PROCESS AUTOMATION

Freelance 2019

Referenz-Handbuch
DMS / API

PROCESS AUTOMATION

Freelance 2019

Referenz-Handbuch
DMS / API

Dokumentennummer: 3BDD012508-111
Revision: A
Veröffentlichung: Februar 2019

Hinweis

Dieses Dokument enthält Informationen über ABB Produkte und kann außerdem Beschreibungen von Normen bzw. Verweise auf Normen enthalten, die allgemein für ABB Produkte relevant sind. Das Vorliegen solcher Beschreibungen von Normen bzw. von Verweisen auf Normen bedeutet nicht, dass alle in diesem Dokument genannten ABB Produkte sämtliche Merkmale der jeweils beschriebenen oder genannten Norm unterstützen. Informationen zu den einzelnen Merkmalen, die ein bestimmtes ABB Produkt unterstützt, finden Sie in der jeweiligen Produktspezifikation des betreffenden ABB Produkts.

ABB verfügt u. U. über Patente oder anhängige Patentanmeldungen zum Schutz der Rechte des geistigen Eigentums an den in diesem Dokument genannten ABB Produkten.

Die in diesem Dokument enthaltenen Informationen können ohne Vorankündigung geändert werden und sollten nicht als eine Verpflichtung von ABB gesehen werden. ABB übernimmt keine Verantwortung für irgendwelche Fehler, die in diesem Dokument auftreten können.

Die in diesem Dokument beschriebenen oder genannten Produkte sind so realisiert, dass sie zuschaltbar sind und Informationen und Daten über ein sicheres Netzwerk übermitteln. Es liegt in der alleinigen Verantwortung des System-/Produkteigentümers, eine sichere Verbindung zwischen dem Produkt und dem Systemnetzwerk und/oder anderen ggf. angebundenen Netzwerken bereitzustellen und dauerhaft aufrechtzuerhalten.

Die System-/Produkteigentümer sind verpflichtet, angemessene Vorkehrungen (u. a. Installation von Firewalls, Anwendung von Maßnahmen zur Authentifizierung, Verschlüsselung von Daten, Installation von Virenschutzprogrammen) zu treffen, um das System sowie die zugehörigen Produkte und Netzwerke vor Sicherheitslücken, unberechtigtem Zugriff, Störungen, Eingriffen, Verlusten und/oder Diebstahl von Daten oder Informationen zu schützen.

ABB überprüft das ordnungsgemäße Funktionieren der freigegebenen Produkte und Aktualisierungen. Dennoch sind letztendlich die System-/Produkteigentümer dafür verantwortlich, dass Systemaktualisierungen (u. a. Code-Änderungen, Änderungen an Konfigurationsdateien, Updates oder Patches der Software von Drittanbietern, Austausch von Hardware) mit den eingeführten Sicherheitsmaßnahmen kompatibel sind. Die System-/Produkteigentümer müssen verifizieren, dass das System und die zugehörigen Produkte in der Umgebung, in der sie implementiert sind, erwartungsgemäß funktionieren.

ABB haftet nicht für unmittelbare, mittelbare, konkrete, beiläufig entstandene oder Folgeschäden irgendeiner Art, die durch die Verwendung dieses Dokuments entstanden sind. Ebenso wenig haftet ABB für beiläufig entstandene oder Folgeschäden, die durch die Verwendung von in diesem Dokument beschriebener Software oder Hardware entstanden sind.

Weder dieses Dokument noch Teile davon dürfen ohne schriftliche Zustimmung von ABB reproduziert oder kopiert werden, der Inhalt darf nicht an eine dritte Partei weitergegeben werden, ebenfalls darf er nicht für unzulässige Zwecke genutzt werden.

Die in diesem Dokument beschriebene Software und Hardware unterliegt einer Lizenz und darf nur in Übereinstimmung mit den Lizenzbestimmungen genutzt, vervielfältigt oder weitergegeben werden. Dieses Produkt entspricht den Anforderungen der EMV-Richtlinie 2014/30/EU und der Niederspannungsrichtlinie 2014/35/EU.

Marken

Alle Urheberrechte sowie Rechte an eingetragenen Marken und Warenzeichen liegen bei ihren jeweiligen Eigentümern.

Inhaltsverzeichnis

4 Namensverwaltung

Anhang C DMS-API-Dateien

Stichwortverzeichnis

Hinweise zu diesem Handbuch

Vorsicht-, Achtung-, Information- und Tipp-Symbole

In diesem Dokument werden die folgenden Hinweise verwendet, um für die Sicherheit relevante und andere wichtige Informationen hervorzuheben: **Vorsicht**, **Achtung** und **Information**. Daneben existieren **Tipps**, um auf dem Leser nützliche Hinweise zu geben. Die zugehörigen Symbole haben folgende Bedeutung:

Stromschlag-Symbol: Weist auf Gefahren durch *Stromschlag* hin.

Vorsicht-Symbol: Weist auf Gefahren hin, die zu *Personenschäden* führen können.

Achtung-Symbol: Weist auf wichtige Informationen oder Warnungen in Zusammenhang mit dem im Text erläuterten Thema hin. Kann auf Gefahren hinweisen, die zu *Software-Datenverfälschungen* oder *Sachschäden* führen können.

Informations-Symbol: Weist den Leser auf wichtige Fakten und Voraussetzungen hin.

Tipp-Symbol: Weist auf Ratschläge hin, z.B. zum Projektentwurf oder zur Nutzung einer bestimmten Funktion.

Obwohl die mit **Vorsicht** bezeichneten Gefahren auf mögliche Personenschäden hinweisen und die mit **Achtung** bezeichneten Gefahren auf mögliche Sachschäden hinweisen, beachten Sie, dass die Benutzung beschädigter Ausrüstung zu Personenschäden, d.h. zu Verletzungen und auch zum Tode führen kann. Beachten Sie daher unbedingt die mit **Vorsicht** und **Achtung** gekennzeichneten Hinweise.

Terminologie

Das Glossar enthält Bezeichnungen und Abkürzungen, die ABB-spezifisch sind oder deren Gebrauch bzw. Definition von den in der Industrie üblichen Gepflogenheiten abweicht. Bitte machen Sie sich damit vertraut. Das Glossar finden Sie am Ende des ***Engineering-Handbuchs Systemkonfiguration.***

Typographische Konventionen

Zur Unterscheidung der verschiedenen Textelemente dienen in diesem Dokument die folgenden Konventionen:

- Für die Bezeichnung von Tasten werden Großbuchstaben verwendet, wenn diese auf der Tastatur benannt sind. Beispiel: Drücken Sie die ENTER-Taste.
- Drücken Sie STRG+C bedeutet, dass Sie die STRG-Taste gedrückt halten müssen, während Sie die Taste C drücken (in diesem Fall heißt das z.B., dass ein angewähltes Objekt kopiert wird).
- Drücken Sie **ESC, E, C** bedeutet, dass Sie die angegebenen Tasten nacheinander in der angegebenen Reihenfolge drücken müssen.
- Die Bezeichnungen von Schaltflächen bzw. Buttons werden fett hervorgehoben. Beispiel: Drücken Sie **OK**.
- Die Bezeichnungen von Menüs und Menüeinträgen werden fett dargestellt. Beispiel: das **Datei**-Menü.
  - Die folgende Darstellung wird für Menüaktionen verwendet: MenüName > MenüEintrag > UnterMenüEintrag
    Beispiel: Wählen Sie **Datei** > **Neu** > **Typ**
  - Das **Start**-Menü bezeichnet immer das **Start**-Menü auf der Windows-Taskleiste.

- Eingabeaufforderungen und Systemmeldungen werden in der Schriftart Courier dargestellt; Eingabe und Antworten des Anwenders werden in der Schriftart Courier fett dargestellt.

  Wenn Sie z. B. eine Eingabe machen, die außerhalb des zulässigen Wertebereichs liegt, wird die folgende Meldung angezeigt:

  ```
  Der eingegebene Wert ist ungültig. Der Wert muss
  zwischen 0 und 300 liegen.
  ```

  Oder Sie werden aufgefordert, die Zeichenfolge TIC132 in ein Feld einzugeben. Die Zeichenfolge wird wie folgt in der Prozedur dargestellt:

  TIC132

  Variablennamen werden mit Kleinbuchstaben dargestellt.

  sequence name

1 Applikationsschnittstelle Freelance für Windows

1.1 Allgemeine Beschreibung - Applikationsschnittstelle

DMS / API

steht für Digimatik Message Specification und Application programable Interface.

DMS ist ein Subset von MMS - Manufacturing Message Specification nach ISO 9506.

Der Applikationsrechner wird am Freelance -Systembus (Ethernet) betrieben und nutzt mit DMS / API die Kommunikation in gleicher Weise wie Freelance intern und damit alle zur Verfügung stehenden Möglichkeiten von Freelance.

DMS / API

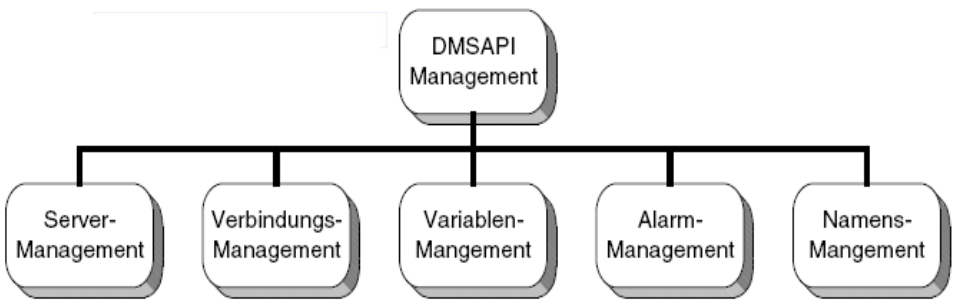
- ist die Applikationsschnittstelle um in Anwendungsprogrammen auf einem externen Rechner (Host) direkt mit Freelance zu kommunizieren.
- ist eine in "C" programmierte Funktionsbibliothek, die unter Windows läuft.
- wird zu einer programmierten Anwendung hinzugebunden und enthält alle nötigen Hochsprachenkommandos um in einfacher Weise schnellen Datenaustausch zwischen der Applikation und dem Freelance System zu führen.
- ist im Client - Server Konzept realisiert. Der Applikationsrechner wird als Gateway im Freelance Engineering aufgenommen und versorgt per Download den Applikationsrechner mit den freigegebenen Messstellen auf welche die Applikation zugreifen will.
- Das Setup von DMS / API wird auf Datenträger ausgeliefert und menügeführt installiert in gleicher Weise wie alle Freelance Produkte.

Das DMS kommt in folgenden Freelance Applikationen zum Einsatz:

- Freelance Engineering

- Freelance Operations
- Freelance - Prozessstationen
- Freelance - CSO Gateway
- Freelance - OPC Gateway

Die externen API-Anbindungen werden im Freelance Engineering als Gateway konfiguriert und geladen. Danach stehen auf allen Gatewaystationen die Freelance Messstellen-Adressen zur Verfügung.
Das implementierte DMS-API gliedert sich in folgende Funktionsbereiche:



Flow chart._gr.bmp

1.2 MMS (Manufacturing Message Specification ISO 9506)

MMS ist eine Norm um "Verständigungsschwierigkeiten" bei der Kommunikation zwischen Rechnern in der industriellen Automatisierung zu vermeiden. Diese Norm wurde 1988 als Internationaler Standard verabschiedet. Sie ging aus einer Initiative, die General Motors zu Beginn der 80er Jahre als MAP (Manufacturing Automation Protocol) startete, hervor.

Sie ist im OSI 7-Schichten Modell in der obersten Schicht, der Anwendungsschicht (ApplicationLayer) angesiedelt.

Anwendungsschicht (MMS)
Darstellungsschicht
Sitzungsschicht
Transportschicht
Netzwerkschicht
Sicherungsschicht
Physikalische Schicht

Anwendungen sind:

- Prozessleittechnik
- Speicherprogrammierbare Steuerungen
- numerische Steuerungen
- Roboter

MMS versucht Nachrichten und Anweisungen, die zwischen den einzelnen Systemkomponenten in einem heterogenen Rechnernetz ausgetauscht werden müssen, in eine Sprache zu fassen.

Dabei liegt den Diensten des MMS ein Client-Server Modell zugrunde. Ein Client stellt eine Dienstanforderung an einen Server. (Request). Der Server bearbeitet diesen Auftrag und gibt die Antwort (Response) an den Client zurück.

MMS beschreibt alles, was ein Server zu verstehen und auszuführen hat. Dazu werden Objekte definiert, mit denen etwas geschehen soll und Operationen auf diese Objekte, die beschreiben, was mit diesen Objekten geschehen soll. Es gibt insgesamt 16 verschieden Objekte und 79 Operationen für MMS.

1.3 DMS (Digimatik Message Specification)

DMS realisiert nur Teile des MMS. Bei der Erstellung des DMS wurde pragmatisch vorgegangen:

- welche Kommunikations-Anforderungen gibt es
- welche MMS-Dienste gibt es, die diese Anforderungen erfüllen

Freelance ist ein klassisches Client/Server-Modell. Der PC (als Engineeringstation bzw. Leitstation) regiert als Client und lässt von den MSR-Baugruppen Dienste (Messen/Steuern/Regeln) ausführen.

Zwischen Client und Server besteht eine logische Verbindung. Beide Seiten erkennen, wann eine Verbindung abgebrochen wird bzw. wann nach einem Abbruch neu aufgebaut wird.

Auf dem Engineering-PC werden Programme geschrieben, die in ausführbare Code übersetzt werden und deren Inhalt auf die MSR-Karte geladen werden soll.

Diese hinuntergeladenen Programmen werden von der MSR-Baugruppe ausgeführt, d.h. sie sind vom PC aus start- und stoppbar.

Die hinuntergeladenen Programme messen /steuern /regeln. Die Werte, die bei diesem Prozess entstehen, werden auf dem PC dargestellt bzw. archiviert.

Der Benutzer kann Werte des Prozesses lesen und aktiv verändern.

Grenzwerte, die während des Prozesses überschritten werden, werden auf dem PC sofort als Adresse dargestellt.

1.4 DMS / MMS -Funktionsbereiche

Environment and General Management Services bieten Dienste zur Verwaltung und Verbindungsbehandlung. Das Verbindungsmangement wurde im Freelance System nach den Bedürfnissen eines Prozessleitsystems optimiert.

Domain Management Services sind Dienste zum Laden und Verwalten von Programm und Datenbereichen. Im DMS wurden folgende Dienste implementiert:

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

Dienste des Domain Managements sind nicht im DMS / API enthalten.

