DATA {
                DMS_WORD32          dwAccessRights;
                DMS_RES_NO          ResNo;
                DMS_OBJNO           ObjClass;
                DMS_OBJNO           ObjNo;
                DMS_CMPNO           CmpNo;
} DMS_NAME_TAG_DATA;
```

dwAccessRights can take the following values:

DMS_READ_ONLY

DMS_READ_WRITE

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_INVALID_CONF	No tags available for the address in the project
E_DMSAPI_INTERNAL_ERROR	Internal error

4.7 Object class position information

Several binary domains are received through the server procedures of domain management:

Analog counter: Object class 267

Variable name	Access	Data type	Component Number
Enable	RW	DIGI_BOOLEAN	1
Input	R	DIGI_FLOAT32	2
...			

Binary monitor: Object class 279

Variable name	Access	Data type	Component Number
Enable	R	DIGI_BOOLEAN	1
...			

Data block: Object class 510 (names are defined by the user)

Variable name	Access	Data type	Component Number
Struct1	RW	Data block 510	1
Struct2	R	Data block 510	n
...			

Data block: Object class 520 (names are defined by the user)

Variable name	Access	Data type	Component Number
Elem1	RW	DIGI_BOOLEAN	1
Elem2	R	DIGI_FLOAT32	2
....			
Elemn	RW	DIGI_INT32	n

In the example, data block 520 represents multi-level addressing:

The multi-level addressing through variable name may thus be as follows:

StructTst/Struct2/Elem2

The component number is then calculated:

Component number of StructTst +

Component number of Struct2 +

Component number of StrucElem2.

Any depth of nesting is possible. Recursion is absolutely not permitted.

- DMSAPI_GetFirstCmpOfObjClass
- DMSAPI_GetNextCmpOfObjClass

!!!

When these functions are used on multitasking operating systems, care should be taken in the application to ensure that suitable locking mechanisms are provided. This means that if the GetNext function is called from two tasks alternately, then both tasks do not receive all the elements from the object class domain, but rather they receive the next elements from the domain alternately.

!!!

After this station domain is reconfigured by Freelance Engineering, the GetFirst procedure must always be called again. The GetNext routine will return an error.

4.7.1 DMSAPI_GetFirstCmpOfObjClass

SYNTAX

```
DMS_RC DMSAPI_GetFirstCmpOfObjClass(
                DMS_RES_NO          OwnResNo     /* Own resource number */,
                DMS_OBJNO           ObjClass          /* Object class */,
                DMS_UINT32          *lpulNoOfCmp   /* Number of components */,
                DMS_UINT32          CmpNameLen   /* Max. length comp. name */,
                DMS_CHAR            *lpCmpName       /* Component name */,
                DMS_NAME_OBJ_DATA*lpObjInfo     /* Object info */
);
```

This procedure returns the details of the first component in the specified object class.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- ObjClass: Object number of sought object class
- lpulNoOfCmp: The number of available components is returned
- CmpNameLen: Length of the next buffer in sequence (DMS_MAX_COMPNAME_LEN)
- lpCmpName: Buffer for component name
- lpObjInfo: Information on the first component in the object class

```
typedef struct DMS_NAME_OBJ_DATA {
        DMS_WORD16      nRWFlag;
        DMS_CMPNO       CmpNo;
        DMS_VAR_TYPE VarType;
        DMS_WORD16      Reserved;
} DMS_NAME_OBJ_DATA;
```

dwAccessRights can take the following values:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_EMPTY_CONF	No object class of this type available

Function	Description
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_INTERNAL_ERROR	Internal error

4.7.2 DMSAPI_GetNextCmpOfObjClass

SYNTAX

```
DMS_RC DMSAPI_GetNextCmpOfObjClass(
        DMS_RES_NO      OwnResNo        /* Own resource number */,
        DMS_OBJNO       ObjClass        /* Object class */,
        DMS_UINT32      CmpNameLen/* Max. length comp. name */,
        DMS_CHAR        *lpCmpName      /* Component name */,
        DMS_NAME_OBJ_DATA*lpObjInfo     /* Object info */
);
```

This procedure returns the details of all other components in the specified object class.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- ObjClass: Object number of sought object class
- CmpNameLen: Length of the next buffer in sequence (DMS_MAX_COMPNAME_LEN)
- lpCmpName: Buffer for component name
- *lpObjInfo Information on the first component in the object class

```
typedef struct DMS_NAME_OBJ_DATA {
        DMS_WORD16      nRWFlag;
        DMS_CMPNO       CmpNo;
        DMS_VAR_TYPE VarType;
```

```
        DMS_WORD16      Reserved;
} DMS_NAME_OBJ_DATA;
```

dwAccessRights can take the following values:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	No further information available on this object class
E_DMSAPI_INTERNAL_ERROR	Internal error

4.8 Address conversion

In addition to these basic functions there is the opportunity to convert “variable names” to “Freelance object path” and vice versa.

4.8.1 DMSAPI_GetVarNameByOPath

SYNTAX

```
DMS_RC DMSAPI_GetVarnameByOPath (
        DMS_RES_NO      OwnResNo    /* Own resource number */,
        DMS_RES_NO      ResNo       /* Res number */,
        DMS_OBJ_PATH    *lpOPath    /* Object path */,
        DMS_UINT32      VarNameLen  /* Max. length var name */,
```

DMS_CHAR *lpVarName /* Variable name */);

The procedure converts an object path into a variable name.

Parameters:

- OwnResNo: Own station's resource number. The DMSAPI_Init must be called.
- ResNo: Resource number of server station.
- lpOPath: Object path of the variable being searched for

typedef struct {

DMS_OBJNO ObjNo;

DMS_CMPNOCmpNo;

} DMS_OBJ_PATH;

- VarNameLen: Length of the next buffer in sequence (DMS_MAX_VARNAME_LEN)
- lpVarName: Buffer for variable name

Possible return values:

Function	Description
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_SMALL_RCV_BUFF	The buffer passed is too small.
E_DMSAPI_INVALID_CONF	No information available about the variable
E_DMSAPI_INTERNAL_ERROR	Internal error

4.8.2 DMSAPI_GetVarInfoByName

SYNTAX

```
DMS_RC DMSAPI_GetVarInfoByName (
                DMS_RES_NO      OwnResNo      /* Own resource number  */,
                DMS_CHAR        *lpVarName    /* Variable name */,
                DMS_RES_NO      *lpResNo      /* Res number */,
                DMS_OBJ_PATH    *lpOPath      /* Object path */,
                DMS_VAR_TYPE    *lpVarType    /* DigiType */,
DMS_WORD32              * lpAccessRights      /* Access Rights */
);
```

The procedure converts a variable name into an “object path”. Variable name in this situation does not have to refer to a variable received through functions GetFirstVar and GetNextVar, it can also refer to a variable composed of “date” and component name.

Parameters:

- OwnResNo: Own station’s resource number. The DMSAPI_Init must be called.
- lpVarName: Name of the variable being searched for
- lpResNo: Resource number of server station.
- lpOPath: Object path of the variable being searched for

```
typedef struct {
                DMS_OBJNO ObjNo;
                DMS_CMPNOCmpNo;
} DMS_OBJ_PATH;
```

- lpVarType can take various different values (See DMS variable types on page 147).
- lpAccessRights can take the following values:

DMS_READ_ONLY

DMS_READ_WRITE

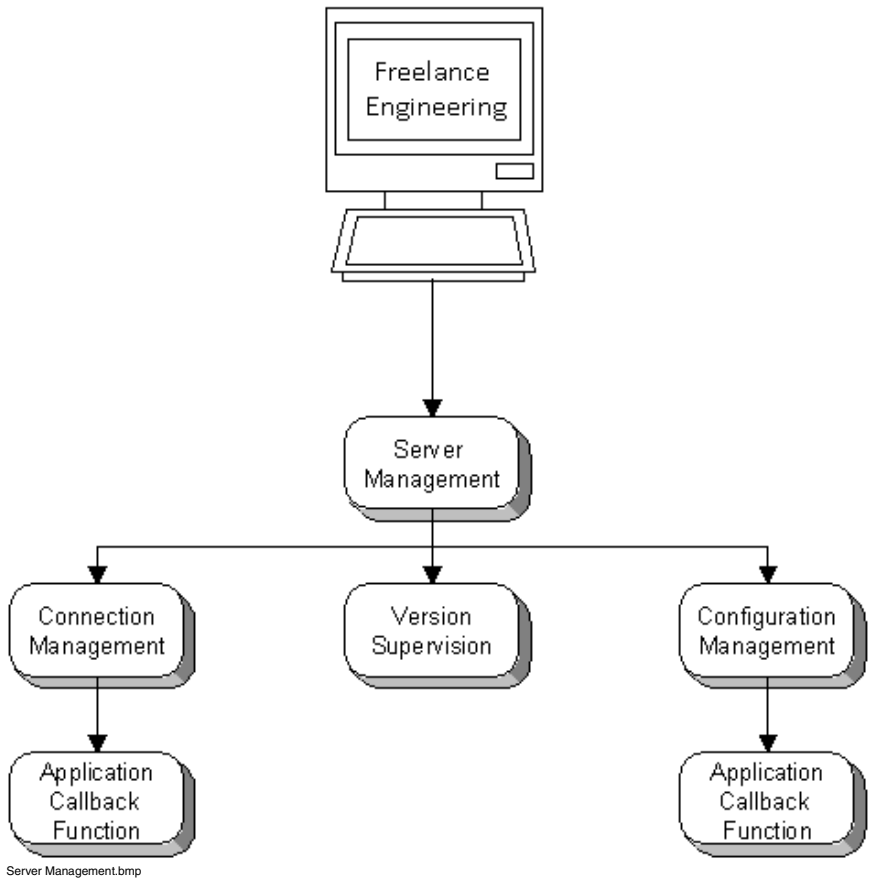
Possible return values:

Function	**Description**
E_DMSAPI_NOT_INIT	The function was called although the DMS layer for this resource number was not initialized.
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_NO_CONF	No project available
E_DMSAPI_INVALID_CONF	No information available about the variable
E_DMSAPI_INTERNAL_ERROR	Internal error

5 Server management

There are pre-defined functions that can be used to perform the server management required by Freelance Engineering:

- Storing the configuration domain for name management on disk
- Starting/stopping the DMS during a reconfiguration
- Reading the version information.



Server Management.bmp

6 DMS utilities

6.1 DMSAPI_GetStringByValue

SYNTAX

```
DMS_RC DMSAPI_GetStringByValue (
   DMS_UINT32      ulStrLen       /* Size of storage area referenced to   the pointer
                                                                          */,
   DMS_CHAR        *lpszString  /* Storage for string */,
   DMS_VAR_TYPEVarType        /* Variable type */,
   DMS_VALUE       *lpvVarValue    /* Value of variable*/
);
```

The procedure converts a Freelance value into a printable string.

Parameters:

- ulStrlen: Maximum length of buffer
- lpszString: Buffer for string
- Vartype: Type of value
- lpvVarValue: Freelance value

Possible return values:

Function	Description
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_SMALL_RCV_BUFF	Buffer too small
E_DMSAPI_INTERNAL_ERROR	Internal error

6.2 DMSAPI_GetValueByString

SYNTAX