Program Invocation Management Services bietet Dienste zum kreieren, starten, stoppen, zurücksetzen und löschen von Programmen.

Dabei wurde bei der Implementierung im DMS die Dienste CreatePI und DeletePI "gespart". Sie werden automatisch durch Download bzw. Löschen der Domain (TaskDomains) gelöscht bzw. kreiert.

- StartPI
- StopPI
- ResetPI

Variable Access Services dienen dem Lesen und Schreiben von Variablen aus dem laufenden Prozess. DMS hat dazu folgende Dienste implementiert.

- Read
- Write
- Define Named Variable List
- Information Report; der Server sendet "unaufgefordert" diesen Report ohne dass hierfür auf der DMS-Schicht eine Quittung verlangt wird. In Freelance werden die Reports für Langzeitarchive von Kurvendaten, Störablaufprotokolle und zum kurzfristigen Updaten von Wertefenstern/Trendkurven bzw. allen Grafiken im Freelance Operations verwendet.

Event Management Services bieten ereignisgesteuerte Dienste wie Alarmierung und Quittierung:

- GetAlarmSummary
- EventNotification
- Acknowledge EventNotification

Journal Management befasst sich mit dem Abspeichern und Abrufen von Informationen. In Freelance gibt es hierfür die konfigurierbaren Bausteine:

- Trendbaustein
- Signalfolgeprotokol

Die Information in diesen Bausteinen kann über die Dienste des Variable Access ausgelesen werden.

Freelance Name Management dient zum koordinierten Zugriff auf gültige Variablen- und MSR-Stellennamen in Freelance und Wandlung auf Freelance Adressen.

1.5 Adressierbare Freelance Objekte

1.5.1 Variablen

- vordefinierte Variablen (projektunabhängig)
- benutzerdefinierte Variablen
- benutzerdefinierte strukturierte Variablen
- Kurven und Störablaufprotokolle
- Variablen der MSR-Stellen

Alle Variablen werden über folgende Adressierung gelesen:

- Stationsnummer
- ObjektNummer
- KomponentNummer
- Typ der Variablen

Die Umwandlung von VariablenNamen -> Freelance Adressierung geschieht über die Funktionen des Namensmanagement.

1.5.2 MSR-Stellen

- Funktionsbausteine
- Ablaufsteuerung

- MSR-Tasks
- MSR-Programmlisten
- MSR-IPC-Programme

MSR-Stellen werden folgendermaßen adressiert:

  – Stationsnummer
  – ObjektNummer
  – KlassenNummer

1.5.3 Systemobjekte

- MSR-Ressource

1.6 Freelance -Kommunikationschichtenmodell

Anwendungsschicht (DMS)
Schicht 6 (fehlt)
BasisTransport
TCP / UDP
IP-Protocol
CSMA/CD -Verfahren
Physikalische Schicht

Auf Freelance wurde die Kommunikationsschicht aufgeteilt in eine DMS (Freelance Message Specification)-Schicht und eine BTR(Basis-Transport)-Schicht. Die Kommunikation von einer Clientapplikation über das Netz zu einer Serverapplikation läuft nach folgendem Schema:

ClientApplikationen-> DMS-> BTR-> TCP/IP-> Kabel-> TCP/IP-> BTR-> DMS Server

Während die DMS-Schicht betriebssystemunabhängig in C implementiert wurde, gibt es die BTR-Schicht betriebssystemabhängig für folgende Plattformen:

- PSOS
- WINDOWS

Im betriebsystemabhängigen Kommunikationsteil werden die verschiedenen Kommunikations-Tasks verwaltet:

Unter PSOS / NT gibt es z.B. folgende verschiedene Tasks:

- ListenTask (wird nur gebraucht, falls Station serverfähig ist)
- SendeTasks (wartet an MailBox auf Sendeaufträge)
- ReceiveTasks (wartet an Socket auf eintreffende Daten)
- UDP-Tasks :
  - Senden / Empfangen von zyklischen Variablen Listen
  - Zeitsynchronisation
  - Verbindungsaufbau

Die betriebssystemabhängige Kommunikationsschicht kennt weder die Struktur der Kommunikations Pakete, noch die Stationsnummern der Stationen oder sonstige Informationen.

Für eine Portierung auf ein anderes Betriebssystem müssen nur diese Schicht, sowie Funktionen zur Speicher- und Semaphoren-Verwaltung neu implementiert werden.

1.7 Installation von DMS / API

Das DMS / API wird als Setup auf Datenträger ausgeliefert und ist wie alle Freelance-Produkte menügeführt auf dem Applikationsrechner zu installieren.

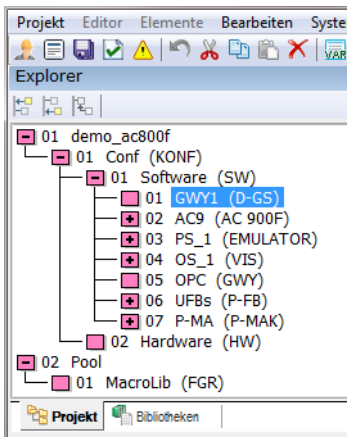
Soll DMS / API auf dem gleichen Rechner wie Freelance Engineering installiert werden, muss das DMS / API ins Standard Verzeichnis von Freelance geladen werden.

Die DLL von DMS / API müssen im gleichen Verzeichnis wie die von Freelance sein.

1.8 Konfiguration des DMS / API-Gateway im Freelance Engineering

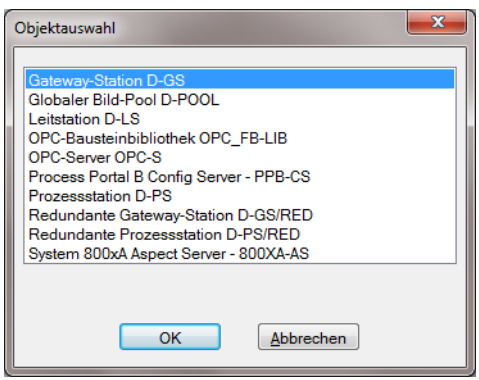
Damit Adressinformationen verfügbar sind, muss im Freelance Engineering jedes Gateway im Projektbaum angelegt und konfiguriert werden.

Siehe auch *Engineering-Handbuch Systemkonfiguration*.



project_tree_gr.png

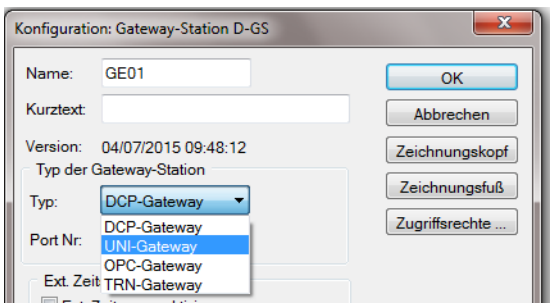
Soll unter der Konfiguration eine neue API/DMS-Ressource konfiguriert werden, wird in der Konfiguration der Stationstyp Gateway-Station ausgewählt.



object_selection_gr.png

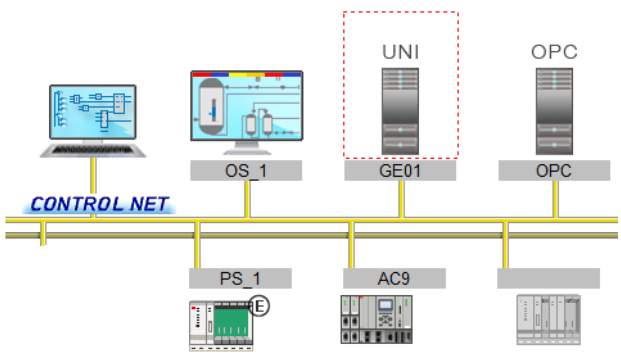
Beim Editieren des Gateway Kopf Dialoges kann der Gateway Typ festgelegt werden:

- DCP-Gateway
- UNI-Gateway (für eigene API-Applikationen)
- OPC-Gateway
- TRN-Gateway



gateway_type_gr.png

Das Gateway stellt sich im Hardware-Editor als PC dar, da es aus Sicht der Prozessstation wie eine Leitstation wirkt.



hw_struc_gr.png

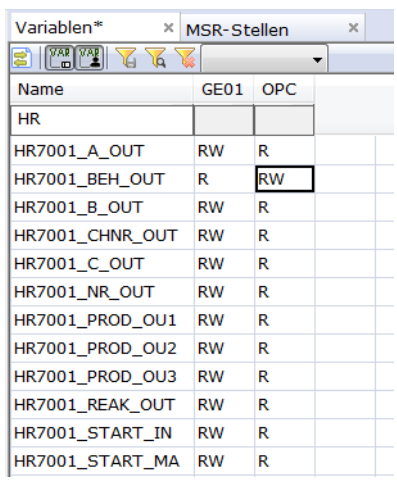
Nach dem Einfügen des Gateways wird konfiguriert zu welchen Prozessstationen das Gateway Lese- und Schreibdienste durchführen soll.

Zusätzlich muss konfiguriert werden, für welche Variablen und MSR-Stellen das Gateway die Adressinformation bekommen soll. Neben einer Normalansicht gibt es

in den Variablen und MSR-Stellen eine Stationsansicht in der für jede Variable und jeden MSR-Stelle ein Lese- und/oder Schreibflag gesetzt werden kann.

conf_gateway_gr.png

Variablenliste



var_list_gr.png

MSR-Stellenliste

Variablen | MSR-Stellen

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_gr.png

1.9 Laden des DMS/API-Gateways

1.9.1 Erstkonfiguration

Die Prozessstationen sind immer vor den Gateways zu laden. Nach dem Laden können die Gateways auf die Freelance Adressinformation zugreifen.

1.9.2 Umkonfiguration

Bei Umkonfiguration der Prozessstationen müssen auf Prozessstationen und Gateways die geänderten Objekte geladen werden. Die Prozessstation kennt weder Variablennamen noch MSR-Stellennamen sondern nur die DMS-Adressierung mit Objektnummer und Komponentennummer.

Durch "ungeschicktes" Umkonfigurieren ist es möglich, dass 2 Objekte die Objektnummer wechseln (Löschen und Neueinfügen von Objekten). In solchen Fällen können Schreibzugriffe durch das Gateway auf die Prozessstation (nach dem Laden der Prozessstation und vor dem Laden des Gateways) zu einem ungewollten Verhalten führen.

Werden nur Objekte zu einer Prozessstation dazugefügt bzw. geändert und keine Objekte gelöscht oder verschoben ändert sich die Adressierung der alten Objekte

nicht. In diesem Fall erhält das Gateway nur Information über neue Objekte. Es kann zu keiner Fehlbedienung durch das Gateway kommen.

Dies sollte bei der Erstellung einer eigenen DMS/API-Applikation berücksichtigt werden.

Es gibt z.B. folgende Lösungsmöglichkeiten:

- "vor dem Laden der Prozessstationen durch Benutzereingriff das DMS/API-Gateway in einen Konfigurationszustand bringen
- "wird aus Freelance Engineering heraus auf das DMS/API-Gateway vor dem Laden der Prozessstation neu initialisiert, kann die zu schreibende Applikation darauf reagieren.
- "die Versionskontrolle aktivieren:
  - Lesezugriffe werden immer zugelassen
  - Schreibzugriffe werden nur bei Versionsgleichheit zugelassen.
- "nach einer Umkonfiguration des Gateways eigene "Datenbank" überprüfen und Lesezugriffe und AlarmSummary evtl. neu aufsetzen.
- "Die DMS/API-Applikation wird von jeder Umkonfigurierung des Gateways durch Freelance Engineering über Callbackfunktionen benachrichtigt.

1.10 DMS / API-Funktionsübersicht

Clientmanagement, Evironment and General Management Services

DMSAPI_Init	Initialisieren einer DMS-Sitzung
DMSAPI_Exit	Beenden einer DMS-Sitzung
DMSAPI_ConnectByName	Verbindung zu einem DMS-Server aufbauen
DMSAPI_ConnectByAddr	Verbindung zu einem DMS-Server aufbauen
DMSAPI_ConnectByNo	Verbindung zu einem DMS-Server aufbauen
DMSAPI_Disconnect	Verbindung zu einem DMS-Server aufbauen
DMSAPI_GetConnectionData	Verbindung zu einem DMS-Server überprüfen
DMSAPI_SetSystemTime	Uhrzeit im Freelance System setzen
DMSAPI_SetSystemTimeBy-DmsType	Uhrzeit im Freelance System setzen
DMSAPI+_SetSystemTimeBy-String	Uhrzeit im Freelance System setzen
DMSAPI_RestartResource	Warm-, Kaltstarten bzw. Toggeln einer Freelance Station

Variable Access Services

DMSAPI_VLCreate	Erzeugen einer Variablenliste
DMSAPI_VLAddWriteVarByName	Hinzufügen einer Variablen zum Schreiben
DMSAPI_VLAddReadVarByName	Hinzufügen einer Variablen zum Lesen
DMSAPI_VLAddWriteVarByAddr	Hinzufügen einer Variablen zum Schreiben
DMSAPI_VLAddReadVarByAddr	Hinzufügen einer Variablen zum Lesen

Variable Access Services

DMSAPI_VLCreate	Erzeugen einer Variablenliste
DMSAPI_VLChangeValue	Ändern des Wertes innerhalb der Variablenliste
DMSAPI_VLDelVar	Löschen einer Variablen aus Variablenliste
DMSAPI_VLClear	Löschen aller Variablen aus einer Variablenliste
DMSAPI_VLRead	Einfaches Lesen einer Variablenliste
DMSAPI_VLReadCycle	Zyklisches Lesen einer Variablenliste
DMSAPI_VLWrite	Einfaches Schreiben einer Variablenliste
DMSAPI_VLStopCycleVar	Stoppen einer zyklischen Variablenliste
DMSAPI_VLDelete	Löschen einer Variablenliste

Event Management Services

DMSAPI_GetAlarmSummary	Anfordern des AlarmSummary eines DMS-Server. Ab diesem Zeitpunkt sendet der Server automatisch alle anfallenden Alarme
DMSAPI_AckAlarmByList	Durchführung der Quittierung mit vollständig ausgefüllten Liste

Client Receive

DMSAPI_RegisterClientCB	Registrieren einer anwenderprogrammierten CallbackFunktion, die bei Empfangen von DMS-Nachrichten für den Client aufgerufen werden.
API_CallbackReceive	CallbackReceiveFunktion wird beim Empfangen von DMS-Nachrichten asynchron aufgerufen