```
DMS_RC DMSAPI_GetValueByString(
   DMS_UINT32     ulValLen       /* Size of storage area referenced to
                                                the pointer */,
   DMS_VALUE      *lpvVarValue/* Storage for value of variable */,
   DMS_VAR_TYPEVarType       /* Variable type */,
   DMS_CHAR       *lpszString    /* Value as a string */
);
```

The procedure converts a string that has been read in into a Freelance value.

Parameters:

- ulValLen: Maximum length of buffer
- lpvVarValue: Buffer for value
- Vartype: Type of value
- lpvVarValue: Freelance value as a string

Possible return values:

Function	Description
E_DMSAPI_INVALID_ARG	The parameters passed are invalid.
E_DMSAPI_SMALL_RCV_BUFF	Buffer too small
E_DMSAPI_INTERNAL_ERROR	Internal error

6.3 DMSAPI_GetVarLen

SYNTAX

int DMSAPI_GetVarLen (

DMS_VAR_TYPE VarType /* Variable type */

);

The procedure returns for a Freelance data type the amount of storage required in a variables list by a variable of that type.

Parameters:

- VarType can take various different values. (See DMS variable types on page 147)

Possible return values:

The length of the data type in bytes, or -1 if an invalid data type is passed.

6.4 DMSAPI_DumpRecData

SYNTAX

void DMSAPI_DumpRecData (

DMS_REC_DATA * DmsRecData /* */

);

The procedure passes (dumps) the structure of a receive structure to standard output.

Parameter:

- RecData: Receive data

Appendix A Variable types and error codes

A.1 DMS variable types

In the DMSAPI, variables always consist of a type and a value. The variable types can take the following values:

Definition of variable type	Value for variable type	Typedef in UNION DMS_VALUE
DMS_VAR_TYPE_BOOLEAN	0x01	DMS_BOOLEAN Boolean; typedef unsigned char DMS_BOOLEAN
DMS_VAR_TYPE_CHAR	0x02	DMS_CHAR Char; typedef char DMS_CHAR
DMS_VAR_TYPE_BYTE	0x03	DMS_BYTE Byte; typedef unsigned char DMS_BYTE
DMS_VAR_TYPE_INT8	0x04	DMS_INT8 Int8; typedef char DMS_INT8
DMS_VAR_TYPE_WORD16	0x05	DMS_WORD16 Word16; typedef unsigned short DMS_WORD16
DMS_VAR_TYPE_UINT16	0x06	DMS_UINT16 Uint16; typedef unsigned short DMS_UINT16
DMS_VAR_TYPE_INT16	0x07	DMS_INT16 Int16; typedef short DMS_INT16
DMS_VAR_TYPE_WORD32	0x08	DMS_WORD32 Word32; typedef unsigned long DMS_UINT32
DMS_VAR_TYPE_UINT32	0x09	DMS_UINT32 Uint32; typedef unsigned long DMS_UINT32
DMS_VAR_TYPE_INT32	0x0A	DMS_INT32 Int32; typedef long DMS_INT32

Definition of variable type	**Value for variable type**	**Typedef in UNION DMS_VALUE**
DMS_VAR_TYPE_FLOAT32	0x0B	DMS_FLOAT32 Float32; typedef float DMS_FLOAT32
DMS_VAR_TYPE_TIME	0x0C	DMS_TIME DmsTime; typedef long DMS_INT32
DMS_VAR_TYPE_DT	0x0D	DMS_DT DmsDT; /* ms since 1.1.1970 0.00 hrs GMT */ typedef struct { DMS_WORD32 dwMSecondsHigh; DMS_WORD32 dwMSecondsLow; } DMS_DT
DMS_VAR_TYPE_STRING8	0x0E	DMS_STRING8 String8; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[10]; } DMS_STRING8
DMS_VAR_TYPE_STRING16	0x0F	DMS_STRING16 String16; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[18]; } DMS_STRING16
DMS_VAR_TYPE_STRING32	0x10	DMS_STRING32 String32; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[34]; } DMS_STRING32

Definition of variable type	**Value for variable type**	**Typedef in UNION DMS_VALUE**
DMS_VAR_TYPE_STRING64	0x11	DMS_STRING64 String64; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[66]; } DMS_STRING64
DMS_VAR_TYPE_STRING128	0x12	DMS_STRING128 String128; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[130]; } DMS_STRING128
DMS_VAR_TYPE_STRING256	0x13	DMS_STRING256 String256; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[258]; } DMS_STRING256
DMS_VAR_TYPE_OBJNO	0x2C	DMS_OBJNO ObjNo; typedef unsigned long DMS_UINT16
DMS_VAR_TYPE_CMPNO	0x2D	DMS_CMPNO CmpNo; typedef unsigned long DMS_UINT16

A.2 DMS error codes

Definition of error	Value for error	Description of error
E_DMSAPI_OK	0x00000000	No error
E_DMSAPI_NOT_INIT	0x00000001	The DMSAPI is not initialized for the specified resource
E_DMSAPI_INVALID_CONF	0x00000002	No configuration available for specified names
E_DMSAPI_INVALID_ARG	0x00000003	Function was called with an invalid parameter
E_DMSAPI_SMALL_RCV_BUFF	0x00000004	The buffer passed is too small.
E_DMSAPI_EMPTY_CONF	0x00000005	No information is available for the specified name management class
E_DMSAPI_INTERNAL_ERROR	0x00000006	An internal DMS error has occurred. For reasons of safety the application should be closed as quickly as possible, and with the minimum possible data loss.
E_DMSAPI_ACCESS_ERROR	0x00000007	The specified station or variable cannot be accessed.
E_DMSAPI_NO_CONF	0x00000008	No configuration is available for the specified resource.
E_DMSAPI_INVALID_DMS_HANDLE	0x00000009	The DMS-Handle passed is invalid
E_DMSAPI_INVALID_CONN_HANDLE	0x0000000a	The ConnectionHandle passed is invalid
E_DMSAPI_NO_RESOURCE	0x0000000b	The DMS does not have any resources at present. The situation can occur where resources cannot be returned because Callback functions are blocking the application.
E_DMSAPI_VARLIST_IN_USE	0x0000000c	A variable list cannot be modified while a function is not yet completed.

Definition of error	Value for error	Description of error
E_DMSAPI_NO_CALLBACK	0x0000000d	No Callback function is specified for the CallbackId passed.
E_DMSAPI_DUPLICATE_CALLBACK	0x0000000e	A callback function has already been logged under the specified CallbackId.
E_DMSAPI_INVALID_INDEX	0x00000000f	There is no valid variable in the variables list under the specified variable.
E_DMSAPI_INVALID_VARTYPE	0x00000010	The value of the variable type is invalid
E_DMSAPI_INVALID_VARMODE	0x00000011	The variables list was created for a function, and is now to be used for a different function.
E_DMSAPI_NO_CONNECTION	0x00000012	No connection to the specified station
E_DMSAPI_ALREADY_INIT	0x00000013	The DMSAPI has already been initialized for this station
E_DMSAPI_MAX_APPLICATION	0x00000014	The DMSAPI can only be initialized for a specific number of resources
E_DMSAPI_MAX_CONNECTION	0x00000015	The DMSAPI can only establish connections to a specific number of resources.
E_DMSAPI_TIMEOUT	0x00000016	The function could not be executed within the specified timeout
E_DMSAPI_INVALID_DIR	0x00000017	The specified directory does not exist.

Appendix B Application interface freelance examples

B.1 DMSAPI samples

When the DMSAPI is installed, the samples will be copied under the designated Freelance directory (for example, **c:\Freelance**) by Setup as follows:

... \dmsapi\ - incluce

- lib

- samples

B.2 Variable access services

B.2.1 One-time read “read.c”

```
/*
*/


#if 0
FILENAME        $Workfile:   read.c $
VERSION           $Revision:   1.0 $(0)
HISTORY
HISTORY_END
/* $Log: read.c_v $
*/
#endif
```

```
/*
   Demo program for DMSAPI-communication (Windows):
      – Calling convention: dmsard <OwnStationNo> <Variablename>
      – Init of DMSAPI
      – Register of a Callback-Function
      – Connect to a Station
      – Create a VariableList
      – In a loop the Variable given as argument will be read once with async and
        once with sync option
*/
#include <windows.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include "dmstyp.h"
#include "dmsapi.h"
#include "dmserr.h"
int StationConnect=0;
int ReadFlag=0;
DMS_RC OwnDMSAPICallback (DMS_REC_DATA *lpDmsRec) {
/* Callback-function called by DMSAPI
Attention: this function is called in the context of a communication thread
```

```
which has a higher priority than the main thread
you have to protect your data and code!
*/
                    DMS_REC_VARLIST_DATA *lpVarList;
                    int i;
   switch (lpDmsRec->SrvType) {
      case DMS_REC_CONN_TYPE:
           /* DMSAPI calls Callback every time a station connects or
              disconnects */
         if (!lpDmsRec->DmsRc)
            StationConnect=1;
         else
            StationConnect=0;
         break;
      case DMS_REC_VARLIST_TYPE:
          /* case value for a received variable value */
         DMSAPI_DumpRecData(lpDmsRec);
         ReadFlag=1;
lpVarList = lpDmsRec->SrvBuff.lpVarList;
for (i = 0; i < lpVarList->MaxVarNo; i++)
{
if (lpVarList->lpVar[i].VarStatus!= DMS_VAR_DELETED)
{
if (lpVarList->lpVar[i].VarRc)
{
/* DMSAPI reports an error in Read Operation */
```

```
                    } else
                    if (lpVarList->lpVar[i].VarStatus!= DMS_VAR_NOT_VALID)
                    {
                    /* Read was successful :
                     Datatype in lpVarList->lpVar[i].VarType,
                    Value in    lpVarList->lpVar[i].VarValue
                    */
                                                                    }
                                                                    }
                                                            }
            break;
        default:
           printf ("unexpected Case\n");
}
   return(0);
}
/*-----------------------------------------------------------------
      If there is no valid config in c:\digimat\gwy\resxxx
      this function waits for config from Freelance Engineering
      you can change the ProjectDir by SetProjectDir
      before calling DMSAPI_Init
-----------------------------------------------------------------*/
void WaitForConfig(DMS_RES_NO OwnResNo) {
DMS_CHAR              Resname[10];
   DMS_UINT32            NoOfRes;
   DMS_NAME_RESOURCE_DATA ResInfo;
```

```
    if (DMSAPI_GetFirstResourceInfo(OwnResNo,&NoOfRes,10,Resname,
        &ResInfo)) {
      printf ("No Config for GWY-Id %d: Configure from Control Builder F and
press any key to continue\n",
          OwnResNo);
      for (;;) {
        Sleep(100);
        if (kbhit()) {
          getch();
          break;
        }
      }
    }
}
int wmain (int argc, TCHAR ** argv) {
    DMS_HANDLE          nVLHandle=-1;
    DMS_RC              rc;
    DMS_RES_NO          OwnStationNo=37;
    DMS_RES_NO          StationNo=5;
    DMS_CONN_HANDLE     ConnHandle;
    DMS_REC_VARLIST_DATA  *lpRecVar;
    DMS_INT16           Index;
    DMS_INT16           OwnCallBackId=1;
    char                szAscStation[20];
    char                szAscVarName[20];
    BOOL                fUnicodeError=FALSE;
```

```
    DMS_OBJ_PATH Path;
    DMS_VAR_TYPE Dtype;  /* DigiTyp */
    DMS_WORD32 Access;
    char Temp[3500];
    DMS_REC_DATA RecData;
       /* Sessionstart */
    if (argc!=3) {
       printf (“Calling Convention: dmsadr <iOwnStationNo> <VarName>”);
       return(0);
    }
    wprintf (L"%s %s %s\n",argv[0],argv[1],argv[2]);
    WideCharToMultiByte(CP_ACP ,0,argv[1],-1,
                                (LPSTR)szAscStation,10,NULL,&fUnicodeError);
    sscanf(szAscStation,"%d",&OwnStationNo);
    WideCharToMultiByte(CP_ACP ,0,argv[2],-1,
                                (LPSTR)szAscVarName,20,NULL,&fUnicodeError);
     /* init with standard GWY */
    if (DMSAPI_Init(OwnStationNo,DMS_OS_GWY,1,TRUE)!=E_DMSAPI_OK)
{
       printf (“Error in DMSAPI_Init : %x\n”);
       goto _LBL_FNC_XIT;
       }
     /* register CallBack - Function */
rc=DMSAPI_RegisterCltCB(OwnCallBackId,OwnDMSAPICallback);
    if (rc) {
       printf(“Fehler beim Register Proc \n”);
```

```
    goto _LBL_FNC_XIT;
  }
  /* check, if there is a valid config */
  WaitForConfig(OwnStationNo);
  /* look for variable in configuration */
  rc=DMSAPI_GetVarInfoByName(OwnStationNo,szAscVarName,
    &StationNo,&Path,&Dtype,&Access);
  if (rc){
      printf("Variable not found in configuration \n");
    goto _LBL_FNC_XIT;
  }
  else printf ("%s : Station %d Path %d - %d Type %d Access %d\n",
            szAscVarName,(int) StationNo,(int) Path.ObjNo,(int) Path.CmpNo,
            (int)Dtype,(int)Access);
  /* Connecting to Station */
if ((rc=DMSAPI_ConnectByNo(OwnStationNo,StationNo,
    &ConnHandle,DMSAPI_STD_ASYNC))!=E_DMSAPI_OK) {
    printf("Fehler beim Connect %08lx to Station %d\n",rc,StationNo);
      goto _LBL_FNC_XIT;
  }
  while (!StationConnect) {
    Sleep(100);
  printf ("trying to connect to Station %d ..\n",StationNo);
    if (kbhit()) goto _LBL_FNC_XIT;
  }
  printf ("Station connected\n");
```

```
/* create VarList */
   if ((rc=DMSAPI_VLCreate
(ConnHandle,DMSAPI_VL_SINGLE_READ,&nVLHandle))!=E_DMSAPI_OK)
{
      printf("Error in VLCreate %lx\n",rc);
      goto _LBL_FNC_XIT;
   }
   /* build VarListe*/
   if ((rc=DMSAPI_VLAddReadVarByName (nVLHandle,szAscVarName,
            &lpRecVar,&Index))!=E_DMSAPI_OK) {
            printf("Error in AddVar : %lx\n",rc);
            goto _LBL_FNC_XIT;
   }
   for (;;) {
       /* Async Read Loop */
      if
((rc=DMSAPI_VLRead(nVLHandle,OwnCallBackId,DMSAPI_STD_ASYNC))!=
E_DMSAPI_OK) {
         printf("Error in VLRead %lx\n",rc);
      }
         /* Antwort auswerten */
      while (!ReadFlag) {
         Sleep(10);
         if (kbhit()) goto _LBL_FNC_XIT;
      }
      ReadFlag=0;
      lpRecVar=(DMS_REC_VARLIST_DATA *) Temp;
```

```
        if
((rc=DMSAPI_VLRead(nVLHandle,0,DMSAPI_SYNCHRON,1000,3500,lpRecV
ar))
            !=E_DMSAPI_OK) {
            printf("Fehler beim VLRead sync %lx\n",rc);
            goto _LBL_FNC_XIT;
        }
        RecData.SrvType=DMS_REC_VARLIST_TYPE;
        RecData.SrvBuff.lpVarList=lpRecVar;
        DMSAPI_DumpRecData(&RecData);
    }
_LBL_FNC_XIT:
    /* Disconnect */
    if ((rc=DMSAPI_VLDelete(nVLHandle))!=E_DMSAPI_OK)
        printf(“Error in  VLDelete %lx\n”,rc);
    DMSAPI_Disconnect(ConnHandle);
while (StationConnect) {
        Sleep(100);
        if (kbhit()) break;
    }
        /* DMS_Ende */
    DMSAPI_Exit(OwnStationNo);
    return(0);
}
```

B.2.2 Cyclical read “acycle.c”

```
/*
```

```
*/

#if 0
FILENAME   acycle.c

HISTORY
      1   deu create
HISTORY_END
#endif

/*
DMSAPI-demo showing the use of the ReadCyclic call

      –   Init of DMSAPI
      –   Register of a Callback-Function
      –   Connect to a Station
      –   Create a VariableList
      –   In a loop the Variable given as argument will be read cyclic

*/

#include <windows.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include "dmstyp.h"
#include "dmsapi.h"
#include "dmserr.h"
int StationConnect=0;
int ReadFlag=0;

/*-----------------------------------------------------------------
      If there is no valid config in c:\digimat\gwy\resxxx
      this function waits for config from Control Builder F
      you can change the ProjectDir by SetProjectDir
      before calling DMSAPI_Init
-----------------------------------------------------------------*/

void WaitForConfig(DMS_RES_NO OwnResNo) {

   DMS_CHAR              Resname[10];
   DMS_UINT32            NoOfRes;
   DMS_NAME_RESOURCE_DATA ResInfo;
   if (DMSAPI_GetFirstResourceInfo(OwnResNo,&NoOfRes,10,Resname,
         &ResInfo)) {

      printf ("No Config: Configure from Control Builder F and press any key to
continue\n");
```

```
        for (;;) {
            Sleep(100);
            if (kbhit()) {
                getch();
                break;
            }
        }
    }
}

DMS_RC OwnDMSAPICallback (DMS_REC_DATA  *lpDmsRec) {

/* Callback-function called by DMSAPI
Attention: this function is called in the context of a communication thread
which has a higher priority than the main thread
you have to protect your data and code!
*/
                    DMS_REC_VARLIST_DATA *lpVarList;
                    int i;
    switch (lpDmsRec->SrvType) {
        case DMS_REC_CONN_TYPE:
            if (!lpDmsRec->DmsRc)
                StationConnect=1;
            else
                StationConnect=0;
            break;
```

```
        case DMS_REC_VARLIST_TYPE:
                ReadFlag=1;
                        DMSAPI_DumpRecData(lpDmsRec);
                                lpVarList = lpDmsRec-
>SrvBuff.lpVarList;
                                for (i = 0; i < lpVarList->MaxVarNo;
i++)
{
                                        if (lpVarList-
>lpVar[i].VarStatus!= DMS_VAR_DELETED)
{
if (lpVarList->lpVar[i].VarRc)
{
                                                /*
DMSAPI reports an error in Read Operation */
                                                } else
if (lpVarList->lpVar[i].VarStatus!= DMS_VAR_NOT_VALID)
{
/* Read was successful:
Datatype in lpVarList->lpVar[i].VarType,
Value in    lpVarList->lpVar[i].VarValue
                                                */
                                                }
                                        }
                                }
           break;
        default:
```

```
            printf (“unknown case\n”);
        }

    return(0);
}

int wmain (int argc, TCHAR ** argv) {

    DMS_HANDLE          nVLHandle=-1;
    DMS_RC              rc;
    DMS_RES_NO          OwnStationNo=37;
    DMS_RES_NO          StationNo=5;
    DMS_CONN_HANDLE     ConnHandle;
    DMS_REC_VARLIST_DATA    *lpRecVar;
    DMS_INT16           Index;
    DMS_INT16           OwnCallBackId=1;
    DMS_INT16           i;
    char                szAscStation[20];
    char                szAscVarName[20];
    BOOL                fUnicodeError=FALSE;
    DMS_OBJ_PATH        Path;
    DMS_VAR_TYPE        Dtype;  /* DigiTyp */
    DMS_WORD32          Access;

      /* session start */
```

```
    if (argc<3) {
        printf ("Calling Convention: dmsacyc <OwnStationNo> <VarName>
<VarName> .....");
        return(0);
    }
    wprintf (L"%s %s %s\n",argv[0],argv[1],argv[2]);

    WideCharToMultiByte(CP_ACP ,0,argv[1],-1,
                                (LPSTR)szAscStation,10,NULL,&fUnicodeError);
    sscanf(szAscStation,"%d",&OwnStationNo);

    if (DMSAPI_Init(OwnStationNo,DMS_OS_GWY,1,TRUE)!=E_DMSAPI_OK)
        goto _LBL_FNC_XIT;

        /* register CallBack - Funktion  */

    rc=DMSAPI_RegisterCltCB(OwnCallBackId,OwnDMSAPICallback);
    if (rc) {
        printf("Error in Register Proc \n");
        goto _LBL_FNC_XIT;
    }
    WaitForConfig(OwnStationNo);
/* Connecting to Station */

    WideCharToMultiByte(CP_ACP ,0,argv[2],-1,
                                (LPSTR)szAscVarName,20,NULL,&fUnicodeError);
```

```
    rc=DMSAPI_GetVarInfoByName(OwnStationNo,szAscVarName,
        &StationNo,&Path,&Dtype,&Access);
    if (rc) {
      printf(“not found %s \n”,szAscVarName);
      goto _LBL_FNC_XIT;
    }
    if ((rc=DMSAPI_ConnectByNo(OwnStationNo,StationNo,
      &ConnHandle,DMSAPI_STD_ASYNC))!=E_DMSAPI_OK) {
      printf(“Error in Connect %08lx\n”,rc);
        goto _LBL_FNC_XIT;
    }

    while (!StationConnect) {
      Sleep(100);
      if (kbhit()) goto _LBL_FNC_XIT;
    }
    printf (“Station connected\n”);
/* create VariablenList */
if ((rc=DMSAPI_VLCreate
(ConnHandle,DMSAPI_VL_CYCLE_READ,&nVLHandle))!=E_DMSAPI_OK) {
      printf(“Error in VLCreate %lx\n”,rc);
      goto _LBL_FNC_XIT;
    }

    /* build VariableList */
```

```
    for (i=2;i<argc;i++) {
        WideCharToMultiByte(CP_ACP ,0,argv[i],-1,
                                    (LPSTR)szAscVarName,20,NULL,&fUnicodeError);
        rc=DMSAPI_GetVarInfoByName(OwnStationNo,szAscVarName,
            &StationNo,&Path,&Dtype,&Access);
        if (rc) printf("Var not found in config %s \n",szAscVarName);
        else printf ("%s : Station %d Path %d - %d Type %d Access %d\n",
                szAscVarName,(int) StationNo,(int) Path.ObjNo,(int) Path.CmpNo,
                (int)Dtype,(int)Access);
        if ((rc=DMSAPI_VLAddReadVarByName (nVLHandle,szAscVarName,
                &lpRecVar,&Index))!=E_DMSAPI_OK) {
                printf("Error in AddVar :%lx\n",rc);
                goto _LBL_FNC_XIT;
        }
    }

for (;;) {

                /* read cyclic */
        if ((rc=DMSAPI_VLReadCycle(nVLHandle,1000,OwnCallBackId,
            DMSAPI_STD_ASYNC))!=E_DMSAPI_OK) {
                printf("Error in VLRead %lx\n",rc);
                goto _LBL_FNC_XIT;
        }
```

```
        printf (“Readcycle\n”);
/* check response     */

        while (!ReadFlag) {
           Sleep(1);
           if (kbhit()) goto _LBL_FNC_XIT;
        }

        if ((rc=DMSAPI_VLStopCycle(nVLHandle))!=E_DMSAPI_OK) {
           printf(“Fehler beim VLStop %lx\n”,rc);
              goto _LBL_FNC_XIT;
        }
        ReadFlag=0;

    }

_LBL_FNC_XIT:

        /* Disconnect */

    Sleep(1000);
    if ((rc=DMSAPI_VLDelete(nVLHandle))!=E_DMSAPI_OK)
        printf(“Error in VLDelete %lx\n”,rc);
   DMSAPI_Disconnect(ConnHandle);
    while (StationConnect) {
        Sleep(100);
```

```
        if (kbhit()) break;
    }
        /* the end */

    DMSAPI_Exit(OwnStationNo);
    return(0);
}
```

B.2.3 One-time write “awrite.c”

```
/*
*/

#if 0
FILENAME        awrite.c
#endif
#if 0
                HISTORY
                1 deu create
                HISTORY_END
#endif

/*
    Demo program for  DMSAPI-communication (Windows):
```

```
        – Calling convention : dmsawrt <OwnStationNo>
        – Init of DMSAPI
        – Register of a Callback-Function
        – look for the first float variable in the config.
        – Connect to the Station with this variable
        – In a loop the Variable will be written
*/
#include <windows.h>
#include “cgen.h”
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include “digityp.hg”
#include “dmstyp.h”
#include “dmsapi.h”
#include “dmserr.h”
                int         StationConnect = 0;
                int         WriteFlag = 0;
                int         ListNo = 0;
                int         RespNo = 0;
                int       success=0;
                int       failed=0;
```

```
/*-----------------------------------------------------------------
      DMSAPI-Callback
-----------------------------------------------------------------*/

                DMS_RC        OwnDMSAPICallback(DMS_REC_DATA *
lpDmsRec)
{
/* Callback-function called by DMSAPI
Attention: this function is called in the context of a communication thread
which has a higher priority than the main thread
you have to protect your data and code!
*/
                DMS_REC_VARLIST_DATA *lpVarList;
                DMS_INT16 i;

                switch        (lpDmsRec->SrvType)
                {
                        case          DMS_REC_CONN_TYPE:
/* DMSAPI calls Callback every time a station connects or disconnects */
                if            (!lpDmsRec->DmsRc)
                 StationConnect = 1;
                else
                {
                        StationConnect = 0;
                        }
```

```
                        break;
                case DMS_REC_VARLIST_TYPE:
                /* case value for a received variable value or a write conf. */
                        lpVarList = lpDmsRec->SrvBuff.lpVarList;
                        for (i = 0; i < lpVarList->MaxVarNo; i++)
{
if (lpVarList->lpVar[i].VarStatus != DMS_VAR_DELETED)
{
if (lpVarList->lpVar[i].VarRc)
{
/* error occured writing the value ! */
failed++;
                                                                        }
else
{
/* write was successful*/

success++;
                                                                        }
                                                                }
                                                        }
                        WriteFlag = 1;
                        ListNo++;
                        RespNo++;
                        break;
                default:
```

```
                    printf("unexpected Case\n");
                }

                return (0);
}

/*---------------------------------------------------------------
    Main-Programm
---------------------------------------------------------------*/

int
wmain(int argc, TCHAR ** argv)
{
                DMS_HANDLE      nVLHandle = -1;
                DMS_RC          rc;
                DMS_RES_NO      OwnStationNo = 123;
                DMS_RES_NO      StationNo = 5;
                int             i, TempStationNo;
                DMS_CONN_HANDLE ConnHandle = -1;
                DMS_REC_VARLIST_DATA *lpRecVar;
                DMS_INT16       Index;
                DMS_INT16       OwnCallBackId = 1;
                DMS_VALUE       DmsValue;
                DMS_FLOAT32     AddConst;
                DMS_UINT32      NoOfVar;
                DMS_NAME_VAR_DATA VarInfo;
```

```
                DMS_CHAR        Name[50];
                DMS_CHAR        szAscStation[50];
                BOOL            fUnicodeError = FALSE;
                DMS_INT16       j, ActVarNo, NoAnswer = 0;
                DWORD           dwOldTicks = GetTickCount();
/*-----------------------------------------------------------------
  If station no. is handed over, it will be converted
-----------------------------------------------------------------*/

                if (argc <= 1)
                {
                        printf("Calling Convention: dmsawrt
<iOwnStationNo>");
                        return (0);
                }
                if (argc > 1)
                {
                        WideCharToMultiByte(CP_ACP, 0, argv[1], -1,
                                        (LPSTR) szAscStation, 10, NULL,
&fUnicodeError);
                        sscanf(szAscStation, "%d", &TempStationNo);
                        OwnStationNo = (DMS_RES_NO) TempStationNo;
                }
/* ----------------------------------------------------------------
            DMSAPI-Init
-----------------------------------------------------------------*/
```

```
                /* init with standard GWY */
                if ((rc = DMSAPI_Init(OwnStationNo, DMS_OS_GWY, 1, TRUE))
!= E_DMSAPI_OK)
                {
                        printf(“Error DMSAPI_Init %08lx \n”, rc);
                        goto _LBL_FNC_XIT;
                }
/* ---------------------------------------------------------------
        Setting CallBack - Function
---------------------------------------------------------------*/

                rc = DMSAPI_RegisterCltCB(OwnCallBackId,
OwnDMSAPICallback);
                if (rc)
                {
                        printf(“Error DMSAPI_Register Proc %08lx \n”, rc);
                        goto _LBL_FNC_XIT;
                }

/* ---------------------------------------------------------------
        Read first variable of datatype Float
---------------------------------------------------------------*/

                rc = DMSAPI_GetFirstVarInfo(OwnStationNo, &NoOfVar, 50,
Name, &VarInfo);
```

```
if (rc)
{
        printf("No Config for GWY-Id %d : Configure from
Control Builder F and press any key to continue\n",
            OwnStationNo);

        for (;;)
        {
            Sleep(100);
            if (kbhit())
            {
                getch();
                break;
            }
        }

        rc = DMSAPI_GetFirstVarInfo(OwnStationNo,
&NoOfVar, 50, Name, &VarInfo);
        if (rc)
        {
    printf("Error DMSAPI_GetFirstVarInfo %08lx \n", rc);
            goto _LBL_FNC_XIT;
        }
}
for (;;)
{
```

```
                        if (VarInfo.VarType == DIGI_FLOAT32)
                                break;
rc = DMSAPI_GetNextVarInfo(OwnStationNo, 50, Name, &VarInfo);
                        if (rc)
                        {
                                printf("Error
DMSAPI_GetNextVarInfo %08lx \n", rc);
                                goto _LBL_FNC_XIT;
                        }
                }

/* ----------------------------------------------------------------
        Connect to  corresponding station
----------------------------------------------------------------*/

                StationConnect = 0;
                if ((rc = DMSAPI_ConnectByNo(OwnStationNo, VarInfo.ResNo,
&ConnHandle,

DMSAPI_STD_ASYNC))!= E_DMSAPI_OK)
                {
                        printf("Error DMSAPI_ConnectByNo %08lx\n");
                        goto _LBL_FNC_XIT;
                }
                while (!StationConnect)
                {
```

```
                    Sleep(100);
                    if (kbhit())
                                goto _LBL_FNC_XIT;
          }
/* ---------------------------------------------------------------
          Creating VariableList
---------------------------------------------------------------*/

          if ((rc = DMSAPI_VLCreate(ConnHandle,
DMSAPI_VL_SINGLE_WRITE, &nVLHandle))!= E_DMSAPI_OK)
          {
                    printf("Fehler beim VLCreate %lx\n", rc);
                    goto _LBL_FNC_XIT;
          }

/* ---------------------------------------------------------------
          Adding Variable to List
          we are filling the List with 280 variables (always the same)
---------------------------------------------------------------*/
          DmsValue.Float32 = (DMS_FLOAT32) 0.0;
          AddConst = (DMS_FLOAT32) 1.0;
          for (ActVarNo = 0; ActVarNo < 280; ActVarNo++)
          {
if ((rc = DMSAPI_VLAddWriteVarByName(nVLHandle, Name, DIGI_FLOAT32,
               &DmsValue, &lpRecVar, &Index))!= E_DMSAPI_OK)
{
```

```
                                        printf("Error
DMSAPI_VLAddWriteVarByName:%lx\n", rc);
                                        break;


                        }
                }
                ActVarNo--;
                printf("Anzahl der Var %d\n", ActVarNo);

/* ---------------------------------------------------------------
                Loop: writes  1.Var from 0.0 to 1000.0, then from
                1000.0 to 0.0
---------------------------------------------------------------*/
                for (;;)
                {
                        if (DmsValue.Float32 == (DMS_FLOAT32) 0.0)
                                AddConst = (DMS_FLOAT32) 1.0;
                        else
                        if (DmsValue.Float32 == (DMS_FLOAT32) 1000.0)
                                AddConst = (DMS_FLOAT32) - 1.0;
/* ---------------------------------------------------------------
            Write VariableList
---------------------------------------------------------------*/
rc = DMSAPI_VLWrite(nVLHandle, OwnCallBackId, DMSAPI_STD_ASYNC);
                        if (rc)
                        {
```

```
                                        printf(“Error in VLWrite %08lx\n”, rc);
                                        if ((rc =
DMSAPI_VLClear(nVLHandle)) != E_DMSAPI_OK)
                                        {
                                                printf(“Error VLClear
%lx\n”, rc);
                                                goto _LBL_FNC_XIT;
                                        }
                                }