Freelance Names Management

DMSAPI_SetProjectDir	Setzen eines Projektpfades zum Laden / Speichern von Konfigurationsinformation
DMSAPI_ChangeProject	Wechsel auf ein anderes Projekt
DMSAPI_LockOV	Sperren des Namensmanagement gegen Umkonfiguration durch Freelance Engineering
DMSAPI_UnlockOV	Aufheben der Sperrung
DMSAPI_GetProjectInfo	Holen der aktuellen Projektversion
DMSAPI_GetFirstResourceInfo	Anfordern der Konfigurationsinformation für die erste Station
DMSAPI_GetNextResourceInfo	Anfordern der Konfigurationsinformation für die alle weiteren Stationen
DMSAPI_GetFirstVarInfo	Anfordern der Konfigurationsinformation für die erste Variable
DMSAPI_GetNextVarInfo	Anfordern der Konfigurationsinformation für die alle weiteren Variablen
DMSAPI_GetFirstTagInfo	Anfordern der Konfigurationsinformation für die erste MSR-Stelle
DMSAPI_GetNextTagInfo	Anfordern der Konfigurationsinformation für alle weiteren MSR-Stellen
DMSAPI_GetTagByAddr	Anfordern der Konfigurationsinformation für eine bestimmte MSR-Stelle
DMSAPI_GetFirstCmpOfObjCls	Anfordern der Konfigurationsinformation für die erste Komponente einer Objektklasse
DMSAPI_GetNextCmpOfObjCls	Anfordern der Konfigurationsinformation für alle weiteren Komponenten einer Objektklasse

Freelance Names Management

DMSAPI_GetVarnameByOPath	Umwandlung eines Variablennamens in Freelance Adressinformation
DMSAPI_GetVarInfoByName	Umwandlung einer Freelance Adressinformation in einen Variablennamen

DMS Utilities

DMSAPI_SetVarCode	Setzen der Umwandelformate
DMSAPI_GetValueByString	Umwandlung String -> DMS value
DMSAPI_GetStringByValue	Umwandlung DMS-Value - > string
DMSAPI_GetVarLen	Länge die eine Variable innerhalb einer Variablenliste benötigt
DMSAPI_DumpRecData	Gibt die Struktur der Receivedaten auf STDOUT aus

2 Basic Transport Application Interface (BTR)

Die BasisTransport-Schicht ist protokollunabhängig. Es ist möglich sowohl mit dem P-Protkoll (das Protokoll für AC 870P / Melody) als auch mit dem Freelance Protokoll aufzusetzen.

Dieses Kapitel kann übersprungen werden, falls das DMS-API auf einem Betriebssystem läuft, auf dem die BTR-Schicht Implementiertung vorhanden ist. Auf anderen Betriebssystemen muss die BTR-Schicht neu implementiert werden.

Die BTR-Schicht muss von den aufrufenden Applikationen initialisiert werden. (In diesem Fall ist das DMS-API die Applikation und nicht die Appliktation, die das DMS-API benutzt.)

Die Initialisierungsroutine heißt:

BTR_Init

Vor dem Beenden sollte die Applikation die Routine

BTR_Exit aufrufen.

Die BTR-Schicht verbindet eine Clientapplikation mit einer Serverapplikation. Innerhalb der BTR-Schicht wird jede Verbindung durch einen eindeutigen ConnectionHandle identifiziert.

Die Applikationen stellen der BTR-Schicht drei "Callback-Funktionen" zur Verfügung:

AbortProc, wird aufgerufen, wenn eine Verbindung abbricht. Als Übergabeparameter wird der ConnectionHandle und der Grund des Verbindungsabbruchs mitgeteilt.

KeepAliveProc, wird aufgerufen, wenn innerhalb eines TimeOuts kein Sendeauftrag vorliegt. Übergabeparameter ist neben dem ConnectionHandle ein Buffer, in den das (protokollspezifische) Paket zur Verbindungsüberwachung kodiert werden muss.

ReceiveProc, wird aufgerufen, wenn Daten auf einer Verbindung ankommen. Übergabeparameter ist neben dem ConnectionHandle ein Buffer, in dem die Daten stehen.

2.1 Funktionsweise für (TCPIP)

Die Serverapplikation ruft eine Prozedur "BTR_OpenServer" auf.

Die Serverapplikation übergibt dabei die drei CallbackFunktionen an die BTR-Schicht.

Durch Aufruf der OpenServerProzedur wird eine Task (PSOS) / Thread (WindowsNT) gestartet, die darauf wartet, dass ein Client versucht eine Verbindung aufzubauen.

Bei jedem Verbindungsaufbau werden automatisch zwei weitere Tasks/Threads gestartet:

- Send
- Empfang (Receive)

3 DMS ClientManagement

Alle Dienste, die eine Aktion auf dem Ethernet auslösen, bzw. auf ein Ereignis auf dem Ethernet warten, besitzen ein SyncFlag und ein TimeOut. Spätestens nach Ablauf dieses Timeouts kehrt die Prozedur zurück. Das SynchronFlag kann folgende Werte annehmen:

- synchron mit Receive (kommt die Antwort nicht innerhalb des Zeitintervalls, muss sie mit Receive abgeholt werden)
- synchron mit Callback (kommt die Antwort nicht innerhalb des Zeitintervalls, wird die Callback-Funktion aufgerufen)
- asynchron mit Receive (das Timeout gilt nur für das Senden der Nachricht)
- asynchron mit Callback (das Timeout gilt nur für das Senden der Nachricht)

Alle Prozeduren mit Antwort sind gekennzeichnet mit:

Die Antworten werden im Kapitel "Empfangen/Dekodieren von Daten" beschrieben.

In vielen Prozeduren existiert der Fehler:

E_DMSAPI_INTERNAL_ERROR

In diesem Fall ist das DMS auf einen nicht bekannten Fehler gelaufen, z.B.:

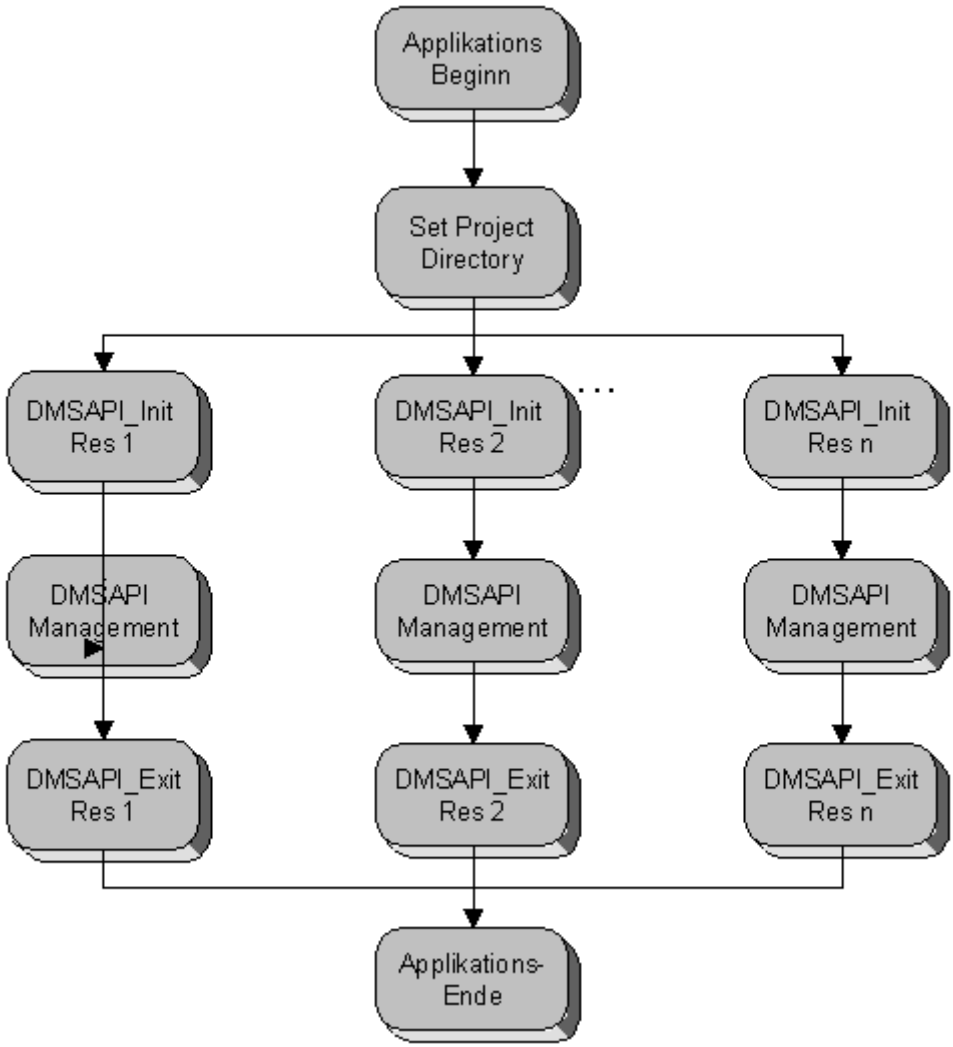
- Die Applikation hat mit einem uninitialisierten Pointer auf Daten des DMS geschrieben
- Das TCPIP ist aus unerklärlichen Gründen nicht mehr lauffähig.

Die Applikation sollte möglichst "benutzerdatenschonend" verlassen und neu gestartet werden

3.1 Environment and General Management Services

3.1.1 Initialisierung und Beendigung einer DMS-Sitzung

Das DMSAPI ist "Multiprojekting" fähig, d.h. eine DMSAPI-Applikation kann in mehrenen Freelance -Prokjekten als Gateway mit verschiedenen Ressourcenummern eingetragen werden. Jedes Freelance Engineering kann für "sein" Gateway die Konfiguration hinunterladen. Über die verschiedenen Projekte kann die DMSAPI-Applikation gleichzeitig auf die verschiedenen Ressourcen und Objekte der einzelnen Projekte zugreifen.


Applikations Beginn
Set Project Directory
DMSAPI_Init Res 1
DMSAPI_Init Res 2
DMSAPI_Init Res n
DMSAPI Management
DMSAPI Management
DMSAPI Management
DMSAPI_Exit Res 1
DMSAPI_Exit Res 2
DMSAPI_Exit Res n
Applikations-Ende

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_BOOLEAN     bStandardServer/* use of StandardServer */
)
```

Initialisierung der DMS-Applikationsschicht. Übergeben wird die eigene Ressourcenummer und die Anzahl der gleichzeitig bestehenden Serververbindungen. Soll die zu schreibende Applikation von Freelance Engineering mit der Adressinformation versorgt werden, ist als Ressourcennummer die gleiche zu wählen, die in Freelance Engineering für das Gateway vergeben wurde. Die Anzahl der Serververbindungen ist auf 1 zu setzen.

Jede Ressource kann nur ein Projekt gleichzeitig verwalten. Soll über das DMS-API auf das Namensmanagement mehrerer Projekte gleichzeitig zugegriffen werden, ist die Initialisierungsroutine mit verschiedenen Ressourcennummern mehrmals aufzurufen. In den verschiedenen Projekten muss dann auch das Gateway mit diesen verschiedenen Ressourcenummern konfiguriert werden.

Soll eine eigene DMSAPI-Serverapplikation geschrieben werden, ist als Wert für den Parameter bStandardServer auf FALSE einzusetzen. In diesem Fall müssen die Prozeduren aus Kapitel 7 benutzt werden.

Wird keine Serverfunktionalität benötigt, wird die NoOfServerConn auf 0 gesetzt.

Im DMSDEF.H stehen in der Section:

- DMSAPI_MAX_APPLICATION
- DMSAPI_MAX_CONNECTION

Parameter:

- OwnResNo RessourcenNummer der eigenen Station innerhalb des Freelance Systems. Logische DMS-Verbindungen bestehen immer zwischen 2 Ressourcen (Wert liegt zwischen 1 und 255). Auf einem Rechner können mehrere logische Ressourcen initialisiert werden. Diese müssen unterschiedliche Ressourcennummern bekommen.
- OwnResType: eigener RessourceTyp
  - DMS_OS_DIGIVIS
  - DMS_OS_DIGITOOL
  - DMS_OS_EPROM
  - DMS_OS_MSR
  - DMS_OS_DDE_GWY
  - DMS_OS_P_GWY
  - DMS_OS_GWY
    (dieser Typ wird für DMSAPI-Applikationen üblicherweise benutzt)
- NoOfSrvConn Anzahl der möglichen Serververbindungen
- bStandardServer:
  TRUE: Applikation kann als Server zu Freelance Engineering eingesetzt werden => Freelance Engineering kann die Adressinformation auf den Server hinunterladen. Das Namensmangement wird durch das Laden aktiviert.
  FALSE: Applikation kann eigene Serverfunktionen implementieren Possible return values:

Mögliche Returnwerte:

E_DMSAPI_ALREADY_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer schon initialisiert war.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INTERNAL_ERROR	Fehler bei der Initialisierung der DMS-Schicht.

E_DMSAPI_ALREADY_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer schon initialisiert war.
E_DMSAPI_MAX_CONNECTION	Die Anzahl der Serververbindungen überschreitet die Anzahl der möglichen Verbindungen.
E_DMSAPI_MAX_APPLICATION	Die Funktion kann nur von einer maximalen Anzahl von Applikationen aufgerufen werden.

DMSAPI_Exit

SYNTAX

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

Beenden der DMS-Applikationsschicht für eine Ressource. Alle an dieser Ressource hängenden DMS-Objekte (Verbindungen, Variablen, ...) werden aufgeräumt.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station innerhalb des Freelance Systems. Logische DMS-Verbindungen bestehen immer zwischen 2 Ressourcen. (Wert liegt zwischen 1 und 255).

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert war.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INTERNAL_ERROR	Fehler beim Beenden der DMS-Schicht.

3.1.2 Verbindungsmanagement

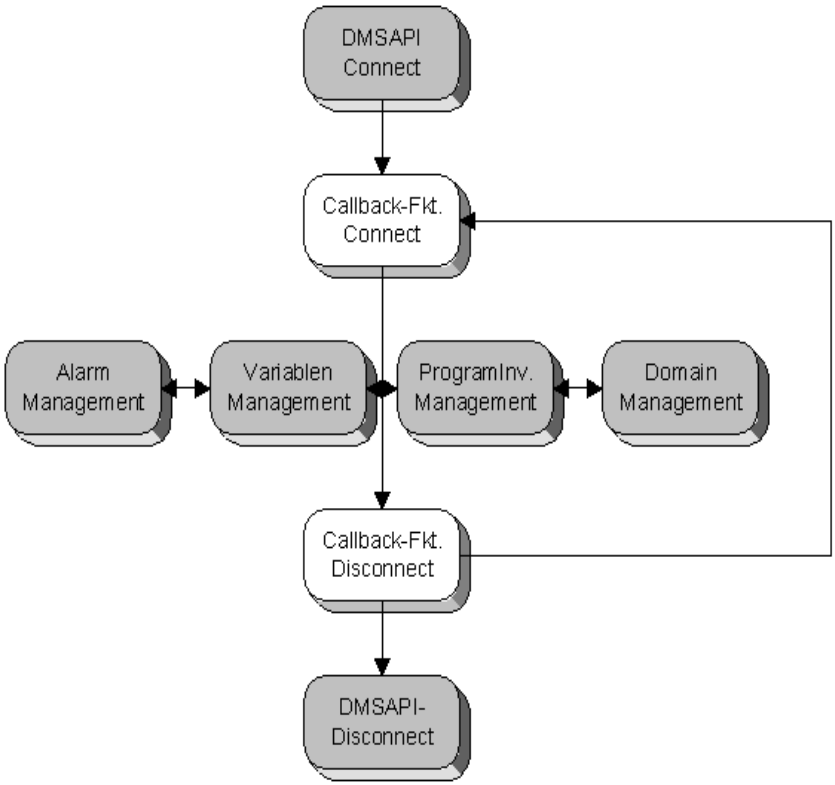
Skizzierung des Verbindungsaufbau zwischen Client und Serverstation

- Die Clientstation sendet über UDP einen DMSNameServerrequest an die beiden übergebenen IPAdressen und einen festeingestellten UDP-Port.
- Beide Serverstationen senden eine DMSNameServerResponse zurück. In dieser Response steht als Information:
  - Station ist Primary oder Secondary
  - Station hat eine ControlPortNummer, auf der die TCPIP-Verbindung aufgebaut werden kann
- Clientstation führt zum (als ersten antwortenden) Primary einen Connect auf den übertragenden ControlPort aus
- Serverstation, die auf diesem Controlport mit einem "Listen" gewartet hat, lässt die Verbindung öffnen.
- Clientstation schickt auf der geöffneten Verbindung ein DmsInit-Kommando, in der sie folgende Information überträgt:
  - Portnummer des UDP-Ports, an den die zyklischen Variablenlisten gesendet werden sollen
  - Timeout , mit der die Verbindung überwacht werden soll
  - Versionsnummer der aktuellen DMS-Version
  - eigene Ressourcenummer
- Serverstation schickt auf dieses DMSInit-Kommando eine DMSInit-ResponseOwn resource number
  - eigene Ressourcenummer
  - eigener Ressourcetyp
  - Versionsnummer
- Clientstation überprüft die zurückgebenen Werte und gibt den Status an die Callbackfunktionen der Applikationen

Sicht einer DMSAPI-Applikation auf die Prozessstation

Die Callbackfunktionen werden automatisch bei Verbindungsauf- und abbau von der DMSAPI-Schicht aufgerufen.

Die Funktionsbereiche ProgramInv. und Domainmanagement werden nur von Freelance Engineering genutzt.



DMSAPI_ConnectByAddr

SYNTAX

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

```
    DMS_UINT32          ulIPAddr2       /* 2.IPAddr. Res */,
    DMS_RES_NO          ResNo           /* Resource No */,
    DMS_RES_TYPE        ResType         /* Resource Typ */,
    DMS_UINT16          uKeepAliveT     /* KeepAliveTimeout */,
    DMS_CONN_HANDLE*lpConnHandle /* ConnectionHandle */,
    DMS_INT16           nSyncFlag       /* Synchron Flag */,
    DMS_UINT32          ulProcT         /* ProzedurTimeout */,
    DMS_UINT32          ulRecConnLen    /* Grösse des Speichers    auf den
                                                        Pointer referenziert */,
    DMS_REC_CONN_DATA   *RecConn        /* RecStruct der Conn */
)
```

Verbindungsaufbau zu einer DMS-ServerStation: Der Rückgabewert ist ein Verbindungshandle, über den später die verbundene Ressource identifiziert wird. Dadurch ist es möglich, zu einer Ressource mehrere Verbindungen aufzubauen. Dann kann z.B. eine Verbindung für Alarmierung, eine weitere für Aktualisierung oder Bedienung benutzt werden. Es können aber auch alle Dienste auf einer Verbindung ausgeführt werden. Handelt es sich bei der Serverstation um eine redundant ausgelegte Station wird automatisch die Verbindung zur aktiven Station geöffnet. Zwischen Client und Serverstation findet eine Verbindungsüberwachung statt. Die Verbindung wird mit dem Parameter "KeepAliveTimeout" überwacht. Ist das Synchronflag auf DMSAPI_SNYC gesetzt und wird die Verbindung innerhalb des angegebenen Prozedurtimeout aufgebaut wird die ConnectionStruktur mit Werten gefüllt.

Nach einem Verbindungsabbruch wird die Verbindung automatisch neu aufgebaut. Benötigt die Applikation diese Verbindung nicht mehr, ist die Prozedur DMSAPI_-Disconnect aufzuru-fen.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- nBTRLnk: Gibt an über welche BTR-Schicht die DMS-Dienste übertragen werden.

DMS_BTR_TCPIP (

DMS_BTR_REDLNK (Redundancy link on process station)

- ulIPAddr1: IPAdresse der ServerStation
- ulIPAddr2: 2. IPAdresse der ServerStation, falls diese redundant ausgelegt ist.
- ResNo: RessourcenNummer der Serverstation. Sind auf der ServerStation mehrere RessourcenNummern installiert, wird die Verbindung zum richtigen Server hergestellt.
- ResType: RessourcenTyp der Serverstation. Gültige Werte sind:
  - 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_GWY
  - DMS_OS_GWY
- uKeepAliveT: Verbindungsüberwachung in Sekunden/ Millisekunden zwischen den beiden Ressourcen, d.h. ein Verbindungabbruch (z.B. bei Kabelbruch, Ausfall der verbundenen Ressource, o.ä.) wird spätestens nach Ablauf des Timeouts erkannt. Senden die beiden Ressourcen keine Daten, tauschen sie innerhalb des halben Timeouts ein KeepAlivePaket aus.
- lpConnHandle: Nach dem Verbindungsaufbau wird der Datenaustausch auf dieser Verbindung über diesen ConnectionHandle adressiert.
- nSyncFlag

DMSAPI_SYNCHRON: Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf den Verbindungsaufbau. Wird die Verbindung aufgebaut liefert die RecStruct gültige Werte zurück. Auch nach Ablauf des Timeouts läuft der Verbindungsaufbau weiter.

DMSAPI_ASYNCHRON: Der Verbindungsaufbau wird über die Callback-Funktion angezeigt bzw. kann über die Funktion DMSAPI_GetConnectionStatus gelesen werden. Das Prozedurtimeout wird nicht ausgewertet.

- ulProcT:

DMSAPI_NO_TIMEOUT kein Timeout Wert in Millisekunden DMSAPI_WAIT_-FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt wer-den kann.

ulRecConnLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt. Dann hat es die Länge der Structur DMS_REC_CONN_DATA

- **RecConn:**

typedef struct DMS_REC_CONN_DATA

```
{DMS_RES_NO        OwnResNo;        /* Eigene RessourcenId */
 DMS_RES_NO        ResNo;           /* RessourcenId der Station */
 DMS_RES_TYPE      ResType;         /* RessourcenTyp der Serverstation */
DMS_CONN_STATUS    ConnStatus;      /* Verbindungsstatus der Station */
DMS_UINT32         ulIPAddr;        /* IPAdresse der verb. Station */
DMS_UINT32         ulBoardType;     /* BoardType*/
DMS_UINT32         ulConnFlag;      /*  */
} DMS_REC_CONN_DATA;
```

Der ConnStatus kann folgende Werte annehmen:

```
DMS_CONN_OK,                        /* alles in Ordnung */
DMS_CONN_ABORT,                     /* keine Verbindung */
DMS_CONN_INVALID_RES_TYPE, /* falscher RessourceTyp */
DMS_CONN_INVALID_RES_NO,/* falsche Ressourcennummer */
DMS_CONN_NO_OS,             /* kein Betriebssystem */
DMS_CONN_SECONDARY,  /* Nur zum Secondary connected =>
                        falsche Konfiguration */
DMS_CONN_INVALID_VERSION /* falsche DMS_Version */
```

Der ulBoardType kann folgende Werte annehmen:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

Das ulConnFlag kann folgende Werte annehmen:

DMS_RES_PRIMARY /* Verbindung zu einem Primary Server */

DMS_RES_SECONDARY /* Verbindung zu einem Secondary Server */

DMS_RES_CLIENT /* Verbindung zu einem Client */

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_TIMEOUT	Verbindungsaufbau konnte innerhalb des angebenen Timeouts nicht durchgeführt werden.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Verbindungsaufbau
E_DMSAPI_MAX_CONNECTION	Die DMS-Schicht erlaubt nur eine maximale Anzahl von Verbindungen.

DMSAPI_ConnectByName

SYNTAX

DMS_RC DMSAPI_ConnectByName (

DMS_RES_NO OwnResNo /* Own Resource No */,

DMS_CHAR *ResName /* Name der resource */,
DMS_CONN_HANDLE *lpConnHandle /* ConnectionHandle */,

DMS_INT16 nSyncFlag /* Synchron flag */,

```
DMS_UINT32          ulProcT          /* Prozedur timeout */,
DMS_UINT32          ulRecConnLen     /*Größe des Speichers auf
                                       den Pointer referenziert */,
DMS_REC_CONN_DATA   *RecConn         /* RecStruct der Conn. */
)
```

Die Prozedur baut die Verbindung zu einer DMS-Ressource auf. Für die eigene Ressource muss das von Freelance Engineering geladene Projekt greifbar sein. Der Ressourcennname wird über die Funktionen des Namensmanagemet umgewandelt zu:

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

Verbindungsaufbau zu einer DMS-ServerStation: Der Rückgabewert ist ein Verbindungshandle, über den später die verbundene Ressource identifiziert wird. Dadurch ist es möglich, zu einer Ressource mehrere Verbindungen aufzubauen. Dann kann z.B. eine Verbindung für Alarmierung, eine weitere für Aktualisierung oder Bedienung benutzt werden. Es können aber auch alle Dienste auf einer Verbindung ausgeführt werden. Handelt es sich bei der Serverstation um eine redundant ausgelegte Station wird automatisch die Verbindung zur aktiven Station geöffnet. Zwischen Client und Serverstation findet eine Verbindungsüberwachung statt. Die Verbindung wird mit dem Parameter KeepAliveTimeout überwacht. Ist das Synchronflag auf DMSAPI_SNYC gesetzt und wird die Verbindung innerhalb des angegebenen Prozedurtimeout aufgebaut wird die ConnectionStruktur mit Werten gefüllt .

Nach einem Verbindungsabbruch wird die Verbindung automatisch neu aufgebaut. Benötigt die Applikation diese Verbindung nicht mehr, ist die Prozedur DMSAPI_-Disconnect aufzuru-fen.

Parameter

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ResName: Name der Ziel-Ressource, wie er in Freelance Engineering konfiguriert wurde. .
- lpConnHandle: Nach dem Verbindungsaufbau wird der Datenaustausch auf dieser Verbindung über diesen ConnectionHandle adressiert.
- nSyncFlag

DMSAPI_SYNCHRON: Die Prozedur wartet, solange wie das angegebene ProzedurTimeout, auf den Verbindungsaufbau. Wird die Verbindung aufgebaut liefert die RecStruct gültige Werte zurück. Auch nach Ablauf des Timeouts läuft der Verbindungsaufbau

DMSAPI_ASYNCHRON: Der Verbindungsaufbau wird über die Callback-Funktion angezeigt bzw. kann über die Funktion DMSAPI_GetConnectionStatus gelesen werden. Das Prozedurtimeout hat keine Bedeutung.

- **ulProcT:**
  DMSAPI_NO_TIMEOUT kein Timeout Wert in Millisekunden

  DMSAPI_WAIT_FOREVER:

Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

- **ulRecConnLen**: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt. Dann hat es die Länge der Structur DMS_REC_CONN_DATA
- **RecConn:**

```
typedef struct DMS_REC_CONN_DATA {
        DMS_RES_NO      OwnResNo;     /*Eigene Stationsnummer */
        DMS_RES_NO      ResNo;        /* StationsNummer */
        DMS_RES_TYPE    ResType;      /* RessourcenTyp der
                                         Serverstation*/
```

```
DMS_CONN_STATUSConnStatus;              /* Verbindungsstatus der Station*/
        DMS_UINT32      ulIPAddr;       /*IPAdresse der verb. Station*/
        DMS_UINT32      ulBoardType;    /*  Board type
        DMS_UINT32      ulConnFlag;   /*  */
} DMS_REC_CONN_DATA;
```

Der ConnStatus kann folgende Werte annehmen:

```
DMS_CONN_OK,                        /* alles in Ordnung */
DMS_CONN_ABORT,                     /* keine Verbindung */
DMS_CONN_INVALID_RES_TYPE,  /* falscher RessourceTyp */
DMS_CONN_INVALID_RES_NO,  /* falsche
                                    Ressourcennummer */
DMS_CONN_NO_OS,                 /* kein Betriebssystem */
DMS_CONN_SECONDARY,             /* Nur zum Secondary
                        connected => falsche Konfiguration */
DMS_CONN_INVALID_VERSION    /* falsche DMS_Version */
```

Der ulBoardType kann folgende Werte annehmen:

```
DMS_CPU_UNKNOWN
DMS_CPU_DCP02
DMS_CPU_DCP10
DMS_CPU_PC
```

Das ulConnFlag kann folgende Werte annehmen:

```
DMS_RES_PRIMARY             /* Verbindung zu einem Primary Server*/
DMS_RES_SECONDARY           /* Verbindung zu einem Secondary Server*/
DMS_RES_CLIENT              /* Verbindung zu einem Client */
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_NO_CONF	kein Projekt vorhanden
E_DMSAPI_INVALID_CONF	keine Information über die angegebene Station vorhanden
E_DMSAPI_NO_RESOURCE	Die Station kann zur Zeit nicht connected werden, da sich noch Stationen im Zustand disconnecting befinden.
E_DMSAPI_TIMEOUT	Verbindungsaufbau konnte innerhalb des angegebenen Timeouts nicht durchgeführt werden.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Verbindungsaufbau
E_DMSAPI_MAX_CONNECTION	Die DMS-Schicht erlaubt nur eine maximale Anzahl von Verbindungen.