/* ---------------------------------------------------------------
        Wait for Answer
-----------------------------------------------------------------*/
                                else
                                {
                                        i = 0;
                                        while (!WriteFlag)
                                        {
                                                i++;
                                                if (kbhit())
                                        goto _LBL_FNC_XIT;
                                                if (i > 10000)
                                                {

NoAnswer++;
if ((rc = DMSAPI_VLDelete(nVLHandle)) != E_DMSAPI_OK)
```

```
printf(“Error VLDelete %lx\n”, rc);
if ((rc = DMSAPI_VLCreate(ConnHandle,

DMSAPI_VL_SINGLE_WRITE, &nVLHandle))!= E_DMSAPI_OK)
                                                                {

printf(“Fehler beim VLCreate %lx\n”, rc);
                                goto _LBL_FNC_XIT;
                                                                }

/* ----------------------------------------------------------------
            Adding Variable to List
----------------------------------------------------------------*/


DmsValue.Float32 = (DMS_FLOAT32) 0.0;
                                AddConst = (DMS_FLOAT32) 1.0;

                                for (j = 0; j < ActVarNo; j++)
                                                                {
if ((rc = DMSAPI_VLAddWriteVarByName(nVLHandle, Name, DIGI_FLOAT32,

&DmsValue, &lpRecVar, &Index)) != E_DMSAPI_OK)
{

printf(“Error DMSAPI_VLAddWriteVarByName:%lx\n”, rc);
```

```
                goto _LBL_FNC_XIT;

                                                                    }
                                                                    }
                            break;
                                                        }
                                                }
                            }
                            printf("Received WriteRequests: %d LostWriteNo %d
write failed %d VarsPerSec %d\r",
                              RespNo, NoAnswer,failed, (RespNo * ActVarNo *
1000) / (GetTickCount() - dwOldTicks));
                            RespNo = 0;
                            dwOldTicks = GetTickCount();
                            WriteFlag = 0;


/* ---------------------------------------------------------------
        Change Value for next Write
-----------------------------------------------------------------*/

                            DmsValue.Float32 += AddConst;

                            for (j = 0; j < ActVarNo; j++)
                            {
```

```
                                                rc =
DMSAPI_VLChangeValue(nVLHandle, j,

DIGI_FLOAT32, &DmsValue, &lpRecVar);
                                                if (rc)
                                                {
                                                        printf(“Error
DMSAPI_VLChangeValue Index% d %08lx\n”, j, rc);
                                                        goto _LBL_FNC_XIT;
                                                }
                                        }
                        }


_LBL_FNC_XIT:

/* ---------------------------------------------------------------
        Deleting VariableList
---------------------------------------------------------------*/

                if ((rc = DMSAPI_VLDelete(nVLHandle)) != E_DMSAPI_OK)
                        printf(“Error VLDelete %lx\n”, rc);
/* ---------------------------------------------------------------
        Disconnecting Station
---------------------------------------------------------------*/
                Sleep(2000);
```

```
                if (ConnHandle != -1)
                        DMSAPI_Disconnect(ConnHandle);

                while (StationConnect)
                {
                        Sleep(100);
                        if (kbhit())
                                break;
                }
/* ---------------------------------------------------------------
        DmsApi-Exit
---------------------------------------------------------------*/
                DMSAPI_Exit(OwnStationNo);
                printf(“Exit done\n”);
                return (0);
}
```

B.3 Alarm services “aalarm.c”

```
/*
*/
#if 0
FILENAME   aalarm.c
HISTORY
      1   deu create
HISTORY_END
#endif
```

```
/*
DMSAPI-Demo showing the use of the message function calls
Init  DMSAPI
      –  Register a Callback-Function
      –  Connect a Station
      –  call to GetAlarmSummary
      –  receive the messages
      –  AutoAcknowledge of all non acknowledged messages
*/
#include “cgen.h”
#include <windows.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include “dmstyp.h”
#include “dmsapi.h”
#include “dmserr.h”
#define ESCAPE goto _LBL_FNC_XIT;
   DMS_INT16 OwnCallBackId=1;
/*---------------------------------------------------------------
      If there is no valid config in c:\digimat\gwy\resxxx
      this function waits for config from Control Builder F
```

```
        you can change the ProjectDir by SetProjectDir
        before calling DMSAPI_Init
-------------------------------------------------------------------*/
void WaitForConfig(DMS_RES_NO OwnResNo) {

    DMS_CHAR              Resname[10];
    DMS_UINT32            NoOfRes;
    DMS_NAME_RESOURCE_DATA ResInfo;

    if (DMSAPI_GetFirstResourceInfo(OwnResNo,&NoOfRes,10,Resname,
          &ResInfo)) {

        printf (“No Config: Configure from Control Builder F and press any key to
continue\n”);
        for (;;) {
          Sleep(100);
          if (kbhit()) {
            getch();
            break;
          }
        }
    }
}
DMS_RC OwnDMSAPICallback (DMS_REC_DATA  *lpDmsRec) {

/* Callback-function called by DMSAPI
```

```
Attention: this function is called in the context of a communication thread
which has a higher priority than the main thread
you have to protect your data and code  !
*/
DMS_RC rcloc;
   DMS_REC_ACKALARM AckAL[DMSAPI_MAX_ALARM_IN_ACKAL];
   DMS_REC_ALARMLIST_DATA *lpRecAL;
   int i;
   DMS_INT16 Ackno=0;
   DMS_HANDLE DmsHandle;

   DMSAPI_DumpRecData(lpDmsRec);
   switch (lpDmsRec->SrvType) {
      case DMS_REC_CONN_TYPE:
         if (!lpDmsRec->DmsRc) {
            if (lpDmsRec->SrvBuff.lpConn->ulConnFlag != DMS_RES_CLIENT) {
/* every time a station connects, a getAlarmsummary should be called */
               rcloc=DMSAPI_GetAlarmSummary(lpDmsRec->ConnHandle,
                  OwnCallBackId,DMSAPI_STD_ASYNC);
               if (rcloc) {
                  printf("GetAlarmSummary %08lx\n",rcloc);
               }
            }
         }
         break;
case DMS_REC_ALARMLIST_TYPE:
```

```
                    /* this case is for the messages */
            lpRecAL=lpDmsRec->SrvBuff.lpAlarmList;
            for (i=0;i<lpRecAL->ActAlarmNo;i++) {
              if (lpRecAL-
>lpAlarm[i].CurrAlarmStatus==DMS_ALARM_INACT_INACTNACKED ||
                  lpRecAL-
>lpAlarm[i].CurrAlarmStatus==DMS_ALARM_ACT_ACTNACKED ||
                  lpRecAL-
>lpAlarm[i].CurrAlarmStatus==DMS_ALARM_INACT_ACTNACKED) {
                  AckAL[Ackno].ObjectId=lpRecAL->lpAlarm[i].ObjectId;
                  AckAL[Ackno].AlarmIndex=lpRecAL->lpAlarm[i].AlarmIndex;
                  AckAL[Ackno].AlarmStatus=lpRecAL->lpAlarm[i].CurrAlarmStatus;
                  AckAL[Ackno].rc=E_DMSAPI_OK;
                  Ackno++;
              }
            }
            if (Ackno) {
                    /* there are some messages to acknowledge */
              rcloc=DMSAPI_AckAlarmByList(lpDmsRec-
>ConnHandle,&DmsHandle,
                    OwnCallBackId,Ackno,AckAL,DMSAPI_STD_ASYNC);
              if (rcloc) printf (“Error in Alarmacknowledge %08lx\n”,rcloc);
              else printf (“Acknowledge %d\n”,Ackno);
            }
            break;
          default:
break;
```

```
    }

    return(0);
}

int main (int argc, char * * argv) {

    DMS_RC rc;
    DMS_RES_NO OwnStationNo=88;
    DMS_RES_NO StationNo=5;
    DMS_CONN_HANDLE ConnHandle;

     if (argc!=3) {
        printf ("Calling Convention: dmsala <OwnStationNo> <MsrStationNo>");
        return(0);
    }
                    sscanf(argv[1],"%d",&OwnStationNo);
                    sscanf(argv[2],"%d",&StationNo);

    /* DMSAPI-Init */
if
((rc=DMSAPI_Init(OwnStationNo,DMS_OS_GWY,1,TRUE))!=E_DMSAPI_OK)
{
        printf("Error DMSAPI_Init %08lx \n",rc);
        goto _LBL_FNC_XIT;
    }
```

```
rc=DMSAPI_RegisterCltCB(OwnCallBackId,OwnDMSAPICallback);
   if (rc) {
      printf(“Error DMSAPI_Register Proc %08lx \n”,rc);
      goto _LBL_FNC_XIT;
   }

   WaitForConfig(OwnStationNo);

   if ((rc=DMSAPI_ConnectByNo(OwnStationNo,StationNo,
      &ConnHandle,DMSAPI_STD_ASYNC))!=E_DMSAPI_OK) {
         printf(“Error DMSAPI_ConnectByNo %08lx\n”);
         goto _LBL_FNC_XIT;
   }
   for (;;) {
/* nothing to do here, it all happens in the callback function */
      Sleep(100);
      if (kbhit()) {
         getch();
         goto _LBL_FNC_XIT;
      }
   }

_LBL_FNC_XIT:

      /* Disconnect */
   rc=DMSAPI_RegisterCltCB(OwnCallBackId,NULL);
```

```
    DMSAPI_Disconnect(ConnHandle);

    Sleep(1000);

      /* DMS_Ende */
    DMSAPI_Exit(OwnStationNo);
    printf("Exit erreicht\n");
    return(0);
}
```

B.4 Name services “name”

```
/*
*/
#if 0

Projekt:                    Freelance
FILENAME          name.c $
COMMENT
          DMSAPI - Demo showing the use of the name management
COMMENT_END

VERSION           $Revision:   1.0  $ (0)

HISTORY
HISTORY_END
```

```
/* $Log: name.c_v  $

*/
#endif

/*
/*
    DMSAPI - Demo showing the use of the name management

        – Init of DMSAPI
        – register a Callback-Function
        – Output of all informations the name management can give
        – conversion routines for the variable names
*/

#include <windows.h>
#include “dmstyp.h”
#include “dmsapi.h”
#include “dmserr.h”
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <malloc.h>
#include <conio.h>
```

```
#include <time.h>

int StationConnect=0;
/*---------------------------------------------------------------
      DMS-API-Callback
---------------------------------------------------------------*/

DMS_RC OwnDMSAPICallback (DMS_REC_DATA  *lpDmsRec) {

/* Callback-function called by DMSAPI
Attention: this function is called in the context of a communication thread
which has a higher priority than the main thread
you have to protect your data and code!
*/
DMSAPI_DumpRecData(lpDmsRec);
   switch (lpDmsRec->SrvType) {
      case DMS_REC_CONN_TYPE:
         if (!lpDmsRec->DmsRc)
            StationConnect=1;
         else
            StationConnect=0;
         break;
      default:
         printf (“unknown case\n”);
      }
```

```
   return(0);
}

/*-----------------------------------------------------------------
      If there is no valid config in c:\digimat\gwy\resxxx
      this function waits for config from Control Builder F
      you can change the ProjectDir by SetProjectDir
      before calling DMSAPI_Init
-----------------------------------------------------------------*/

void WaitForConfig(DMS_RES_NO OwnResNo) {

   DMS_CHAR              Resname[10];
   DMS_UINT32            NoOfRes;
   DMS_NAME_RESOURCE_DATA ResInfo;

   if (DMSAPI_GetFirstResourceInfo(OwnResNo,&NoOfRes,10,Resname,
         &ResInfo)) {
printf (“No Config: Configure from Control Builder F and press any key to
continue\n”);
      for (;;) {
         Sleep(100);
         if (kbhit()) {
            getch();
            break;
         }
```

```
        }

    }

}
/*---------------------------------------------------------------
      main-Programm
---------------------------------------------------------------*/
int main (int argc, char * * argv) {

    DMS_RC              rc;
    DMS_RES_NO          StationNo=1;
    DMS_RES_NO          OwnStationNo=19;
    DMS_VAR_TYPE        Dtype;  /* DigiTyp */
    DMS_WORD32          Access;
    DMS_UINT32          j,i,NoOfVar;
    DMS_UINT32          NoOfCmp;
    DMS_CHAR            Name[50] ;
    DMS_CHAR            CmpName[50];
    DMS_NAME_RESOURCE_DATA StatInfo;
    DMS_NAME_VAR_DATA     VarInfo;
    DMS_NAME_TAG_DATA     TagInfo;
    DMS_NAME_OBJ_DATA     ObjInfo;
    DMS_INT16           OwnCallBackId=1;
    DMS_OBJ_PATH        Path;
/*---------------------------------------------------------------
```

```
        check para
-----------------------------------------------------------------*/

   if (argc>1)  {
                sscanf(argv[1],“%d”,&OwnStationNo);
   }
                else
                {
                printf(“Calling Convention: dmsnam <iOwnStationNo> \n\n”);
                        return (0);
                }
/*-----------------------------------------------------------------
        start a session
-----------------------------------------------------------------*/
rc=DMSAPI_Init(OwnStationNo,DMS_OS_MSR,1,TRUE);
   if (rc) {
      printf(“Error in Init %x\n”,rc );
      goto _LBL_FNC_XIT;
   }
/*-----------------------------------------------------------------
      Set CallBack function
-----------------------------------------------------------------*/
rc=DMSAPI_RegisterCltCB(OwnCallBackId,OwnDMSAPICallback);
   if (rc) {
      printf(“Error in Register %x \n”,rc);
      goto _LBL_FNC_XIT;
```

```
    }
WaitForConfig(OwnStationNo);
/* ---------------------------------------------------------------
            get the info about the configured stations
-----------------------------------------------------------------*/
printf("Stations:\n");

rc=DMSAPI_GetFirstResourceInfo(OwnStationNo,&NoOfVar,50,Name,&StatInfo
);
   if (!rc) {
      for (i=0;i<NoOfVar-1;i++) {
         printf(“%s\n”,Name);
         rc=DMSAPI_GetNextResourceInfo(OwnStationNo,50,Name,&StatInfo);
         if (rc) printf (“Error %08lx\n”,rc);
      }
      printf(“%s\n”,Name);
   }
   else printf (“Error %08lx\n”,rc);
/* ---------------------------------------------------------------
            get the info about the configured variables
-----------------------------------------------------------------*/
   printf(“Variables:\n”);
   rc=DMSAPI_GetFirstVarInfo(OwnStationNo,&NoOfVar,50,Name,&VarInfo);
   if (!rc) {
      for (i=0;i<NoOfVar-1;i++) {
```