DMSAPI_ConnectByNo

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         /* Synchron flag */,

DMS_UINT32          ulProcT           /* Prozedurtimeout */,

DMS_UINT32          ulRecConnLen      /* Größe des Speichers
                                      auf den Pointer referenziert */,
```

DMS_REC_CONN_DATA *RecConn /*RecStruct der Conn */

)

Die Prozedur baut die Verbindung zu einer DMS-Ressource auf. In der eigenen Ressource muss das von Freelance Engineering geladene Projekt greifbar sein. Die Ressourcennnummer wird über die Funktionen des Namensmanagemet umgewandelt zu:

- BTRLnk
- IPAddr1
- IPAddr2
- KeepAliveTimeout
- ResourceNo
- ResourceTyp

Verbindungsaufbau zu einer DMS-ServerStation: Der Rückgabewert ist ein Verbindungshandle, über den später die verbundene Ressource identifiziert wird. Dadurch ist es möglich, zu einer Ressource mehrere Verbindungen aufzubauen. Dann kann z.B. eine Verbindung für Alarmierung, eine weitere für Aktualisierung oder Bedienung benutzt werden. Es können aber auch alle Dienste auf einer Verbindung ausgeführt werden. Handelt es sich bei der Serverstation um eine redundant ausgelegte Station wird automatisch die Verbindung zur aktiven Station geöffnet. Zwischen Client und Serverstation findet eine Verbindungsüberwachung statt. Die Verbindung wird mit dem Parameter KeepAliveTimeout überwacht. Ist das Synchronflag auf DMSAPI_SNYC gesetzt und wird die Verbindung innerhalb des angegebenen Prozedurtimeout aufgebaut wird die ConnectionStruktur mit Werten gefüllt .

Nach einem Verbindungsabbruch wird die Verbindung automatisch neu aufgebaut. Benötigt die Applikation diese Verbindung nicht mehr, ist die Prozedur DMSAPI_-Disconnect aufzuru-fen.

Parameter

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ResNo: Nummer der Ziel-Ressource, wie sie in Freelance Engineering konfiguriert wurde.

- lpConnHandle: Nach dem Verbindungsaufbau wird der Datenaustausch auf dieser Verbindung über diesen ConnectionHandle adressiert.
- nSyncFlag

  DMSAPI_SYNCHRON: Die Prozedur wartet, solange wie das angegebene ProzedurTimeout, auf den Verbindungsaufbau. Wird die Verbindung aufgebaut liefert die RecStruct gültige Werte zurück. Auch nach Ablauf des Timeouts läuft der Verbindungsaufbau

  DMSAPI_ASYNCHRON: Der Verbindungsaufbau wird über die Callback-Funktion angezeigt bzw. kann über die Funktion DMSAPI_GetConnectionStatus gelesen werden. Das Prozedurtimeout hat keine Bedeutung.

- ulProcT:

DMSAPI_NO_TIMEOUT kein timeout

Wert in Millisekunden

DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

- ulRecConnLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt. Dann hat es die Länge der Structur DMS_REC_CONN_DATA
- RecConn:

```
typedef struct DMS_REC_CONN_DATA {
DMS_RES_NO          OwnResNo;          /* Eigene Stationsnummer */
DMS_RES_NO          ResNo;             /* StationsNummer der Station */
DMS_RES_TYPE        ResType;           /*RessourcenTyp der Serverstation */
DMS_CONN_STATUS  ConnStatus;           /*Verbindungsstatus der Station */
DMS_UINT32          ulIPAddr;          /*IPAdresse der verb. Station */
DMS_UINT32       ulBoardType;          /*  BoardType */
DMS_UINT32       ulConnFlag;   /*  */
} DMS_REC_CONN_DATA;
```

Der ConnStatus kann folgende Werte annehmen:

DMS_CONN_OK, /* alles in Ordnung */

DMS_CONN_ABORT, /* keine Verbindung */

DMS_CONN_INVALID_RES_TYPE, /* falscher RessourceTyp */

DMS_CONN_INVALID_RES_NO, /* falsche Ressourcennummer */

DMS_CONN_NO_OS, /* kein Betriebssystem */

DMS_CONN_SECONDARY, /* Nur zum Secondary connected => falsche Konfiguration */

DMS_CONN_INVALID_VERSION /* falsche DMS_Version */

Der ulBoardType kann folgende Werte annehmen:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

Das ulConnFlag kann folgende Werte annhemen:

DMS_RES_PRIMARY /* Verbindung zu einem Primary Server r */

DMS_RES_SECONDARY /* Verbindung zu einem Secondary Server */

DMS_RES_CLIENT /* Verbindung zu einem Client */

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_NO_CONF	kein Projekt vorhanden

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_CONF	keine Information über die angegebene Station vorhanden
E_DMSAPI_NO_RESOURCE	Die Station kann zur Zeit nicht connected werden, da sich noch Stationen im Zustand disconnecting befinden
E_DMSAPI_TIMEOUT	IVerbindungsaufbau konnte innerhalb des angegebenen Timeouts nicht durchgeführt werden.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Verbindungsaufbau.
E_DMSAPI_MAX_CONNECTION	Die DMS-Schicht erlaubt nur eine maximale Anzahl von Verbindungen.

DMSAPI_Disconnect

SYNTAX

DMS_RC DMSAPI_Disconnect(

DMS_CONN_HANDLE ConnHandle /* ConnectionHandle */

)

Die Funktion führt einen geregelten Verbindungsabbruch zur angegebenen Ressource durch. Vor dem Beenden einer DMS-Sitzung sollten alle Ressourcen wieder freigegeben werden. Das Connhandle wird nicht nach Abschluss der Prozedur freigegeben, sondern erst nach dem geregelten Verbindungsabbau. Vorher werden noch die angegebenen CallbackFunktionen aufgerufen, d.h erst nach Aufruf der Callbackfunktionen werden die Handles freigegeben.

Parameter:

ConnHandle: ConnectionHandle, der beim Aufruf der Prozedur DMSAPI_Connect zurückgegeben wurde .

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Verbindungsabbau

DMSAPI_GetConnectionData

SYNTAX

```
DMS_RC DMSAPI_GetConnectionData(
                DMS_CONN_HANDLE ConnHandle/* ConnectionHandle */,
                DMS_UINT32      ulRecConnLen /* Größe des Speichers auf
                                                den Pointer referenziert*/
DMS_REC_CONN_DATA *RecConn              /* ReceiveStructure of Conn. */
)
```

Die Funktion liefert zu einem gültigen Verbindungshandle die Verbindungsstruktur zurück. Ist das DMS ohne Callback-Funktion installiert, muss über diese Funktion der Verbindungsstatus der einzelnen Ressourcen kontrolliert werden.

Parameter:

- ConnHandle: ConnectionHandle, der beim Aufruf der Prozedur DMSAPI_Connect zurückgegeben wurde .
- ulRecConnLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt. Dann hat es die Länge der Structur DMS_REC_CONN_DATA
- RecConn:

```
typedef struct DMS_REC_CONN_DATA {
DMS_RES_NO          OwnResNo;       /*Eigene Stationsnummer */
```

```
DMS_RES_NO              ResNo;          /*StationsNummer der Station */
DMS_RES_TYPE            ResType;    /* RessourcenTyp der Serverstation */
DMS_CONN_STATUSConn     Status;         /* Verbindungsstatus der Station */
DMS_UINT32              ulIPAddr;   /*IPAdresse der verbundenen Station */
DMS_UINT32              ulBoardType;    /*  BoardType */
DMS_UINT32              ulConnFlag;     /*  */
} DMS_REC_CONN_DATA;
```

Der ConnStatus kann folgende Werte annehmen:

```
DMS_CONN_OK,                            /*alles in Ordnung */
DMS_CONN_ABORT,                         /* keine Verbindung */
DMS_CONN_INVALID_RES_TYPE,              /* falscher RessourceTyp */
DMS_CONN_INVALID_RES_NO,                /* falsche Ressourcennummer */
DMS_CONN_NO_OS,                         /* kein Betriebssystem */
DMS_CONN_SECONDARY,                     /* Nur zum Secondary connected =>
                                           falsche Konfiguration*/
DMS_CONN_INVALID_VERSION                /* falsche DMS_Version */
```

Der ulBoardType kann folgende Werte annehmen:

- DMS_CPU_UNKNOWN
- DMS_CPU_DCP02
- DMS_CPU_DCP10
- DMS_CPU_PC

Das ulConnFlag kann folgende Werte annehmen:

```
DMS_RES_PRIMARY         /* Verbindung zu einem Primary Server */
DMS_RES_SECONDARY       /* Verbindung zu einem Secondary Server */
DMS_RES_CLIENT          /* Verbindung zu einem Client */
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Verbindungsaufbau

DMSAPI_SetSystemTime

SYNTAX

DMS_RC DMSAPI_SetSystemTime(

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

)

Die Funktion sendet die angegebene Zeit als Broadcast auf dem Ethernet an alle angeschlossenen Freelance-Stationen. Es gibt keine Quittung, ob dieses Zeitpaket bei irgendeiner Station angekommen ist.

Unter Windows lässt sich die aktuelle Uhrzeit über die Funktion GetLocal auslesen.

Parameter:

- Time: Typ ist der WindowsTyp SYTEMTIME, der die zu stellende Zeit beinhaltet.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft. Es werden keine Zeiten vor 1984 (Beginn der MMS-Time) gesendet
E_DMSAPI_INTERNAL_ERROR	Interner Fehler beim Zeitsenden

DMSAPI_RestartResource

SYNTAX

```
DMSAPI_RestartResource(
                DMS_CONN_HANDLE     ConnHandle   /* ConnectionHandle */,
                DMS_RESTART_REASON  RestartRea          /* RestartReason */
)
```

Diese Prozedur führt auf der Freelance Prozessstation einen Kalt- oder Warmstart durch.

Nach dem Start wird die Prozessstation neu gebootet. D.h. die Verbindung ab- und aufgebaut.

Parameter:

- Connhandle: ConnectionHandle für diese Ressource
  - RestartReason

    DMSAPI_RESTART_WARM: Warmstart der Prozessstation

    DMSAPI_RESTART_COLD: Kaltstart der Prozessstation
    DMSAPI_RESTART_TOGGLE: Umschaltung der Prozessstation

von Primary zu Secondary. Diese Befehl kann nur an eine redundante Prozessstation gesendet werden.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_CONNECTION	Keine Verbindung zu dieser Station.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.2 Variable Access Services

Über die Dienste des Variablenmanagements können Daten von einem DMS-Server:

- einmalig gelesen
- zyklisch gelesen
- einmalig geschrieben werden.

Über das DMS-API werden vollständige Variablenlisten gelesen und geschrieben. Es müssen für die verschiedenen Dienste (Einmaliges Lesen, Einmaliges Schreiben, Zyklisches Lesen) leere Listen erzeugt werden, in die dann Variablen eingefügt werden. Alle Variablen innerhalb einer Variablenliste werden zum gleichen Zeitpunkt gelesen bzw. geschrieben. Gleicher Zeitpunkt heißt hier, dass die Berechnung der MSR-Funktionen und Tasks für die Dauer der Variablenlistenoperation unterbrochen wird.

Nach Beendigung der Lese- bzw. Schreiboperation kann die Variablen-Liste folgendermaßen weiterverwendet werden:

==> Löschen von vorhandenen Variablen aus der Variablenliste

==> Hinzufügen von neuen Variablen in die Variablenliste

==> Ändern von Werten innerhalb der Variablenliste

==> Löschen von allen Variablen aus der Variablenliste

==> Löschen der Variablenliste

Danach kann der Lese bzw.Schreibdienst erneut ausgeführt werden.

Nicht mehr benötigte Variablenlisten müssen immer explizit gelöscht werden. Auch beim Verbindungsabbruch zu einer Station muss die Variablenliste gelöscht werden. Nach dem Löschen einer Variablenliste wird für diese Variablenliste nichts mehr empfangen. Wird das Löschen von Variablenlisten von der Applikation "vergessen", kann die Applikation nach einer bestimmten Anzahl von verlorengegangenen Variablenlisten keine neue mehr kreieren.

Zyklische Variablenlisten müssen vor Änderungen gestoppt werden und können nach dem Ändern neu gestartet werden. Auch nach dem Stoppen einer Variablenliste wird für diese Variablenliste nichts mehr empfangen.

Einmalig zu lesende oder zu schreibende Variablenlisten lassen sich erst nach Empfang der Antwort ändern. (Es macht wenig Sinn Variablenlisten vor Empfang der Antwort zu löschen)

In einer Variablenliste können nur Variablen der gleichen Station enthalten sein.

3.3 Achtung !!!

Das Lesen / Schreiben von Variablen belastet den DMS-Server. Werden die Lese- oder Schreibroutinen aus dem API zyklisch aufgerufen, eventuell so schnell wie möglich, führt dies auf den Prozessstation zu einer CPU-Belastung von bis zu 80 %. Deswegen sollten folgende Regeln beachtet werden:

- für zyklische Leseaufträge auch den Dienst ReadCycleVarList benutzen und nicht selbst zyklisch den Dienst ReadVarList benutzen.

- zum Lesen und Schreiben möglichst viele Aufträge in einer Variablenliste durchführen und nicht jeden Variablendienst in einer eigenen Variablenliste durchführen. Erst Variablenanforderungen sammeln und dann durchführen.

Im normalen Betrieb antwortet die Prozessstation nach einer Zeit von 20 -100 msec.

Die Struktur der Variablenliste ist auf Seite B-70, Empfangen / Dekodieren von Daten beschrieben.

Die Prozeduren für Variablenlistenzugriffe lassen sich in folgendermaßen gliedern:

Erzeugen einer Variablenliste: DMSAPI_VLCreate

Ändern der Variablenliste: : DMSAPI_VLAddWriteVarByName(nur für Write)
DMSAPI_VLAddReadVarByName(nur für Read)
DMSAPI_VLAddWriteVarByAddr(nur für Write)
DMSAPI_VLAddReadVarByAddr(nur für Read)
DMSAPI_VLChangeValue (nur für Write)
DMSAPI_VLDelVar
DMSAPI_VLClear

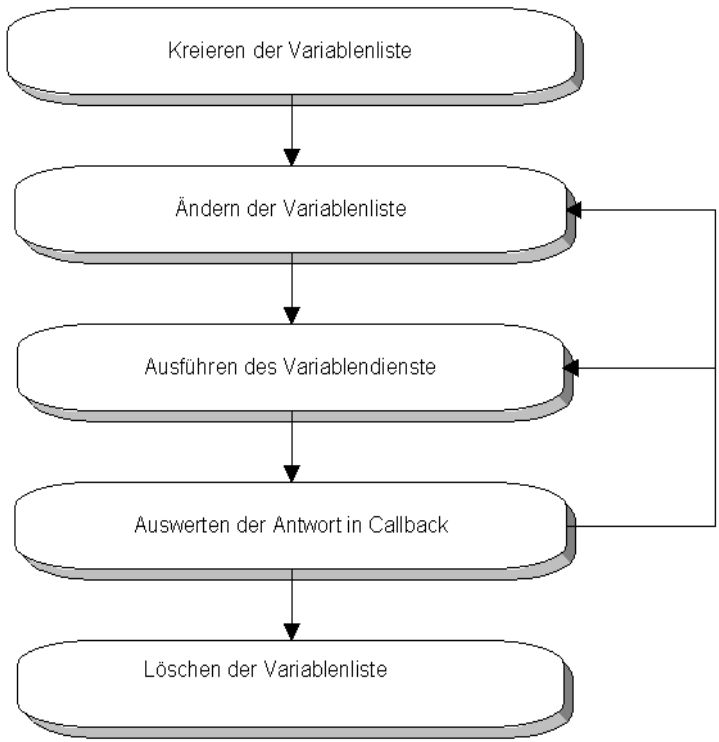
einfache Variablendienste: DMSAPI_VLRead
DMSAPI_VLWrite

zyklische Variablendienste: DMSAPI_VLReadCycle

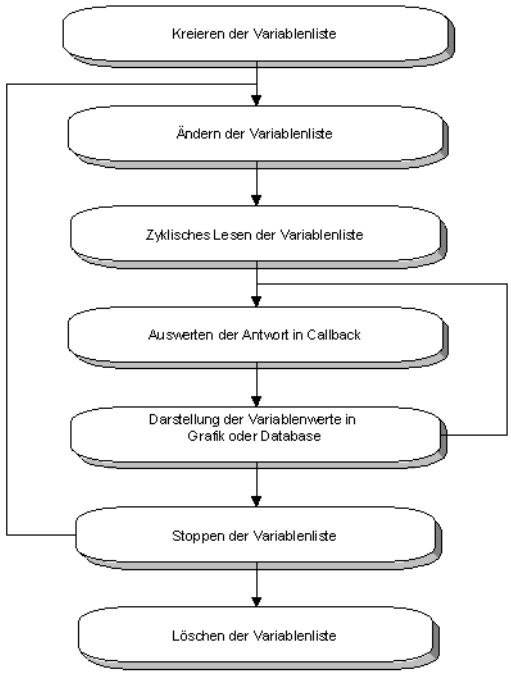
stoppen zykl. Variablenlisten:DMSAPI_VLStopCycle

Löschen von Variablenlisten:DMSAPI_VLDelete

Lebensdauer einer Variablenliste für Lese- und Schreibdienste



Lebensdauer einer Variablenliste für zyklische Lesedienste



3.3.1 DMSAPI_VLCreate

SYNTAX

```
DMS_RC DMSAPI_VLCreate(
        DMS_CONN_HANDLE  ConnHandle /*  ConnectionHandle */,
        DMS_INT16        nVLService     /* Art des VarList Service */,
        DMS_HANDLE       *lpDmsHandle   /* Identifier für Varlist */
)
```

Durch diese Prozedur wird der Speicher und ein eindeutiger DMS-Handle für eine DMS-Variablenliste erzeugt. Nach dem Erzeugen einer Variablenliste können Variablen in diese Liste eingefügt werden.

Gefüllte Variablenlisten können über die Dienste Read/ Readcycle/ Write genutzt werden.

Speicher und DMS-Handle werden nur über die Funktion DMSAPI_DeleteVarList gelöscht.

Parameter:

- Connhandle: ConnectionHandle für diese Ressource
- nVLService:

DMSAPI_VL_SINGLE_READ: einmaliges Lesen dieser Variablenliste

DMSAPI_VL_CYCLE_READ: zyklisches Lesen dieser Variablenliste

DMSAPI_VL_SINGLE_WRITE: einmaliges Schreiben dieser VariablenlistelpD-msHandle Handle dieser Variablenliste, über den alle weiteren Operationen auf diese Variablenliste gesteuert werden.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Keine Ressourcen(Speicher / DMSHandles), um diese Variablenliste zu kreieren
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_VLAddReadVarByName

SYNTAX

```
DMS_RC DMSAPI_VLAddReadVarByName(
   DMS_HANDLE          DmsHandle          /*  VarListHandle */,
   DMS_CHAR            *lpszVarname       /* Variablenname */,
```

```
    DMS_REC_VARLIST_DATA  **lplpRecVar      /* Pointer auf RecVarStruct */,
    DMS_INT16             *lpnIndex         /* Index in RecVarStruct */
)
```

Diese Prozedur fügt zu einer bestehenden Variablenliste ein Element hinzu. Das Element wird über den Variablennamen adressiert. Die Prozedur wandelt Variablennamen zu DMS-Adressen um. Dadurch benötigt sie mehr Rechenzeit als die Routine

DMSAPI_VLAddVarReadByAddr.

Nachdem eine Variablenliste gefüllt wurde, kann der gewünschte Dienst ausgeführt werden. Erst nachdem ein Lesedienst beendet (bzw. gestoppt) wurde, kann die Variablenliste über die Add und DeleteDienste verändert werden.

Eine Variablenliste kann nur eine begrenzte Anzahl von Bytes aufnehmen. Diese Anzahl von Bytes bestimmt die Anzahl der Variablen die kommuniziert werden. Da Variablen verschiedener Datentypen unterschiedlich viel Speicher benötigen, lässt sich keine Konstante DMSAPI_MAX_VAR_IN VARLIST definieren. Die Prozedur DMSAPI_GetVarLen gibt den Speicherbedarf der einzelnen Variablentypen innerhalb einer Variablenliste zurück. Die Konstante DMSAPI_VL_MAX_BYTES gibt die maximale Speichergröße einer Variablenliste an. Die Struktur DMS_REC_-VARLIST_DATA gibt immer den freien Speicherplatz innerhalb der Variablenliste an.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- lpszVarname: Name der zu lesenden Variablen
- lplpRecVar: Struktur der ausgefüllten Variablenliste

```
typedef struct DMS_REC_VARLIST_DATA {
        DMS_HANDLEDmsHandle;
        DMS_INT16    ActVarNo;   /* aktuelle Anzahl von Variablen */
        DMS_INT16   MaxVarNo;   /* max. Anzahl von Variablen mit
                                   leeren Einträgen */
        DMS_INT16   FreeBytes;    /* Anzahl von freien Bytes in der VL */
```

```
        DMS_REC_VAR *lpVar;     /*Eigentliche Variablen-Liste */
} DMS_REC_VARLIST_DATA;
```

Die Struktur DMS_REC_VAR:

```
typedef struct DMS_REC_VAR {
        DMS_VAR_STATUS   VarStatus;      /* Status der variable */
        DMS_VAR_RC       VarRc;          /* ReturnCode nach Dienst */
        DMS_OBJ_PATH     ObjPath;        /* ObjektPfad auf Server */
        DMS_CHAR         VarName;        /* Variablenname bzw. NULL */
        DMS_UINT32       ValueSize;      /* Größe des ValueBuffer */
        DMS_VAR_TYPE     VarType;        /*Typ des Wertes */
        DMS_VALUE        *VarValue;      /* Wert der Variablen oder
                                            NULL */
} DMS_REC_VAR;
```

Der VarStatus kann folgende Werte annehmen:

DMS_VAR_NOT_VALID	Nach Einfügen des Wertes und vor Ausführen des Dienstes bzw. falls bei Ausführung des Dienstes Fehler auftrat
DMS_VAR_CHANGED	Nach Ausführen eines Dienstes
DMS_VAR_DELETED	Variable wurde über DMSAPI_VLDelVar gelöscht, der Eintrag ist noch vorhanden

Der VarRc kann verschiedene Fehler vom Server annehmen. Siehe Anhang Fehlercodes.

Der ObjPath hat die Struktur DMS_OBJ_PATH und beinhaltet die Adressierung auf dem Server.

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

} DMS_OBJ_PATH;

Der VarType nimmt verschiedene Werte an. (siehe Anhang DMS-Variablentypen)

Der VarValue ist ein Pointer der auf den Wert der Variablen nach Durchführung des Lesedienstes (siehe Anhang DMS-Variablentypen) zeigt.

- lpnIndex innerhalb der lplpRecVar

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Variablenliste ist voll. Es muss eine neue Variablenliste angelegt werden.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_NO_CONF	kein Projekt vorhanden
E_DMSAPI_INVALID_CONF	keine Information über die angegebene Variable vorhanden
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INVALID_CONN_HANDLE	Die angegebene Variable befindet sich nicht auf dem angegebenen Ressource
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_VLAddWriteVarByName

SYNTAX

```
DMS_RC DMSAPI_VLAddWriteVarByName(
                DMS_HANDLE      DmsHandle       /*  VarListHandle */,
                DMS_CHAR        *lpszVarname    /*  Variablenname */,
```

```
DMS_VAR_TYPE        VarType;                /* Typ des Wertes */
DMS_VALUE           *lpVarValue;        /* Wert der Variablen
                                                oder NULL */
DMS_REC_VARLIST_DATA **lplpRecVar   /* Pointer auf
                                                RecVarStruct */,
DMS_INT16           *lpnIndex       /*  Index in RecVarStruct */
)
```

Diese Prozedur fügt zu einer bestehenden Variablenliste ein Element hinzu. Das Element wird über den Variablennamen adressiert. Die Prozedur wandelt Variablennamen zu DMS-Adressen um. Dadurch benötigt sie mehr Rechenzeit als die Routine

DMSAPI_VLAddVarWriteByAddr.

Nachdem eine Variablenliste gefüllt wurde, kann der Schreibdienst ausgeführt werden. Nachdem ein Schreibdienst beendet wurde, kann die Variablenliste über die "Add, Change Delete"-Dienste verändert werden.

Eine Variablenliste kann nur eine begrenzte Anzahl von Bytes aufnehmen. Diese Anzahl von Bytes bestimmt die Anzahl der Variablen die kommuniziert werden. Da Variablen verschiedener Datentypen unterschiedlich viel Speicher benötigen, lässt sich keine Konstante DMSAPI_MAX_VAR_IN VARLIST definieren. Die Prozedur DMSAPI_GetVarLen gibt den Speicherbedarf der einzelnen Variablentypen innerhalb einer Variablenliste zurück. Die Konstante DMSAPI_VL_MAX_BYTES gibt die maximale Speichergröße einer Variablenliste an. Die Struktur DMS_REC_-VARLIST_DATA gibt immer den freien Speicherplatz innerhalb der Variablenliste an.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- lpszVarname: Name der zu lesenden Variablen
- VarType nimmt die folgenden verschiedenen Werte an. (siehe Anhang DMS-Variablentypen)
- lpVarValue ist eine Referenz auf den Wert der Variablen zur Durchführung des Schreibdienstes (siehe Anhang DMS-Variablentypen).

- lplpRecVar: Struktur der ausgefüllten Variablenliste
- lpnIndex: Index innerhalb der lplpRecVar

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Variablenliste ist voll. Es muss eine neue Variablenliste angelegt werden.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_VARTYP	Übergebene Variable ist vom falschen Typ.
E_DMSAPI_INVALID_NO_CONF	kein Projekt vorhanden
E_DMSAPI_INVALID_CONF	keine Information über die angegebene Variable vorhanden
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INVALID_CONN_HANDLE	Die angegebene Variable befindet sich nicht auf dem angegebenen Ressource
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_VLAddReadVarByAddr

SYNTAX

```
DMS_RC DMSAPI_VLAddReadVarByName (
        DMS_HANDLE          DmsHandle          /*  VarListHandle */,
        DMS_OBJ_PATH        lpOPath;       /* Objektpfad auf server */
        DMS_VAR_TYPE        VarType;           /* Typ des Wertes */
        DMS_REC_VARLIST_DATA**lplpRecVar   /* Pointer auf
                                              RecVarStruct */,
        DMS_INT16           *lpnIndex      /*  Index in
                                              RecVarStruct*/ )
```

Diese Prozedur fügt zu einer bestehenden Variablenliste ein Element hinzu. Das Element wird über den Objektpfad und den Variablentyp adressiert. Die Prozedur hat zu der Prozedur DMSAPI_VLAddReadVarByName einen Zeitvorteil, da der Variablenname nicht in eine DMS-Adresse gewandelt werden muss.

Nachdem eine Variablenliste gefüllt wurde, kann der gewünschte Dienst ausgeführt werden. Nachdem ein LeseDienst beendet (bzw. gestoppt) wurde, kann die Variablenliste über die Add und DeleteDienste verändert werden.

Eine Variablenliste kann nur eine begrenzte Anzahl von Bytes aufnehmen. Diese Anzahl von Bytes bestimmt die Anzahl der Variablen die kommuniziert werden. Da Variablen verschiedener Datentypen unterschiedlich viel Speicher benötigen, lässt sich keine Konstante DMSAPI_MAX_VAR_IN VARLIST definieren. Die Prozedur DMSAPI_GetVarLen gibt den Speicherbedarf der einzelnen Variablentypen innerhalb einer Variablenliste zurück. Die Konstante DMSAPI_VL_MAX_BYTES gibt die maximale Speichergröße einer Variablenliste an. Die Struktur DMS_REC_-VARLIST_DATA gibt immer den freien Speicherplatz innerhalb der Variablenliste an.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- ObjPath hat die Struktur DMS_OBJ_PATH und beinhaltet die Adressierung auf dem
- Server.
- typedef struct {

  DMS_OBJNO ObjNo;

  DMS_CMPNOCmpNo;

  } DMS_OBJ_PATH;
- VarTypenimmt die folgenden verschiedenen Werte an. (siehe Anhang DMS-Variablentypen)
- lplpRecVar: : Struktur der ausgefüllten Variablenliste
- lpnIndex: Index innerhalb der lplpRecVar

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Variablenliste ist voll. Es muss eine neue Variablenliste angelegt werden.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlisten-handle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_AddWriteVarByAddr

SYNTAX

```
DMS_RC DMSAPI_VLAddWriteVarByName(
        DMS_HANDLE          DmsHandle          /*  VarListHandle */,
        DMS_OBJ_PATH        lpOPath;           /* Objektpfad auf
                                                  server */
        DMS_VAR_TYPE        VarType;           /* Typ des Wertes */
        DMS_VALUE           *lpVarValue;       /* Wert der Variablen oder
                                                  NULL */
 DMS_REC_VARLIST_DATA**lplpRecVar          /* Pointer auf RecVarStruct */,
      DMS_INT16             *lpnIndex          /*  Index in RecVarStruct */
)
```

Diese Prozedur fügt zu einer bestehenden Variablenliste ein Element hinzu. Das Element wird über den Objektpfad und den Variablentyp adressiert. Die Prozedur hat zu der Prozedur DMSAPI_VLAddWriteVarByName einen Zeitvorteil, da der Variablenname nicht in eine DMS-Adresse gewandelt werden muss.