```
            printf(“%s/DigVal 3 0.22\n”,Name,(int)VarInfo.OPath.ObjNo,
               (int)VarInfo.OPath.CmpNo);
            rc=DMSAPI_GetNextVarInfo(OwnStationNo,50,Name,&VarInfo);
            if (rc) printf (“Error %08lx\n”,rc);
         }
         printf(“%s\n”,Name,(int)VarInfo.OPath.ObjNo,
               (int)VarInfo.OPath.CmpNo);
      }
      else printf (“Error %08lx\n”,rc);
/* ----------------------------------------------------------------
get the info about the configured tags
for every found Tag : get info about the tag (all pins and parameter)
----------------------------------------------------------------*/
      printf(“Tags:\n”);
      rc=DMSAPI_GetFirstTagInfo(OwnStationNo,&NoOfVar,50,Name,&TagInfo);
      if (!rc) {
         for (i=0;i<NoOfVar;i++) {
            printf (“Tag %s : %d\n”,Name,(int)TagInfo.ObjClass);
            rc=DMSAPI_GetFirstCmpOfObjClass(OwnStationNo,TagInfo.ObjClass,
                  &NoOfCmp,50,CmpName,&ObjInfo);
            if (!rc) {
               for (j=0;j<NoOfCmp-1;j++) {
                  if (!rc) {
                     printf(“%s/%s\n”,Name,CmpName);
                  }
                  else printf (“- Error”);
```

```
rc=DMSAPI_GetNextCmpOfObjClass(OwnStationNo,TagInfo.ObjClass,
                    50,CmpName,&ObjInfo);
            }
            if (!rc) {
/*                  if (ObjInfo.nRWFlag)*/
                    printf(“%s/%s\n”,Name,CmpName);
            }
            else printf (“- Error \n”);
         }
         else
            printf(“No components %08lx\n”,rc);

         if (i<NoOfVar-1) {
            rc=DMSAPI_GetNextTagInfo(OwnStationNo,50,Name,&TagInfo);
            if (rc) printf (“Error %08lx\n”,rc);
         }
      }
   }
   else printf (“Error %08lx\n”,rc);

/* ----------------------------------------------------------------
now showing the conversion routine DMSAPI_GetVarInfoByName
----------------------------------------------------------------*/
     printf(“now showing the conversion routine DMSAPI_GetVarInfoByName\n”);
   for (;;) {
```

```
      printf(“give a name of a Variable (quit with ‘q’)\n”);
      scanf(“%s”,Name);
       if (Name[0]=='q' && strlen(Name)==1) goto _LBL_FNC_XIT;
       else {

rc=DMSAPI_GetVarInfoByName(OwnStationNo,Name,&StationNo,&Path,&Dty
pe,
              &Access);
          if (rc) printf(“variable not found \n”);
          else printf (“%s : Station %d Path %d - %d Type %d Access %d\n”,
                Name,(int) StationNo,(int) Path.ObjNo,(int) Path.CmpNo,
                (int)Dtype,(int)Access);
       }
    }

_LBL_FNC_XIT:

/*-------------------------------------------------------------
       the end
-------------------------------------------------------------*/
    DMSAPI_Exit(OwnStationNo);

     return (0);
}
```

B.5 Setting the time “settime.c”

```
/*
*/
#if 0
FILENAME   settime.c


HISTORY
      1   deu create
HISTORY_END
#endif
/*
    DMSAPI-demo showing the use of the     DMSAPI_SetRemoteTimeByString
call


   -Calling convention: dmstime dd.mm.yyyy hh:mm:ss
   - Init
   - DMSAPI_SetRemoteTimeByString
   - Exit DMS
*/
#include <windows.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
```

```
#include <time.h>
#include “dmstyp.h”
#include “dmsapi.h”
#include “dmserr.h”
int main (int argc, char ** argv) {
DMS_RES_NO          OwnStationNo=187;
   char Time[100];
   DMS_RC rc;
if (argc<3) {
      printf (“Calling Convention: dmstime dd.mm.yyyy hh:mm:ss
         on a system with german local settings\n”);
      printf (“Calling Convention: dmstime mm/dd/yyyy hh:mm:ss
         on a system with english local settings\n”);
      return(0);
   }
if (DMSAPI_Init(OwnStationNo,DMS_OS_GWY,1,TRUE)!=E_DMSAPI_OK)
      goto _LBL_FNC_XIT;
sprintf(Time,“%s %s”, argv[1],argv[2]);
/*    the DMSAPI will only accept strings which have the correct syntax
corresponding to the
     settings in your Registry (--> control panel -> international )
*/
if ((rc =DMSAPI_SetSystemTimeByString(Time))!= E_DMSAPI_OK)
      printf (“Error in SetSystemTime %x : \n”,rc);


_LBL_FNC_XIT:
```

```
DMSAPI_Exit(OwnStationNo);
   return(0);
}
```

B.6 Toggle primary/secondary redundancy “toggle.c”

```
/*
*/
#if 0
FILENAME   toggle
HISTORY
      1   deu create
HISTORY_END
#endif
/*
    DMSAPI-demo showing the use of the ReadCyclic call
      –   Init of DMSAPI
      –   Register of a Callback-Function
      –   Connect to a Station
      –   issue a RestartResource with Toggle_cmd
      –   sleep given time
      –   toggle again in loop
*/
#include <windows.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
#include <conio.h>
#include <time.h>
#include “dmstyp.h”
#include “dmsapi.h”
#include “dmserr.h”
int StationConnect=0;
int PIRecv=0;
#define MAX_DMS_VL 10
/*----------------------------------------------------------------
      If there is no valid config in c:\digimat\gwy\resxxx
      this function waits for config from Control Builder F
      you can change the ProjectDir by SetProjectDir
      before calling DMSAPI_Init
----------------------------------------------------------------*/
void WaitForConfig(DMS_RES_NO OwnResNo) {

   DMS_CHAR              Resname[10];
   DMS_UINT32            NoOfRes;
   DMS_NAME_RESOURCE_DATA ResInfo;
if (DMSAPI_GetFirstResourceInfo(OwnResNo,&NoOfRes,10,Resname,
         &ResInfo)) {
printf (“No Config: Configure from Control Builder F and press any key to
continue\n”);
      for (;;) {
         Sleep(100);
```

```
            if (kbhit()) {
                getch();
                break;
            }
        }
    }
}

DMS_RC OwnDMSAPICallback (DMS_REC_DATA  *lpDmsRec) {

/* Callback-function called by DMSAPI
Attention: this function is called in the context of a communication thread
which has a higher priority than the main thread
you have to protect your data and code!
*/
printf("OwnCallback\n");
    DMSAPI_DumpRecData(lpDmsRec);
    switch (lpDmsRec->SrvType) {
        case DMS_REC_CONN_TYPE:
            if (!lpDmsRec->DmsRc)
                StationConnect=1;
            else
                StationConnect=0;
            break;
        default:
            printf ("Was ist das?\n");
```

```
            break;
        }
return(0);
}
int wmain (int argc, TCHAR ** argv) {
    DMS_RC              rc;
    DMS_RES_NO          OwnStationNo=37;
    DMS_RES_NO          StationNo=5;
    DMS_CONN_HANDLE     ConnHandle;
    DMS_INT16           OwnCallBackId=1;
    DMS_INT16           i;
    char                szAscStation[20];
    BOOL                fUnicodeError=FALSE;
    int                 TimeCount,PISnd=0;
/* session start*/
if (argc<4) {
        printf (
    “Calling Convention: dmsatog <OwnStationNo> <StationNo> <ToggleTime
Sec>”);
        return(0);
    }
    wprintf (L“%s %s %s\n”,argv[0],argv[1],argv[2]);

    WideCharToMultiByte(CP_ACP ,0,argv[1],-1,
                                (LPSTR)szAscStation,10,NULL,&fUnicodeError);
    sscanf(szAscStation,“%d”,&OwnStationNo);
```

```
    WideCharToMultiByte(CP_ACP ,0,argv[2],-1,
                                (LPSTR)szAscStation,10,NULL,&fUnicodeError);
    sscanf(szAscStation,"%d",&StationNo);

    WideCharToMultiByte(CP_ACP ,0,argv[3],-1,
                                (LPSTR)szAscStation,10,NULL,&fUnicodeError);
    sscanf(szAscStation,"%d",&TimeCount);

    if (DMSAPI_Init(OwnStationNo,DMS_OS_GWY,1,TRUE)!=E_DMSAPI_OK)
       goto _LBL_FNC_XIT;

       /* register CallBack - Function  */

    rc=DMSAPI_RegisterCltCB(OwnCallBackId,OwnDMSAPICallback);
    if (rc) {
       printf("Error in Register Proc \n");
       goto _LBL_FNC_XIT;
    }
/* check if config available */

    WaitForConfig(OwnStationNo);
       /* Connecting to Station */
if ((rc=DMSAPI_ConnectByNo(OwnStationNo,StationNo,
       &ConnHandle,DMSAPI_STD_ASYNC))!=E_DMSAPI_OK) {
       printf("Error in Connect %08lx\n",rc);
```

```
            goto _LBL_FNC_XIT;
        }

        while (!StationConnect) {
            Sleep(100);
            if (kbhit()) goto _LBL_FNC_XIT;
        }
        printf (“Station connected\n”);
for (;;) {

                    printf (“issue toggle command to station %d\n”,StationNo);
                    /* issue a Red-Toggle on the process station */

DMSAPI_RestartResource(ConnHandle,DMS_RESTART_TOGGLE);
            for (i=0;i<TimeCount;i++) {
Sleep(1000);
            if (kbhit()) goto _LBL_FNC_XIT;
            }

        }
_LBL_FNC_XIT:

            /* Disconnect */

        Sleep(1000);
```

```
    DMSAPI_Disconnect(ConnHandle);

    while (StationConnect) {
      Sleep(100);

      if (kbhit()) break;
    }
      /* the end */

    DMSAPI_Exit(OwnStationNo);
    return(0);
}
```

Appendix C DMSAPI files

C.1 dmstyp.h

```
/*
COMMENT
*****************************************************************
*

                    DMS-API

        Digimatik Message Specification Application Interface
        Communication Protocol for Freelance Process Level
            Type and other Definitions

*****************************************************************
*
COMMENT_END

FILENAME            $Workfile:   dmstyp.h  $

VERSION       $Revision:   1.7.1.1  $  (0)

HISTORY
HISTORY_END
```

```
/*****************************************************************************
********/
 $Log: dmstyp.h_v  $
*****************************************************************************
*******/
#if __cplusplus
extern "C" {
#endif

#ifndef _DMSAPI_TYP_H
#define _DMSAPI_TYP_H


/*---------------------------------------------------------------------
                FREELANCE Basic-Datatypes
---------------------------------------------------------------------*/

typedef unsigned short DMS_WORD16;

typedef unsigned long DMS_WORD32;

typedef float         DMS_FLOAT32;
typedef signed char    DMS_INT8;
```

```
typedef short          DMS_INT16;

typedef long           DMS_INT32;

typedef DMS_WORD16     DMS_UINT16;

typedef DMS_WORD32     DMS_UINT32;

typedef char           DMS_CHAR;

typedef unsigned char  DMS_BYTE;

typedef unsigned char  DMS_BOOLEAN;

typedef DMS_UINT16     DMS_OBJNO;

typedef DMS_UINT16     DMS_CMPNO;

typedef DMS_INT32  DMS_TIME;

typedef struct
{
 DMS_WORD32  dwMSecondsHigh;   /* ms since 1.1.1970 0.00 Uhr GMT (high)
*/
 DMS_WORD32  dwMSecondsLow;    /* ms since 1.1.1970 0.00 Uhr GMT (low)
*/
```

```
} DMS_DT;
/*---------------------------------------------------------------------
                FREELANCE String-Datatypes
---------------------------------------------------------------------*/

#define DMS_STRING_ALGN 2       /* 2 Bytes Allignement at the end of each
String  */
/* **** DMS_STRING8 - Typ ***** */

#define DMS_STRING8_LENGTH 8

typedef struct {
  DMS_WORD16    wMaxStringLen;          /* max Len of String */
  DMS_CHAR      Content[DMS_STRING8_LENGTH+DMS_STRING_ALGN];
/* Content */
} DMS_STRING8;

typedef DMS_STRING8  * LPDMS_STRING8;

/* **** STRING16 - Typ ***** */
#define DMS_STRING16_LENGTH 16
typedef struct {
  DMS_WORD16    wMaxStringLen;   /* max Len of String */
  DMS_CHAR      Content[DMS_STRING16_LENGTH+DMS_STRING_ALGN];
/* Content     */
} DMS_STRING16;
```

```
typedef DMS_STRING16  * LPDMS_STRING16;
/* **** STRING32 - Typ ***** */

#define DMS_STRING32_LENGTH 32

typedef struct {
 DMS_WORD16    wMaxStringLen;   /* max Len of String */
 DMS_CHAR      Content[DMS_STRING32_LENGTH+DMS_STRING_ALGN];
/* Content     */
} DMS_STRING32;

typedef DMS_STRING32  * LPDMS_STRING32;
/* **** DMS_STRING64 - Typ ***** */
#define DMS_STRING64_LENGTH 64
typedef struct {
 DMS_WORD16    wMaxStringLen;   /* max Len of String */
 DMS_CHAR      Content[DMS_STRING64_LENGTH+DMS_STRING_ALGN];
/* content       */
} DMS_STRING64;

typedef DMS_STRING64  * DMS_LPSTRING64;

/* **** DMS_STRING128 - Typ ***** */

#define DMS_STRING128_LENGTH 128
```

```
typedef struct {
  DMS_WORD16    wMaxStringLen;    /* max Len of String */
  DMS_CHAR
Content[DMS_STRING128_LENGTH+DMS_STRING_ALGN];  /* Content    */
} DMS_STRING128;

typedef DMS_STRING128  * LPDMS_STRING128;
/* **** DMS_STRING256 - Typ ***** */

#define DMS_STRING256_LENGTH 256

typedef struct {
  DMS_WORD16    wMaxStringLen;    /* max Len of String */
  DMS_CHAR
Content[DMS_STRING256_LENGTH+DMS_STRING_ALGN]; /* content */
} DMS_STRING256;

typedef DMS_STRING256  * LPDMS_STRING256;
/*---------------------------------------------------------------------
                FREELANCE Datatype-Union
---------------------------------------------------------------------*/

typedef union {
   DMS_WORD16     Word16;
   DMS_WORD32     Word32;
```

```
    DMS_FLOAT32    Float32;
    DMS_INT8       Int8;
    DMS_INT16      Int16;
    DMS_INT32      Int32;
    DMS_UINT16     Uint16;
    DMS_UINT32     Uint32;
    DMS_CHAR       Char;
    DMS_BOOLEAN    Boolean;
    DMS_BYTE       Byte;
    DMS_OBJNO      ObjNo;
    DMS_CMPNO      CmpNo;
    DMS_TIME       DmsTime;
    DMS_DT         DmsDT;
    DMS_STRING8    String8;
    DMS_STRING16   String16;
    DMS_STRING32   String32;
    DMS_STRING64   String64;
    DMS_STRING128  String128;
    DMS_STRING256  String256;
} DMS_VALUE;
/*---------------------------------------------------------------------
                maximale StringLaengen
---------------------------------------------------------------------*/