Nachdem eine Variablenliste gefüllt wurde, kann der gewünschte Dienst ausgeführt werden. Nachdem ein SchreibDienst beendet wurde, kann die Variablenliste über die Add, Change und DeleteDienste verändert werden.

Eine Variablenliste kann nur eine begrenzte Anzahl von Bytes aufnehmen. Diese Anzahl von Bytes bestimmt die Anzahl der Variablen die kommuniziert werden. Da Variablen verschiedener Datentypen unterschiedlich viel Speicher benötigen, lässt sich keine Konstante DMSAPI_MAX_VAR_IN VARLIST definieren. Die Prozedur DMSAPI_GetVarLen gibt den Speicherbedarf der einzelnen Variablentypen innerhalb einer Variablenliste zurück. Die Konstante DMSAPI_VL_MAX_BYTES gibt die maximale Speichergröße einer Variablenliste an. Die Struktur DMS_REC_-VARLIST_DATA gibt immer den freien Speicherplatz innerhalb der Variablenliste an.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- 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 nimmt die folgenden verschiedenen Werte an. (siehe Anhang DMS-Variablentypen)
- lpVarValue ist eine Referenz auf den Wert der Variablen zur Durchführung des Schreibdienstes (siehe Anhang DMS-Variablentypen).
- lplpRecVar: Struktur der ausgefüllten Variablenliste
- lpnIndex: Index innerhalb der lplpRecVar

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Variablenliste ist voll. Es muss eine neue Variablenliste angelegt werden.
E_DMSAPI_INVALID_VARTYP	Übergebene Variable ist vom falschen Typ.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_VLChangeValue

SYNTAX

```
DMS_RC DMSAPI_VLChangeValue(
        DMS_HANDLE      DmsHandle           /* VarListHandle */,
        DMS_INT16       nIndex          /*  Index in RecVarStruct */,
        DMS_VAR_TYPE    VarType;        /* Typ des Wertes */
        DMS_VALUE       *lpVarValue;    /* Wert der Variablen oder
                                           NULL*/
        DMS_REC_VARLIST_DATA**lplpRecVar  /* Pointer auf
                                           RecVarStruct */,
)
```

Diese Prozedur ändert in einer bestehenden Variablenliste den Wert, der geschrieben werden soll. Die Variablenliste darf zu dem Zeitpunkt des Prozeduraufrufs nicht auf die Antwort des vorherigen Schreibzugriff warten.
Die Prozedur wird benötigt, falls mehrmals hintereinander auf die gleichen Variablen geschrieben werden soll. Die Variablenliste muss nicht jedesmal neu kreiert werden.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- nIndex: Index innerhalb der lplpRecVar
- VarType nimmt die folgenden verschiedenen Werte an. (siehe Anhang DMS-Variablentypen)
- lpVarValue ist eine Referenz auf den Wert der Variablen zur Durchführung des Schreibdienstes (siehe Anhang DMS-Variablentypen).

- lplpRecVar: Struktur der ausgefüllten Variablenliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_INDEX	Ungültiger Index in der Variablenliste.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_VARTYP	Übergebene Variable ist vom falschen Typ.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

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/* auf RecVarStruct */,
)
```

Diese Prozedur löscht aus einer bestehenden Variablenliste die Variable, die über den Index adressiert wird. Die Indizes der anderen Variablen werden nicht verändert. Beim Neueinfügen von Variablen in die Variablenliste werden diese Lücken ausgefüllt.

Die Prozedur wird benötigt, falls in einer Grafik durch Benutzereingriff Einblendbilder geöffnet und geschlossen werden können.

Befindet sich eine Variable im zyklischen Lesezugriff, muss sie vor dem Löschen von Variablen gestoppt werden.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste

- nIndex: Index innerhalb der lplpRecVar
- lplpRecVar: Struktur der ausgefüllten Variablenliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_INDEX	Ungültiger Index in der Variablenliste.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlisten-handle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.3 DMSAPI_VLClear

SYNTAX

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

Diese Prozedur löscht aus einer bestehenden Variablenliste alle Variablen. Danach können neue Variablen in die Variablenliste eingefüllt werden.Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.4 DMSAPI_VLRead

SYNTAX

```
DMS_RC DMSAPI_VLRead(
                DMS_HANDLE      DmsHandle           /*  VarListHandle */;
                DMS_INT16         nCBId             /* CallbackId */,
                DMS_INT16       nSyncFlag       /* Synchron flag */,
                DMS_UINT32        ulProcT         /* ProzedurTimeout */,
                DMS_UINT32      ulRecVarLen ^       /* Größe des Speichers
                                                    auf den Pointer referenziert */,
DMS_REC_VARLIST_DATA          *lpRecVar           /* RecStruct der VL */
```

Diese Prozedur führt zu einer gefüllten Variablenliste den einfachen Lesedienst aus. Auf diese Anfrage gibt es eine Antwort. Nach Erhalt und Auswertung der Antwort, kann die Variablenliste über die Lösch- und Zufügprozeduren verändert und neu gelesen werden bzw. komplett gelöscht werden.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- nCBId: CallbackId, bzw. wird die Variablenliste über die DMSAPI-Receivefunktion abgeholt; CBId bekommt den Wert DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON: Die Prozedur wartet, die angegebene ProzedurTimeout, auf die Antwort des Lesedienstes
  DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Lesezugriffs automatisch zu wiederholen.
- ulProcT:
  DMSAPI_NO_TIMEOUT kein timeout

Wert in Millisekunden
DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

- ulRecVarLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt.
- lpRecVar: beim synchronen Lesen die Struktur der gelesenen Variablenliste mit den aktuellen Werten (Null bei Asynchron-Betrieb).

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_VARMODE	Der aufgerufene Dienst stimmt mit dem beim Kreieren übergebenen Variablenlistentyp nicht überein.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_RESOURCE	Lesedienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_-FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.5 DMSAPI_VLReadCycle

SYNTAX

```
DMS_RC DMSAPI_VLReadCycle(
                DMS_HANDLE      DmsHandle           /*  VarListHandle */;
                DMS_UINT32      ulCycleTime         /* Zykluszeit in ms */,
                DMS_INT16       nCBId               /* CallbackId */,
                DMS_INT16       nSyncFlag          /* Synchron flag */,
                DMS_UINT32        ulProcT           /*ProzedurTimeout */,
                DMS_UINT32      ulRecVarLen        /* Grösse des Speichers
                                                    auf den Pointer referenziert */,
                DMS_REC_VARLIST_DATA   *lpRecVar    /* RecStruct der VL */
)
```

Diese Prozedur führt zu einer gefüllten Variablenliste den zyklischen Lesedienst aus. Auf diese Anfrage gibt es eine Antwort und zyklische Variablenlistennachrichten. Der zyklische Lesedienst wird über die Funktion DMSAPI_StopCycleVar gestoppt. Nach dem Stoppen kann die Variablenliste über die Lösch- und Zufügprozeduren verändert werden und neu gelesen werden bzw. komplett gelöscht werden.

Handelt es sich um eine gestoppte Variablenliste, wird der neue zyklische Lesezugriff erst gestartet, wenn die Antwort des vorherigen Stoppens eingetroffen ist. Wird die Variablenliste von der Applikation gestoppt bevor der zyklische Dienst ausgeführt wurde, wird dieser storniert.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- ulCycleTime: Zykluszeit in Millisekunden, die angegebene Zykluszeit wird auf die nächste durch 200 dividierbare Zahl gerundet.

- nCBId: CallbackId, bzw. wird die Variablenliste über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON:Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf die Antwort des Lesedienstes
  DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Lesezugriffs automatisch zu wiederholen
- ulProcT:
  DMSAPI_NO_TIMEOUT kein Timeout
- Wert in Millisekunden
  DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.
- ulRecVarLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt.
- lpRecVar: beim synchronen Lesen die Struktur der gelesenen Variablenliste mit den aktuellen Werten

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_VARMODE	Der aufgerufene Dienst stimmt mit dem beim Kreieren übergebenen Variablenlistentyp nicht überein.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung.

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_RESOURCE	Lesedienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.6 DMSAPI_StopCycle

SYNTAX

DMS_RC DMSAPI_VLStopCycle(

DMS_HANDLEDmsHandle/* VarListHandle */;

)

Die Prozedur stoppt eine laufende zyklische Variabenliste. Nach dem Stoppen einer zyklischen Variablenliste wird für diese Variablenliste nichts mehr empfangen. D.h. wird diese Prozedur aufgerufen, bevor die Applikation die ersten Werte für diese Variablenliste empfangen hat, wird sie die angeforderten Werte nie erhalten und auswerten können. Aus einer gestoppten Variablenliste können Variablen entfernt und hinzugefügt werden. Danach kann diese Variablenliste erneut zyklisch gelesen werden.

Parameter:

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- nSyncFlag
  DMSAPI_SYNCHRON:Die Prozedur wartet, solange wie das angegebene

"ProzedurTimeout", auf die Antwort des Stopdienstes
DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Stopzugriffs automatisch zu wiederholen.

- ulProcT:
  DMSAPI_NO_TIMEOUT kein Timeout

Wert in Millisekunden

DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_VARMODE	Es ist kein zyklischer Lesedienst für diese Variablenliste gestartet.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung => Liste ist automatisch gestoppt.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_RESOURCE	Stoppdienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.7 DMSAPI_VLWrite

SYNTAX

```
DMS_RC DMSAPI_VLWrite(
DMS_HANDLE          DmsHandle                 /*  VarListHandle */;
DMS_INT16           nCBId                     /* CallbackId */,
DMS_INT16           nSyncFlag                 /* Synchron Flag */,
DMS_UINT32          ulProcT                   /* ProzedurTimeout */,
DMS_UINT32          ulRecVarLen               /* Grösse des Speichers
                                              auf den Pointer referenziert */,
DMS_REC_VARLIST_DATA   *lpRecVar              /* RecStruct of VL */
)
```

Diese Prozedur führt zu einer gefüllten Variablenliste den Schreibdienst aus. Auf diese Anfrage gibt es eine Antwort. Nach dem Erhalt der Antwort kann die Variablenliste über die Wertänderungs-, Lösch- und Zufügprozeduren verändert, neu geschrieben bzw. komplett gelöscht werden.

- DmsHandle: Variablenlistenhandle für diese Variablenliste
- nCBId: CallbackId, bzw. wird die Variablenliste über die DMSAPI-Receivefunktion abgeholt, bekommt CBId den Wert DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON:Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf die Antwort des Schreibdienstes
  DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Schreibzugriffs automatisch zu wiederholen.
- ulProcT:
  DMSAPI_NO_TIMEOUT kein Timeout
  Wert in Millisekunden

DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

- ulRecVarLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt.
- lpRecVar: beim synchronen Schreiben die Struktur der gelesenen Variablenliste mit den aktuellen Werten

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_VARMODE	Der aufgerufene Dienst stimmt mit dem beim Kreieren übergebenen Variablenlistentyp nicht überein.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_RESOURCE	Schreibdienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.3.8 DMSAPI_VLDelete

SYNTAX

DMS_RC DMSAPI_VLDelete(

DMS_HANDLE DmsHandle /* VarListHandle */

)

Diese Prozedur löscht eine bestehende Variablenliste. Auch wenn sich die Variablenliste im zyklischen Lesen befindet, wird sie automatisch gestoppt und gelöscht. Nach dem Löschen werden keine Callbackfunktion für diese Variablenliste mehr aufgerufen.

Parameter:

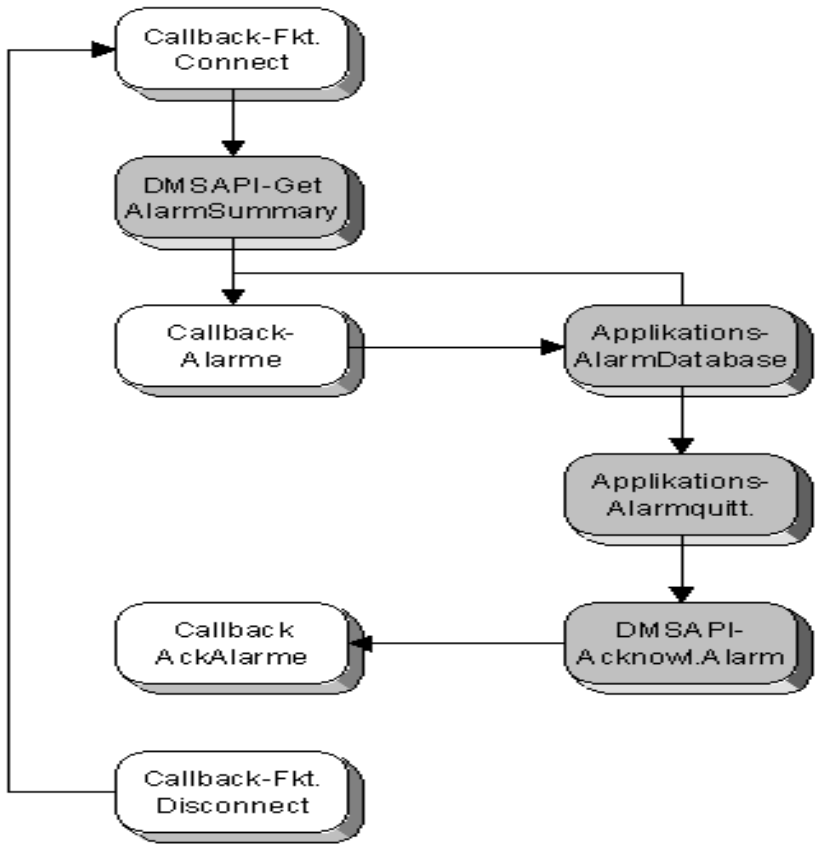
- DmsHandle: Variablenlistenhandle für diese Variablenliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Variablenlistenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4 Alarmmanagement

Starten der Alarmerfassung nach Verbindungsaufbau



Nach dem Starten des GetAlarmSummary wird die Callback-Funktion für die Alarme bei jedem Eintreffen von neuen Alarm automatisch aufgerufen. Die Applikation speichert sich diese Alarme in einer Alarmdatenbank und kann sie bei Bedarf quittieren. Auf jede Alarmquittung gibt es eine Antwort. Die Callback-Funktion für die AcknowledgeAlarme wird aufgerufen.

3.4.1 DMSAPI_GetAlarmSummary

SYNTAX