#define DMS_MAX_RESNAME_LEN     10
#define DMS_MAX_VARNAME_LEN     40
```

```
#define DMS_MAX_TAGNAME_LEN     15
#define DMS_MAX_COMPNAME_LEN    15

/*---------------------------------------------------------------------
            DMS Resource number and type
---------------------------------------------------------------------*/
typedef unsigned short DMS_RES_NO;
typedef unsigned short DMS_RES_TYPE;

/*---------------------------------------------------------------------
            DMS variable types
---------------------------------------------------------------------*/

typedef unsigned short DMS_VAR_TYPE;
#define  DMS_VAR_TYPE_BOOLEAN          0x01
#define  DMS_VAR_TYPE_CHAR             0x02
#define  DMS_VAR_TYPE_BYTE             0x03
#define  DMS_VAR_TYPE_INT8             0x04
#define  DMS_VAR_TYPE_WORD16           0x05
#define  DMS_VAR_TYPE_UINT16           0x06
#define  DMS_VAR_TYPE_INT16            0x07
#define  DMS_VAR_TYPE_WORD32           0x08
#define  DMS_VAR_TYPE_UINT32           0x09
#define  DMS_VAR_TYPE_INT32            0x0A
#define  DMS_VAR_TYPE_FLOAT32          0x0B
#define  DMS_VAR_TYPE_TIME             0x0C
```

```
#define  DMS_VAR_TYPE_DMSTIME            0x0D

#define  DMS_VAR_TYPE_STRING8            0x0E    /* Strings          */
#define  DMS_VAR_TYPE_STRING16           0x0F    /* Strings          */
#define  DMS_VAR_TYPE_STRING32           0x10    /* Strings          */
#define  DMS_VAR_TYPE_STRING64           0x11    /* Strings          */
#define  DMS_VAR_TYPE_STRING128          0x12    /* Strings          */
#define  DMS_VAR_TYPE_STRING256          0x13    /* Strings          */
#define  DMS_VAR_TYPE_OBJNO          0x2C
#define  DMS_VAR_TYPE_CMPNO           0x2D

/*-----------------------------------------------------------------------
            DMS var error type
-----------------------------------------------------------------------*/

typedef unsigned long  DMS_VAR_RC;

/*-----------------------------------------------------------------------
            DMS error
-----------------------------------------------------------------------*/
typedef unsigned long  DMS_RC;


/*-----------------------------------------------------------------------
            DMS ConnectionHandle
-----------------------------------------------------------------------*/
```

```
typedef int  DMS_CONN_HANDLE;
#define DMSAPI_HANDLE_MIN_NO 0
#define DMSAPI_HANDLE_MAX_NO 150
#define DMSAPI_NO_HANDLE -1

/*---------------------------------------------------------------------
        DMS Service Handle
---------------------------------------------------------------------*/
typedef short DMS_HANDLE;
/*---------------------------------------------------------------------
        DMS Variable ObjPath
---------------------------------------------------------------------*/
typedef struct {

 DMS_OBJNO    ObjNo;
 DMS_CMPNO    CmpNo;

} DMS_OBJ_PATH;

/*---------------------------------------------------------------------
        DMS - Client Receive Services
---------------------------------------------------------------------*/

#define DMSAPI_SYNCHRON 1
#define DMSAPI_ASYNCHRON 2
#define DMSAPI_WAIT_FOREVER 0xffffffff
```

```
#define DMSAPI_NO_TIMEOUT 0

#define DMSAPI_STD_ASYNC DMSAPI_ASYNCHRON,
DMSAPI_WAIT_FOREVER, 0, NULL


typedef enum  {

   DMS_REC_CONN_TYPE,
   DMS_REC_VARLIST_TYPE,
   DMS_REC_INFO_REPORT_TYPE,
   DMS_REC_ALARMLIST_TYPE,
   DMS_REC_ACKALARMLIST_TYPE,
   DMS_REC_PROGRAM_INVOCATION_TYPE,
   DMS_REC_DOMAIN_TYPE,
   DMS_REC_VERSION_TYPE

}  DMS_REC_SERVICE_TYPE;

/* ---------------------------------------------------------------------
                Connection management
---------------------------------------------------------------------*/


typedef enum  {

   DMS_CONN_OK,              /* o.k.*/
   DMS_CONN_ABORT,           /* no connection */
```

```
    DMS_CONN_INVALID_RES_TYPE,      /* wrong resource type */
    DMS_CONN_INVALID_RES_NO,        /* wrong resource number */
    DMS_CONN_NO_OS,                 /* no operation system */
    DMS_CONN_SECONDARY,             /* remote station is secondary =>
                                       cannot connect */
    DMS_CONN_INVALID_VERSION        /* wrong DMS_Version */

} DMS_CONN_STATUS;

/* ------------------------------------------------------------------
values for nBTRLnk in connect- routines
------------------------------------------------------------------*/

#define DMS_BTR_TCPIP  1   /* Standard BTR using  TCPIP */
#define DMS_BTR_REDLNK 2  /* BTR only for redundant resource */


/* ------------------------------------------------------------------
                values for connection flag
------------------------------------------------------------------*/


#define DMS_RES_PRIMARY     1   /* connection to a primary server */
#define DMS_RES_SECONDARY   2   /* connection to a secondary server */
#define DMS_RES_CLIENT      3   /* connection to a client */
/* ------------------------------------------------------------------
```

```
                values for the cpu board type
-------------------------------------------------------------------------*/

#define  DMS_CPU_UNKNOWN      0    /* ...                */
#define  DMS_CPU_DCP02        1    /* CPU_01, 960CA/CF   */
#define  DMS_CPU_DCP10        2    /* CPU_02, 960Hx      */
#define  DMS_CPU_PC           3    /* PC                 */

/* ------------------------------------------------------------------------
                values for  OS_RES_TYPE
-------------------------------------------------------------------------*/

#define DMS_OS_DIGIVIS      1
#define DMS_OS_DIGITOOL     2
#define DMS_OS_EPROM        3
#define DMS_OS_MSR          4
#define DMS_OS_DDE_GWY      5
#define DMS_OS_P_GWY        6
#define DMS_OS_GWY          7

typedef struct DMS_REC_CONN_DATA {

   DMS_RES_NO        OwnResNo;        /* Own Resource Id */
   DMS_RES_NO        ResNo;           /* remote resource Id */
   DMS_RES_TYPE      ResType;         /* OS Types */
   DMS_CONN_STATUS   ConnStatus;      /* connection state */
```

```
   DMS_UINT32        ulIPAddr;        /* IP-adresse of remote resource */
   DMS_UINT32        ulBoardType;      /* cpu board type */
   DMS_UINT32        ulConnFlag;       /* ConnectionFlag */

} DMS_REC_CONN_DATA;

typedef enum  {
   DMS_RESTART_WARM,
   DMS_RESTART_COLD,
   DMS_RESTART_FATAL,
   DMS_RESTART_TOGGLE
} DMS_RESTART_REASON;

/* ---------------------------------------------------------------------
                Variable mangement
---------------------------------------------------------------------*/
#define DMSAPI_VL_SINGLE_READ  1
#define DMSAPI_VL_CYCLE_READ   2
#define DMSAPI_VL_SINGLE_WRITE 3

#define DMSAPI_NOACCESS  0x00
#define DMSAPI_READONLY  0x01
#define DMSAPI_WRITEONLY 0x02
#define DMSAPI_READWRITE 0x03

typedef enum  {
```

```
    DMS_VAR_NOT_VALID,
    DMS_VAR_NOT_CHANGED,
    DMS_VAR_CHANGED,
    DMS_VAR_DELETED

} DMS_VAR_STATUS;

typedef struct DMS_REC_VAR {

  DMS_VAR_STATUS    VarStatus;
  DMS_VAR_RC        VarRc;
  DMS_OBJ_PATH      ObjPath;
  DMS_CHAR *        VarName;
  DMS_UINT32        ValueSize;      /* Size of ValueBuffer */
  DMS_VAR_TYPE      VarType;
  DMS_VALUE         *VarValue;

} DMS_REC_VAR;

typedef struct DMS_REC_VARLIST_DATA {

    DMS_HANDLE      DmsHandle;
    DMS_INT16       ActVarNo;  /* actual amount of variables */
    DMS_INT16       MaxVarNo;  /* max. amount of variables */
    DMS_INT16       FreeBytes; /* amount of free bytes in the VL */
```

```
    DMS_REC_VAR *   lpVar;

} DMS_REC_VARLIST_DATA;


/* ------------------------------------------------------------------
                Version data
------------------------------------------------------------------*/

typedef struct DMS_VERSION_DATA {

    DMS_CHAR        *ProjName; /* Projectname */
    DMS_WORD16      wMajorVersion;
    DMS_WORD16      wMinorVersion;
    DMS_WORD16      wPatchVersion;

} DMS_VERSION_DATA;

typedef struct DMS_REC_VERSION_DATA {

    DMS_CHAR        *ProjName; /* Projectname */
    DMS_WORD16      wMajorVersion;
    DMS_WORD16      wMinorVersion;
    DMS_WORD16      wPatchVersion;
    DMS_OBJNO       ObjClass;
    DMS_OBJNO       ObjNo;
```

```
} DMS_REC_VERSION_DATA;

/* ------------------------------------------------------------------
                Alarmmanagement
------------------------------------------------------------------*/

typedef DMS_WORD16 DMS_ALARM_TYPE;

typedef enum  {
   DMS_ALARM_PRIO_0,
   DMS_ALARM_PRIO_1,
   DMS_ALARM_PRIO_2,
   DMS_ALARM_PRIO_3,
   DMS_ALARM_PRIO_4,
   DMS_ALARM_PRIO_5,

} DMS_ALARM_PRIO;

typedef enum  {

   DMS_ALARM_INACT_ACTNACKED,      /*
inactive/active_not_acknowledged */
   DMS_ALARM_ACT_ACTNACKED,        /* active/active_not_acknowledged */
   DMS_ALARM_INACT_INACTNACKED,    /* inactive/ not_acknowledged */
   DMS_ALARM_ACT_ACTACKED,         /* active/ acknowledged */
```

```
   DMS_ALARM_NOT_VALID_4,
   DMS_ALARM_NOT_VALID_5,
   DMS_ALARM_INACT_INACTACKED,     /* inactive/inactive_acknowledged
*/
   DMS_ALARM_NOT_VALID_7,
   DMS_ALARM_AP_DELETED            /* message object was deleted */

} DMS_ALARM_STATUS;

typedef enum  {

   DMS_ALARM_GAS,
   DMS_ALARM_LAST_GAS,
   DMS_ALARM_EVENTS,

} DMS_ALARM_LIST_TYPE;
typedef struct DMS_REC_ALARM {
   DMS_DT              TransitionTime;
   DMS_OBJNO           ObjectId;
   DMS_WORD16          AlarmIndex;
   DMS_ALARM_TYPE      AlarmType;
   DMS_OBJNO           ObjectClass;
   DMS_ALARM_STATUS    CurrAlarmStatus;
   DMS_ALARM_STATUS    PrevAlarmStatus;
   DMS_ALARM_PRIO      Priority;
   DMS_BOOLEAN         NotificationLost;
```

```
    DMS_RC              rc;
    DMS_UINT32          ValueSize;
    DMS_VAR_TYPE        AlarmValType;
    DMS_VALUE           *AlarmValue;
} DMS_REC_ALARM;

#define DMSAPI_MAX_ALARM_IN_AL 43

typedef struct DMS_REC_ALARMLIST_DATA {

    DMS_ALARM_LIST_TYPE ListType;
    DMS_INT16           ActAlarmNo;  /* actual amount of messages */
    DMS_REC_ALARM       *lpAlarm;    /* message list  */

} DMS_REC_ALARMLIST_DATA;

typedef struct DMS_REC_ACKALARM {
    DMS_OBJNO           ObjectId;
    DMS_WORD16          AlarmIndex;
    DMS_ALARM_STATUS    AlarmStatus;
    DMS_RC              rc;
} DMS_REC_ACKALARM;

#define DMSAPI_MAX_ALARM_IN_ACKAL 157

typedef struct DMS_REC_ACKALARMLIST_DATA {
```

```
   DMS_HANDLE         DmsHandle;
   DMS_INT16          ActAckAlarmNo;  /* actual amount of acknowledged
messages */
   DMS_REC_ACKALARM    *lpAckAlarm;      /* acknowledged message list  */

} DMS_REC_ACKALARMLIST_DATA;

/* ----------------------------------------------------------------------
            Receivedata Union
----------------------------------------------------------------------*/
typedef union {

   DMS_REC_CONN_DATA          *lpConn;
   DMS_REC_VARLIST_DATA       *lpVarList;
   DMS_REC_ALARMLIST_DATA     *lpAlarmList;
   DMS_REC_ACKALARMLIST_DATA   *lpAckAlarmList;
   DMS_REC_VERSION_DATA       *lpVersion;

} DMS_REC_SERVICE_DATA;

typedef struct DMS_REC_DATA {

   DMS_CONN_HANDLE        ConnHandle;     /* StationsConnHandle */
   DMS_RC                 DmsRc;          /* ErrorCode */
   DMS_UINT32             BuffSize;       /* Size of DataBuffer */
```

```
    DMS_REC_SERVICE_TYPE    SrvType;        /* ServiceTyp */
    DMS_REC_SERVICE_DATA    SrvBuff;        /* Pointer to DmsData */

} DMS_REC_DATA;

typedef DMS_RC (* DMS_REC_DATA_PROC) ( DMS_REC_DATA  *DmsRec);

#define DMSAPI_MAX_CB 10
#define DMSAPI_NO_CALLBACK 0

/* ----------------------------------------------------------------------
                    Server management
----------------------------------------------------------------------*/
typedef enum  {

    DMS_WRITE_SERVICE_TYP,
    DMS_READ_SERVICE_TYP,
    DMS_GETDATA_ADDR_SERVICE_TYP

}  DMS_VAR_SERVICE_TYP;

typedef struct {

  DMS_OBJ_PATH      ObjPath;
  int               VarLen;
  DMS_VAR_TYPE      VarType;
```

```
 DMS_VALUE        *VarValue;
 DMS_VAR_RC       VarRc;

} DMS_VAR_ELEM;

typedef DMS_RC (* DMS_VAR_SERVER_PROC)
  (
   DMS_CONN_HANDLE        ConnHandle,
   DMS_VAR_SERVICE_TYP    VarServiceTyp,
   int                    VarElemNo,
   DMS_VAR_ELEM           *VarElem
   );

typedef enum  {

   DMS_DLINIT_SERVICE_TYP,
   DMS_DLEXIT_SERVICE_TYP,
   DMS_ULINIT_SERVICE_TYP,
   DMS_ULEXIT_SERVICE_TYP,
   DMS_DELDOM_SERVICE_TYP

}  DMS_DOM_SERVICE_TYP;

typedef enum  {

   DMS_DOM_RAM_TYP,
```

```
    DMS_DOM_PRAM_TYP,
    DMS_DOM_FILE_TYP,
    DMS_DOM_PROC_TYP

} DMS_DOMAIN_TYP;

typedef DMS_RC (* DMS_DOM_SERVER_PROC)
  (
    DMS_CONN_HANDLE       ConnHandle,
    DMS_OBJNO             ObjNo,
    DMS_RC                rc,
    DMS_DOMAIN_TYP        DomainType,
    DMS_INT32             *DomainLen,
    DMS_CHAR              *DomainContent,
    DMS_CHAR              **OwnDomainContent
    );

/* ------------------------------------------------------------------
                DMS Name management
------------------------------------------------------------------*/

typedef struct DMS_NAME_RESOURCE_DATA {

    DMS_WORD32        dwIPAddr1;
    DMS_WORD32        dwIPAddr2;
    DMS_RES_NO        ResNo;
```

```
    DMS_RES_TYPE       ResType;
    DMS_UINT16         wTimeOut; /* in Sek */
    DMS_UINT16         wMajorVersionNo;
    DMS_UINT16         wMinorVersionNo;
    DMS_UINT16         wPatchVersionNo;

} DMS_NAME_RESOURCE_DATA;

typedef struct DMS_NAME_VAR_DATA {

    DMS_WORD32         dwAccessRights;
    DMS_VAR_TYPE       VarType;
    DMS_RES_NO         ResNo;
    DMS_OBJ_PATH       OPath;

} DMS_NAME_VAR_DATA;

typedef struct DMS_NAME_TAG_DATA {
    DMS_WORD32     dwAccessRights;
    DMS_RES_NO     ResNo;
    DMS_OBJNO      ObjClass;
    DMS_OBJNO      ObjNo;
    DMS_CMPNO      CmpNo;

} DMS_NAME_TAG_DATA;
typedef struct DMS_NAME_OBJ_DATA {
```

```
    DMS_WORD16      nRWFlag;
    DMS_CMPNO       CmpNo;
    DMS_VAR_TYPE    VarType;
    DMS_WORD16      Reserved;

    /* component name as string  null terminated and 4 byte alignment */


} DMS_NAME_OBJ_DATA;


/* ---------------------------------------------------------------------
                DMS-Utilities
---------------------------------------------------------------------*/

typedef struct DMS_VAR_CODE {

                        char        szDateStringSyntax[5];
                        char        szDecimal[10];
                        char        sz1000Decimal[10];
                        char        szDate[10];
                        char        szTimeSeparation[10];
                        BOOLEAN     fDigiTimeAsLong;

} DMS_VAR_CODE;
```

```
#endif /* _DMSAPI_TYP_H defined */

#if __cplusplus
}
#endif
```

C.2 dmsapi.h

```
#ifdef CGEN
COMMENT
*****************************************************************
*

                    DMS-API

        Digimatik Message Specification ApplicationInterface
        Communication Protocol for Digimatik Process Level
                    Functions
*****************************************************************
*
COMMENT_END

FILENAME            $Workfile:   DMSAPI.H  $
```

```
VERSION             $Revision:   1.13.1.0  $  (0)

HISTORY
HISTORY_END

/* $Log: DMSAPI.H_v  $
*/
#endif
#if __cplusplus
extern "C" {
#endif

#ifndef _DMSAPI_FNC_H
#define _DMSAPI_FNC_H
#ifdef  __DMS_API_INIT_FKT__
#  ifdef WIN32
#    define CGEXPORT _declspec( dllexport )
#  else
#    define CGEXPORT
#  endif
#else
#  ifdef WIN32
#    define CGEXPORT _declspec( dllimport )
#  else
#    define CGEXPORT
```

```
# endif
#endif /* __DMS_API_INIT_FKT__*/

/* ---------------------------------------------------------------------
        Environment and General Management Services
---------------------------------------------------------------------*/

/*  Initialisation of Dms on a Gateway */

CGEXPORT DMS_RC DMSAPI_Init (
      DMS_RES_NO      OwnResNo       /* Own Resource Id */,
      DMS_RES_TYPE    OwnResType     /* Own Resource Typ */,
      DMS_INT16       NoOfSrvConn    /* Number of ServerConnection */,
      DMS_BOOLEAN     bStandardServer /* TRUE or FALSE */);

/*  Shutdown  Dms */

CGEXPORT DMS_RC  DMSAPI_Exit (
      DMS_RES_NO OwnResNo  /* Own Resource No */);

/* StationConnect */

CGEXPORT DMS_RC DMSAPI_ConnectByAddr(
      DMS_RES_NO        OwnResNo       /* Own Resource Id */,
      DMS_INT16         nBTRLnk        /* take DMS_BTR_TCPIP  from
DMSTYP.h */,
```

```
        DMS_UINT32          ulIPAddr1       /* first ipaddress of remote station */,
        DMS_UINT32          ulIPAddr2       /* second ipaddress of remote station */,
        DMS_RES_NO          ResNo           /* resource no */,
        DMS_RES_TYPE        ResType         /* resource type */,
        DMS_UINT16          ulKeepAliveT    /* KeepAliveTimeout */,
        DMS_CONN_HANDLE     *lpConnHandle   /* ConnectionHandle */,
        DMS_INT16           nSyncFlag       /* synchrone flag */,
        DMS_UINT32          ulProcT         /* prozedure timeout */,
        DMS_UINT32          ulRecConnLen    /* size of RecConn */,
        DMS_REC_CONN_DATA   *RecConn        /* Out -> ReceiveStruct of Conn.
*/);

CGEXPORT DMS_RC DMSAPI_ConnectByName(
        DMS_RES_NO          OwnResNo        /* Own Resource No */,
        DMS_CHAR            *ResName        /* name of resource */,
        DMS_CONN_HANDLE     *lpConnHandle   /* ConnectionHandle */,
        DMS_INT16           nSyncFlag       /* synchrone flag */,
        DMS_UINT32          ulProcT         /* prozedure Timeout */,
        DMS_UINT32          ulRecConnLen    /* size of RecConn */,
        DMS_REC_CONN_DATA   *RecConn        /* Out -> ReceiveStruct of Conn.
*/);

CGEXPORT DMS_RC DMSAPI_ConnectByNo(
        DMS_RES_NO          OwnResNo        /* Own Resource No */,
        DMS_RES_NO          ResNo           /* name of resource */,
        DMS_CONN_HANDLE     *lpConnHandle   /* ConnectionHandle */,
```

```
        DMS_INT16          nSyncFlag       /* synchrone flag */,
        DMS_UINT32         ulProcT         /* prozedure Timeout */,
        DMS_UINT32         ulRecConnLen    /* size of RecConn */,
        DMS_REC_CONN_DATA   *RecConn       /* Out -> ReceiveStruct of Conn.
*/);

/* StationDisConnect */

CGEXPORT DMS_RC DMSAPI_Disconnect(
        DMS_CONN_HANDLE     ConnHandle     /* ConnectionHandle */);

/* ConnectionData */

CGEXPORT DMS_RC DMSAPI_GetConnectionData(
        DMS_CONN_HANDLE     ConnHandle     /* ConnectionHandle */,
        DMS_REC_CONN_DATA   *Data          /* Out -> ReceiveStruct of Conn. */);

/* Set RemoteTime */
#ifdef WIN32

CGEXPORT DMS_RC DMSAPI_SetSystemTime (
         SYSTEMTIME    *NTDT /* Win32 date and time format */);
#endif
CGEXPORT DMS_RC DMSAPI_SetSystemTimeByDmsType (
        DMS_DT  *DateTime  /* DMS date and time */);
```

```
CGEXPORT DMS_RC DMSAPI_SetSystemTimeByString (
      DMS_CHAR * lpszDateTime  /* date and time string  */);

CGEXPORT DMS_RC DMSAPI_RestartResource(
   IN  DMS_CONN_HANDLE     ConnHandle      /*  ConnectionHandle */,
   IN  DMS_RESTART_REASON  RestartReason   /*  RestartReason */);

CGEXPORT DMS_RC DMSAPI_RegisterCltCB(
   DMS_INT16          nCBId          /* CallbackId */,
   DMS_REC_DATA_PROC   CallBackProc   /* Callbackfunction */);

/* ---------------------------------------------------------------------
                Variablemangement
---------------------------------------------------------------------*/

CGEXPORT DMS_RC DMSAPI_VLCreate(
      DMS_CONN_HANDLE     ConnHandle      /*  ConnectionHandle */,
      DMS_INT16          nVLService      /*  Service:
                                 DMSAPI_VL_SINGLE_READ
                                 DMSAPI_VL_CYCLE_READ
                                 DMSAPI_VL_SINGLE_WRITE */,
      DMS_HANDLE         *lpDmsHandle    /*  Identifier for Varlist */);
CGEXPORT DMS_RC DMSAPI_VLAddWriteVarByName(
   DMS_HANDLE            DmsHandle       /*  VarListHandle */,
   DMS_CHAR              *lpszVarname    /*  Variable name */,
   DMS_VAR_TYPE          VarType         /*  Variable type */,
```

```
    DMS_VALUE           *lpvVarValue    /*  Variable value */,
    DMS_REC_VARLIST_DATA    **lplpRecVar    /*  Pointer to RecVarStruct */,
    DMS_INT16           *lpnIndex       /*  Index in RecVarStruct */);

CGEXPORT DMS_RC DMSAPI_VLAddWriteVarByAddr(
    DMS_HANDLE          DmsHandle       /*  VarListHandle */,
    DMS_OBJ_PATH        *lpOpath        /*  Objectpath */,
    DMS_VAR_TYPE        VarType         /*  Variable type */,
    DMS_VALUE           *lpvVarValue    /*  Variable value */,
    DMS_REC_VARLIST_DATA    **lplpRecVar    /*  Pointer to RecVarStruct */,
    DMS_INT16           *lpnIndex       /*  Index in RecVarStruct */);

CGEXPORT DMS_RC DMSAPI_VLAddReadVarByName(
    DMS_HANDLE          DmsHandle       /*  VarListHandle */,
    DMS_CHAR            *lpszVarname    /*  Variable name */,
    DMS_REC_VARLIST_DATA    **lplpRecVar    /*  Pointer to RecVarStruct */,
    DMS_INT16           *lpnIndex       /*  Index in RecVarStruct */);
CGEXPORT DMS_RC DMSAPI_VLAddReadVarByAddr(
    DMS_HANDLE          DmsHandle       /*  VarListHandle */,
    DMS_OBJ_PATH        *lpOpath        /*  Objectpath */,
    DMS_VAR_TYPE        VarType         /*  Variable type */,
    DMS_REC_VARLIST_DATA    **lplpRecVar    /*  Pointer to RecVarStruct */,
    DMS_INT16           *lpnIndex       /*  Index in RecVarStruct */);
CGEXPORT DMS_RC DMSAPI_VLChangeValue(
    DMS_HANDLE          DmsHandle       /*  VarListHandle */,
    DMS_INT16           nIndex          /*  Index in RecVarStruct */,
```

```
    DMS_VAR_TYPE          VarType       /*  Variable type */,
    DMS_VALUE             *lpvVarValue  /*  Variable value */,
    DMS_REC_VARLIST_DATA  **lplpRecVar  /*  Pointer to RecVarStruct */);

CGEXPORT DMS_RC DMSAPI_VLDelVar(
    DMS_HANDLE            DmsHandle     /*  VarListHandle */,
    DMS_INT16             nIndex        /*  Index in RecVarStruct */,
    DMS_REC_VARLIST_DATA  **lplpRecVar  /*  Pointer to RecVarStruct */);

CGEXPORT DMS_RC DMSAPI_VLClear(
    DMS_HANDLE            DmsHandle     /*  VarListHandle */);

CGEXPORT DMS_RC DMSAPI_VLRead(
    DMS_HANDLE            DmsHandle     /*  VarListHandle */,
    DMS_INT16             nCBId         /*  CallbackId */,
    DMS_INT16             nSyncFlag     /*  synchron flag */,
    DMS_UINT32            ulProcT       /*  prozedure timeout */,
    DMS_UINT32            ulRecVarLen   /*  size of RecVarStruct */,
   DMS_REC_VARLIST_DATA   *lpRecVar     /*  Out -> Pointer to RecVarStruct
*/);

CGEXPORT DMS_RC DMSAPI_VLReadCycle(
    DMS_HANDLE            DmsHandle     /*  VarListHandle */,
    DMS_UINT32            ulCycleTime   /*  cycletime in ms */,
    DMS_INT16             nCBId         /*  CallbackId */,
    DMS_INT16             nSyncFlag     /*  synchrone flag */,
```

```
   DMS_UINT32            ulProcT        /*  prozedur timeout */,
   DMS_UINT32            ulRecVarLen    /*  Size of RecVarStruct */,
   DMS_REC_VARLIST_DATA   *lpRecVar     /*  Out-> Pointer to RecVarStruct
*/);

CGEXPORT DMS_RC DMSAPI_VLStopCycle(
   DMS_HANDLE            DmsHandle      /*  VarListHandle */);

CGEXPORT DMS_RC DMSAPI_VLWrite(
   DMS_HANDLE            DmsHandle      /*  VarListHandle */,
   DMS_INT16             nCBId          /*  CallbackId */,
   DMS_INT16             nSyncFlag      /*  synchrone flag */,
   DMS_UINT32            ulProcT        /*  prozedure timeout */,
   DMS_UINT32            ulRecVarLen    /*  Size of RecVarStruct */,
   DMS_REC_VARLIST_DATA   *lpRecVar     /*  Out -> Pointer to RecVarStruct
*/);

CGEXPORT DMS_RC DMSAPI_VLDelete(
      DMS_HANDLE         DmsHandle      /*  VarListHandle */);



/* ----------------------------------------------------------------
                Alarmmangement
----------------------------------------------------------------*/
CGEXPORT DMS_RC DMSAPI_GetAlarmSummary(
   DMS_CONN_HANDLE       ConnHandle     /*  ConnectionHandle */,
```

```
    DMS_INT16           nCBId         /*  CallbackId */,
    DMS_INT16           nSyncFlag     /*  synchrone flag */,
    DMS_UINT32          ulProcT       /*  prozedure timeout */,
    DMS_UINT32          ulRecVarLen   /*  size of AlarmRec */,
    DMS_REC_ALARMLIST_DATA  *lpAlarmRec   /*  Out -> Pointer to
AlarmListStruct */);

CGEXPORT DMS_RC DMSAPI_AckAlarmByList(
    DMS_CONN_HANDLE         ConnHandle    /*  ConnectionHandle */,
    DMS_HANDLE              *lpDmsHandle  /*  Identifier for Acklist */,
    DMS_INT16               nCBId         /*  CallbackId */,
    DMS_INT16               ActAlarmNo    /*  actual amount of messages */,
    DMS_REC_ACKALARM        *lpAlarmAck   /*  Pointer to AlarmAckStruct
*/,
    DMS_INT16               nSyncFlag     /*  synchrone flag */,
    DMS_UINT32              ulProcT       /*  prozedure timeout */,
    DMS_UINT32              ulRecVarLen   /*  size of AckAlarmRec */,
    DMS_REC_ACKALARMLIST_DATA   *lpAckAlarmRec  /*  Out -> Pointer to
AlarmAckListStruct */);


/* ----------------------------------------------------------------------
                DMS Name management
----------------------------------------------------------------------*/

CGEXPORT DMS_RC DMSAPI_LockOV (
```

```
    DMS_RES_NO OwnResNo   /* */);

CGEXPORT DMS_RC DMSAPI_UnlockOV (DMS_RES_NO OwnResNo   /*
GWY Resource Id*/);

CGEXPORT DMS_RC DMSAPI_SetProjectDir(DMS_CHAR * szProjectDir /*
path to new Directory */);

CGEXPORT DMS_RC DMSAPI_ChangeProject(
    DMS_RES_NO          OwnResNo      /* GWY Resource Id */,
    DMS_CHAR            * ProjName /* new project name*/);

CGEXPORT DMS_RC DMSAPI_GetProjectInfo(
    DMS_RES_NO          OwnResNo      /* GWY Resource Id */,
    DMS_VERSION_DATA    * VersionData  /* OUT -> pointer to VersionData */);

CGEXPORT DMS_RC DMSAPI_GetVarInfoByName(
    DMS_RES_NO          OwnResNo      /* GWY Resource Id */,
    DMS_CHAR            * lpVarName     /* variable name */,
    DMS_RES_NO          * lpResNo      /* Out -> remote resource Id */,
    DMS_OBJ_PATH        * lpPath       /* Out -> object path */,
    DMS_VAR_TYPE        * lpVarType     /* Out -> variable type */,
    DMS_WORD32              * lpAccessRights /* Out -> Access Rights */);
CGEXPORT DMS_RC DMSAPI_GetVarnameByOPath (
    DMS_RES_NO          OwnResNo    /* GWY Resource Id */,
    DMS_RES_NO          ResNo       /* remote resource Id */,
```

```
    DMS_OBJ_PATH       *lpPath     /* Out -> ObjectPath */,
    DMS_UINT32         VarNameLen  /* max. size of Varname */,
    DMS_CHAR           *lpVarName  /* Out -> variable name */);
CGEXPORT DMS_RC DMSAPI_GetTagByAddr (
    DMS_RES_NO          OwnResNo       /* GWY Resource Id */,
    DMS_RES_NO          ResNo          /* remote resource Id */,
    DMS_OBJNO           ObjNo          /* ObjectPath */,
    DMS_UINT32          TagNameLen     /* max. Size of tagname */,
    DMS_CHAR            *lpTagName     /* Out -> tagname */,
    DMS_NAME_TAG_DATA      *lpTagInfo     /* Out -> tagInfo */);

CGEXPORT DMS_RC DMSAPI_GetFirstResourceInfo(
    DMS_RES_NO          OwnResNo       /* GWY Resource Id */,
    DMS_UINT32          *lpulNoOfRess  /* Out -> amount of resources */,
    DMS_UINT32          ResNameLen     /* max. size of resname */,
    DMS_CHAR            *lpResName     /* Out -> name of resource */,
    DMS_NAME_RESOURCE_DATA *lpResInfo     /* Out -> ResInfo */);

CGEXPORT DMS_RC DMSAPI_GetNextResourceInfo(
    DMS_RES_NO          OwnResNo        /* GWY Resource Id */,
    DMS_UINT32          ResNameLen      /* max. size of resname */,
    DMS_CHAR            *lpResName      /* Out -> name of resource */,
    DMS_NAME_RESOURCE_DATA *lpResInfo       /* Out -> ResInfo */);
CGEXPORT DMS_RC DMSAPI_GetFirstVarInfo(
    DMS_RES_NO       OwnResNo        /* GWY Resource Id */,
    DMS_UINT32       *lpulNoOfVar    /* amount of variables in config */,
```

```
    DMS_UINT32          VarNameLen          /* max. size of variable name */,
    DMS_CHAR            *lpVarName          /* Out -> variable name */,
    DMS_NAME_VAR_DATA   *lpVarInfo          /* Out -> variable info */);

CGEXPORT DMS_RC DMSAPI_GetNextVarInfo(
    DMS_RES_NO          OwnResNo            /* GWY Resource Id */,
    DMS_UINT32          VarNameLen          /* max. size of variable name */,
    DMS_CHAR            *lpVarName          /* Out -> variable name */,
    DMS_NAME_VAR_DATA   *lpVarInfo          /* Out -> variable info */);

CGEXPORT DMS_RC DMSAPI_GetFirstTagInfo(
    DMS_RES_NO          OwnResNo            /* GWY Resource Id */,
    DMS_UINT32          *lpulNoOfTag        /* amount of tags in config. */,
    DMS_UINT32          TagNameLen          /* max. size of tagname */,
    DMS_CHAR            *lpTagName          /* Out -> tagname */,
    DMS_NAME_TAG_DATA   *lpTagInfo          /* Out -> taginfo */);

CGEXPORT DMS_RC DMSAPI_GetNextTagInfo(
    DMS_RES_NO          OwnResNo            /* GWY Resource Id */,
    DMS_UINT32          TagNameLen          /* max. size of tagname */,
    DMS_CHAR            *lpTagName          /* Out -> tagname */,
    DMS_NAME_TAG_DATA   *lpTagInfo          /* Out -> taginfo */);

CGEXPORT DMS_RC DMSAPI_GetFirstCmpOfObjClass(
    DMS_RES_NO          OwnResNo            /* GWY Resource Id */,
    DMS_OBJNO           ObjClass            /* Object class */,
```

```
    DMS_UINT32          *lpulNoOfCmp        /* amount of components */,
    DMS_UINT32          CmpNameLen          /* max. size of component name */,
    DMS_CHAR            *lpCmpName          /* component name */,
    DMS_NAME_OBJ_DATA   *lpObjInfo          /* ObjectInfo */);

CGEXPORT DMS_RC DMSAPI_GetNextCmpOfObjClass(
    DMS_RES_NO          OwnResNo            /* GWY Resource Id */,
    DMS_OBJNO           ObjClass            /* Object class */,
    DMS_UINT32          CmpNameLen          /* max. size of component name */,
    DMS_CHAR            *lpCmpName          /* component name */,
    DMS_NAME_OBJ_DATA   *lpObjInfo          /* ObjectInfo */);
/* ----------------------------------------------------------------------
                DMS-ServerManagement (Not yet implemented ! )
----------------------------------------------------------------------*/

CGEXPORT DMS_RC DMSAPI_ActivateServer(
    DMS_RES_NO      OwnResNo        /* GWY Resource Id */);

CGEXPORT DMS_RC DMSAPI_DeactivateServer(
    DMS_RES_NO      OwnResNo        /* GWY Resource Id */);

CGEXPORT DMS_RC DMSAPI_OpenVarServer(
    DMS_RES_NO          OwnResNo                /* GWY Resource Id */,
    DMS_VAR_SERVER_PROC DMSReadVarServerProc    /* */,
    DMS_VAR_SERVER_PROC DMSWriteVarServerProc   /* */,
    int                 MaxServer               /* */);
```

```
CGEXPORT DMS_RC DMSAPI_CreateInfoReport(
   DMS_CONN_HANDLE ConnHandle      /*  ConnectionHandle */,
   DMS_INT16       OwnIRId         /*  InforeportId for Client */,
   DMS_HANDLE      *lpDmsHandle    /*  Identifier for Informationreport */);

CGEXPORT DMS_RC DMSAPI_DeleteInfoReport(
   DMS_HANDLE      DmsHandle       /*  Identifier for Informationreport */);

CGEXPORT DMS_RC DMSAPI_GetInfoReportBuffer(
   DMS_HANDLE      DmsHandle       /*  Identifier for Informationreport */,
   DMS_UINT32      ulProcT         /*  prozedure timeout */,
   DMS_UINT32      ulRecVarLen     /*  size of RecVar */,
   DMS_CHAR        **lplpRecVar    /*  Out -> Pointer to InfoReport */);

CGEXPORT DMS_RC DMSAPI_SendInfoReportBuffer(
   DMS_HANDLE      DmsHandle       /*  Identifier for Informationreport */,
   DMS_UINT32      ulProcT         /*  prozedure timeout */,
   DMS_UINT32      ulRecVarLen     /*  size of RecVar */,
   DMS_CHAR        *lpRecVar       /*  Out -> Pointer to InfoReport */);

/* ------------------------------------------------------------------
                DMS-Utilities
------------------------------------------------------------------*/
```

```
CGEXPORT void DMSAPI_DumpRecData(DMS_REC_DATA * DmsRecData /*
*/) ;

CGEXPORT int DMSAPI_GetVarLen(DMS_VAR_TYPE VarType /* variable type
*/);

CGEXPORT DMS_RC DMSAPI_SetVarCode(DMS_VAR_CODE * VarCode);

CGEXPORT DMS_RC DMSAPI_GetStringByValue(
   DMS_UINT32      ulStrLen       /*  size of String */,
   DMS_CHAR        *lpszString    /*  Out ->  String */,
   DMS_VAR_TYPE    VarType        /* variable type */,
   DMS_VALUE       *lpvVarValue   /*  Out -> variable value */);

CGEXPORT DMS_RC DMSAPI_GetValueByString(
   DMS_UINT32      ulValLen       /*  size of VarValue */,
   DMS_VALUE       *lpvVarValue   /*  Out -> Value */,
   DMS_VAR_TYPE    VarType        /*  variable type */,
   DMS_CHAR        *lpszString    /*  Out -> String */);

/* ---------------------------------------------------------------------
(Not yet implemented ! )
---------------------------------------------------------------------*/

CGEXPORT DMS_RC DMSAPI_GetErrStrByErr(
   DMS_UINT32      ulStrLen       /*  size of String */,
```

```
   DMS_CHAR       *lpszString     /*  Out -> String */,
   DMS_RC         Rc              /*  ErrorCode */);

#endif /* _DMSAPI_TYP_H defined */

#if __cplusplus
}
#endif
```

C.3 dmserr.h

```
/*
COMMENT
*****************************************************************
*


                    DMS-API
        Digimatik Message Specification ApplicationInterface
        Communication Protocol for Digimatik Process Level

                    ErrorCodes

*****************************************************************
*
COMMENT_END
```

```
FILENAME            $Workfile:   DMSERR.H  $

VERSION         $Revision:   1.5.1.0  $  (0)

HISTORY
HISTORY_END

$Log: DMSERR.H_v  $

**********************************************************************
*******/

#include "errbase.hg"

#define E_DMSAPI_OK                 0x00

#define E_DMSAPI_NOT_INIT           (E_DMSAPI_BASE + 0x01)
#define E_DMSAPI_INVALID_CONF       (E_DMSAPI_BASE + 0x02)
#define E_DMSAPI_INVALID_ARG        (E_DMSAPI_BASE + 0x03)
#define E_DMSAPI_SMALL_RCV_BUFF     (E_DMSAPI_BASE + 0x04)
#define E_DMSAPI_EMPTY_CONF         (E_DMSAPI_BASE + 0x05)
#define E_DMSAPI_INTERNAL_ERROR     (E_DMSAPI_BASE + 0x06)
#define E_DMSAPI_ACCESS_ERROR       (E_DMSAPI_BASE + 0x07)
#define E_DMSAPI_NO_CONF            (E_DMSAPI_BASE + 0x08)
#define E_DMSAPI_INVALID_DMS_HANDLE   (E_DMSAPI_BASE + 0x09)
#define E_DMSAPI_INVALID_CONN_HANDLE  (E_DMSAPI_BASE + 0x0A)
```

```
#define E_DMSAPI_NO_RESOURCE            (E_DMSAPI_BASE + 0x0B)
#define E_DMSAPI_VARLIST_IN_USE         (E_DMSAPI_BASE + 0x0C)
#define E_DMSAPI_NO_CALLBACK            (E_DMSAPI_BASE + 0x0D)
#define E_DMSAPI_DUPLICATE_CALLBACK     (E_DMSAPI_BASE + 0x0E)
#define E_DMSAPI_INVALID_INDEX          (E_DMSAPI_BASE + 0x0F)
#define E_DMSAPI_INVALID_VARTYPE        (E_DMSAPI_BASE + 0x10)
#define E_DMSAPI_INVALID_VARMODE        (E_DMSAPI_BASE + 0x11)
#define E_DMSAPI_NO_CONNECTION          (E_DMSAPI_BASE + 0x12)
#define E_DMSAPI_ALREADY_INIT           (E_DMSAPI_BASE + 0x13)
#define E_DMSAPI_MAX_APPLICATION        (E_DMSAPI_BASE + 0x14)
#define E_DMSAPI_MAX_CONNECTION         (E_DMSAPI_BASE + 0x15)
#define E_DMSAPI_TIMEOUT                (E_DMSAPI_BASE + 0x16)
#define E_DMSAPI_INVALID_DIR            (E_DMSAPI_BASE + 0x17)
```

Index

—

www.abb.com/freelance
www.abb.com/controlsystems

—

We reserve the right to make technical changes to the products or modify the contents of this document without prior notice. With regard to purchase orders, the agreed particulars shall prevail. ABB does not assume any responsibility for any errors or incomplete information in this document.

3BDD012508-111 A