```
DMS_RC DMSAPI_GetAlarmSummary(
DMS_CONN_HANDLE          ConnHandle      /*  ConnectionHandle */,
                DMS_INT16        nCBId          /*  Callbackid */,
                DMS_INT16        nSyncFlag      /*  Synchron Flag */,
                DMS_UINT32       ulProcT        /* ProzedurTimeout*/,
                DMS_UINT32       ulRecAlaLen    /* Größe des Speichers
                                                   auf den Pointer referenziert */,
DMS_REC_ALARMLIST_DATA*lpAlarmRec   /* Pointer auf AlarmListStruct */
)
```

Nach einem GetAlarmSummary werden alle auf der Prozessstation liegenden und alle ab diesem Zeitpunkt anfallenden Alarme an den Client gesendet. Nach einem Verbindungsabbruch muss dieser Dienst neu angefordert werden.

Parameter:

- Connhandle: ConnectionHandle für diese Ressource
- nCBId: CallbackId, bzw. werden die Alarme über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- nSyncFlag
  DMSAPI_SYNCHRON:Die Prozedur wartet,solange wie das angegebene "ProzedurTimeout", auf die 1. Antwort des GetAlarmSummarydienstes.
  DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Schreibzugriffs automatisch zu wiederholen.

- ulProcT:
  DMSAPI_NO_TIMEOUT kein Timeout
  Wert in Millisekunden
  DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.
- ulRecAlaLen: wird nur bei Benutzung des Synchronflags DMSAPI_SYNCHRON benutzt und enthält die Länge des folgenden Speichers.
- lpAlarmRec: beim synchronen Empfangen der Struktur des GetAlarmSummary mit den ersten ca.43 Alarmen

```
typedef struct DMS_REC_ALARMLIST_DATA {
        DMS_ALARM_LIST_TYPE  ListType;
        DMS_INT16 ActAlarmNo;  /* aktuelle Anzahl von Alarmen */
        DMS_REC_ALARM      *lpAlarm;    /* Alarm liste */
} DMS_REC_ALARMLIST_DATA;
```

Das Element ListType kann folgende Werte annehmen:

DMS_ALARM_GAS alte Alarme, die über ein GetAlarmSummary angefordert worden. Es folgen noch alte Alarme.

DMS_ALARM_LAST_GAS Alle alten Alarme, die über GetAlarmSummary angefordert wurden sind angekommen.

DMS_ALARM_EVENTS Liste mit aktuell anfallenden Alarmen

Beim Element lpAlarm handelt es sich um die folgende AlarmListe mit ActAlarmNo Einträgen.

```
typedef struct DMS_REC_ALARM {
        DMS_DT          TransitionTime;          /* Alarmzeit */
        DMS_OBJNO       ObjectId;                /* ObjNummer */
        DMS_WORD16      AlarmIndex;              /* Alarm index */
        DMS_ALARM_TYPE  AlarmType;               /* Alarm Typ */
        DMS_OBJNO       ObjectClass;             /* ObjektKlasse */
```

```
    DMS_ALARM_STATUS CurrAlarmStatus;   /* aktu. AlarmStat */
    DMS_ALARM_STATUS PrevAlarmStatus;   /* alter AlarmSt. */
    DMS_ALARM_PRIO   Priority;          /* Alarmprio */
    DMS_BOOLEAN      NotificationLost;  /* Alarmburst */
    DMS_RC           rc;                /* AlarmFehler */
    DMS_UINT32       ValueSize;         /* Größe A-Wert */
    DMS_VAR_TYPE     AlarmValType;      /* Datentyp */
    DMS_VALUE        *AlarmValue;       /* Alarmwert */
} DMS_REC_ALARM;
```

Der Alarmtyp kann Werte annehmen, über die der Text für die Systemmeldungen indentifiziert werden kann.

Der CurrAlarmStatus und PrevAlarmStatus kann folgende Werte annehmen:

DMS_ALARM_INACT_INACTNACKED	Inaktiv/NichtQuittiert
DMS_ALARM_ACT_ACTNACKED	Aktiv/Aktiv_NichtQuittiert
DMS_ALARM_INACT_INACTACKED	Inaktiv/Inaktiv_Quittiert
DMS_ALARM_ACT_ACTACKED	Aktiv/Quittiert
DMS_ALARM_INACT_ACTNACKED	Inaktiv/Aktiv_NichtQuittiert
DMS_ALARM_AP_DELETED	Alarmpunkt wurde gelöscht

Der AlarmValType nimmt die verschiedenen Werte für die verschiedenen Datentypen an. (siehe Anhang DMS-Variablentypen)

Der AlarmValue ist eine Referenz auf den Wert des Alarmpunktes (siehe Anhang DMS-Variablentypen).

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_RESOURCE	GetAlarmSummary konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.2 DMSAPI_CreateAckAlarmList

DMSAPI_CreateAckAlarmList(ConnHandle, &DmsHandle);

TDurch diese Prozedur wird der Speicher und ein eindeutiger DMS-Handle für eine DMS-Quittierungsalarmliste erzeugt. Nach dem Erzeugen dieser Liste können AlarmQuittungen eingefügt werden.

Gefüllte Qutittierungslisten können über den Acknowledge Alarm an den Server gesendet werden. Nach dem Empfang kann die Quittierungsliste über die Funktion Clear geleert werden und neue Quittierungsmeldungen eingefügt werden.

Speicher und DMS-Handle werden nur über die Funktion DMSAPI_DeleteAckAlarmList gelöscht.

Parameter:

- Connhandle: ConnectionHandle für diese Ressource
- DMS_Handle: Handle dieser Quittierungsliste, über den alle weiteren Operationen auf die Liste gesteuert werden

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Keine Ressourcen(Speicher / DMSHandles), um diese Quittierungsliste zu kreieren
E_DMSAPI_INVALID_CONN_HANDLE	Es wurde kein gültiger Connectionhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.3 DMSAPI_AddAckAlarmByAddr

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

Diese Prozedur fügt zu einer bestehenden Quittierungsliste ein Element hinzu. Das Element wird über den AlarmPunkt und den AlarmStatus beschrieben.

Nachdem eine Quittierungsliste gefüllt wurde, kann der Quittierungsdienst ausgeführt werden. Nachdem die Quittierungen ausgewertet wurden, kann die Liste geleert und erneut verwendet werden.

Parameter:

- Dmshandle: Handle für die Alarmquittierungsliste
- AlarmPoint des zu quittierenden Alarms
- AlarmStatus des zu quittierenden Alarms
- AlarmIndex innerhalb der zurückkommenden RecStruct

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_NO_RESOURCE	Quittierungsliste ist voll. Sie muss erst versendet werden. Danach können die weiteren Alarme quittiert werden.
E_DMSAPI_INVALID_ARG	Übergabeparameter ist fehlerhaft
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Handle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.4 DMSAPI_ClearAckAlarmList

DMSAPI_ClearAckAlarmList(DmsHandle)

Diese Prozedur löscht alle Quittierungen aus einer bestehenden Alarmquittierungsliste. Die Prozedur kann nicht aufgerufen werden, falls gerade eine Quittierung für diese Liste läuft.

Parameter:

- Dmshandle: Handle für diese Alarmquittierungsliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Handle übergeben.
E_DMSAPI_SERVICE_IN_USE	Die Antwort auf die Quittierung für diese Liste ist noch nicht vom Server eingetroffen.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.5 DMSAPI_AckAlarmList

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

Diese Prozedur führt, zu einer gefüllten Alarmquittierungsliste, den Quittierungsdienst durch. Auf diese Anfrage gibt es eine Antwort. Nach Erhalt und Auswertung der Antwort, kann die Liste über die Lösch- und Zufügeprozeduren verändert werden und erneut quittiert werden bzw. komplett gelöscht werden.

Parameter:

- Dmshandle: Handlefür diese Quittierungsliste
- CBId: CallbackId, bzw. wird die Quttierung über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- SyncFlag / ProcTimeOut
  DMSAPI_SYNCHRON:Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf die Antwort der Quittierung.
- DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des Schreibzugriffs automatisch zu wiederholen.
- ProcTimeOut:
  0 kein Timeout
  Wert in Millisekunden
- DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.
- RecStruct: beim synchronen Aufruf die Struktur der gelesenen Quittierungsliste mit den aktuellen Werten

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_RESOURCE	Quittierungsdienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Listenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.6 DMSAPI_DeleteAckAlarmList

DMSAPI_DeleteAckAlarmList(DmsHandle);

Diese Prozedur löscht eine bestehende Alarmquittierungsliste. Wurde die Antwort für den Quittierungsauftrag noch nicht empfangen, wird sie bei Empfang nicht an die Applikation weitergeleitet.

Parameter:

- Dmshandle: Handle für diese Alarmquittierungsliste

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Handle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.4.7 DMSAPI_AckAlarmByList

SYNTAX

```
DMS_RC DMSAPI_ AckAlarmByList(
DMS_CONN_HANDLE     ConnHandle       /*  ConnectionHandle */,
DMS_HANDLE          *lpDmsHandle     /* Identifier fuer Acklist */,
DMS_INT16           nCBId            /*  CallbackId */,
DMS_INT16           ActAlarmNo       /* aktuelle Anzahl von zu
                                        quittierenden Alarmen */,
DMS_REC_ACKALARM *lpAlarmAck         /* Pointer auf AlarmAckStruct */
DMS_INT16           nSyncFlag          /* Synchron Flag */,
DMS_UINT32          ulProcT            /* ProzedurTimeout*/,
);
```

Diese Prozedur führt zu einer übergebenen, von der Applikation selbstgefüllten, Alarmquittierungsliste den Quittierungsdienst durch. Auf diese Anfrage gibt es eine Antwort. Nach Erhalt und Auswertung der Antwort, wird die Liste automatisch gelöscht, d.h. der DmsHandle ist ungültig. Bei der synchronen Antwort auf diese Anfrage werden in die übergebene Liste die FehlerCodes auf die einzelnen Quittierungen kodiert.

Parameter:

- ConnHandle: Connectionhandle für diese Ressource
- lpDmsHandle: Handle für diese Quittierungsliste
- nCBId: CallbackId, bzw. wird die Quttierung über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- ActAlarmNo: : Anzahl der übergebenen zu quittierenden Alarme

lpAlarmAck: ausgefüllte Liste mit Alarmen, die quittiert werden sollen Beim Element lpAlarm handelt es sich um die folgende AlarmListe mit ActAlarmNo Einträgen.

```
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;
```

Der AlarmStatus kann folgende Werte annehmen:

DMS_ALARM_INACT_INACTNACKED	Inaktiv/NichtQuittiert
DMS_ALARM_ACT_ACTNACKED	Aktiv/Aktiv_NichtQuittiert
DMS_ALARM_INACT_INACTACKED	Inaktiv/Inaktiv_Quittiert
DMS_ALARM_ACT_ACTACKED	Aktiv/Quittiert
DMS_ALARM_INACT_ACTNACKED	Inaktiv/Aktiv_NichtQuittiert
DMS_ALARM_AP_DELETED	Alarmpunkt wurde gelöscht

- nSyncFlag
  DMSAPI_SYNCHRON:Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf die Antwort der Alarmquittierung.

DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden der Alarmquittierung automatisch zu wiederholen.

- ulProcT:

DMSAPI_NO_TIMEOUT kein timeout

Wert in Millisekunden

DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht nicht initialisiert wurde.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der beim Kreieren angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_RESOURCE	Quittierungsdienst konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INVALID_DMS_HANDLE	Es wurde kein gültiger Listenhandle übergeben.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.5 Domainmanagement

Das Domainmangement dient dem Austausch von größeren Konfigurationsdatenmengen zwischen DMS-Client und DMS-Server. Diese Datenmengen werden Domains genannt. Ein Client kann folgende Domaindienste auf dem Server anfordern:

- Hinunterladen von Domains auf den Server
- Heraufladen von Domains vom Server
- Löschen von Domains auf dem Server

Dieses Domainmangement kann nur von Freelance Engineering ausgeführt werden. Die Prozeduren sind daher nicht für die DMSAPI-Applikationen freigegeben und beschrieben.

3.6 ProgramInvokation Management

Zur Zeit ist das Management für "ProgramInvokation" nur für Freelance Engineering implementiert und freigegeben. Das Namensmanagement der DMSAPI-Applikation besitzt zur Zeit keine Konfigurationsinformation zur Umsetzung der Tasknamen auf der Prozessstation zu DMS-Adresse.

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)

Die Prozedur führt auf einer DMS-Serverstation einen "StartProgramInvokation"-Dienst aus. Entweder wird die "ProgramInvokation" über den Namen identifiziert oder über Objektnummer und Verbindung.

Parameter:

- Connhandle: ConnectionHandle für diese Ressource
- CBId: CallbackId, bzw. werden die Alarme über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- PIName: Name des PI-Objektes, das gestartet werden soll, das DMSNamemangement muss aktiviert sein
- ObjNo: Objektnummer des PI-Objektes, das gestartet werden soll
- PILen: Länge der PI-Parameter
- PI-Parameters: Parameter, die an die PI übergeben werden
- Dmshandle: Handle für diese Quittierungsliste
- CBId: CallbackId, bzw. wird die Quttierung über die DMSAPI-Receivefunktion abgeholt bekommt CBId den Wert DMS_NO_CALLBACK
- SyncFlag / ProcTimeOut
  DMSAPI_SYNCHRON: Die Prozedur wartet, solange wie das angegebene "ProzedurTimeout", auf die Antwort des PI-Startens.
  DMSAPI_ASYNCHRON: Es wird nicht auf die Antwort gewartet. Das Timeout wird benutzt um bei Ressourcenmangel das Senden des PIStartzugriffs automatisch zu wiederholen.
- ProcTimeOut:
  0 kein Timeout
  Wert in Millisekunden
- DMSAPI_WAIT_FOREVER: Prozedur kehrt erst zurück, wenn der Auftrag ausgeführt ist, bzw. er nicht ausgeführt werden kann.
- RecStruct: Beim synchronen Aufruf der Prozedur befindet sich darin die Struktur der PI-Antwort mit den aktuellen Werten

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_TIMEOUT	Der aufgerufene Dienst wurde ausgeführt, die synchron angeforderte Antwort wurde noch nicht empfangen. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_-FOREVER angegeben wurde.
E_DMSAPI_NO_CONNECTION	Zu der angegebenen Ressource besteht zur Zeit keine Verbindung.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_INVALID_CONF	Zu dem angegebenen PI-Objekt gibt es keine Information im Namensmagement
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Receivebuffer ist zu klein, nur im synchron Fall möglich.
E_DMSAPI_NO_RESOURCE	StartProgramInvokation konnte zur Zeit nicht ausgeführt werden. Es werden mehr Dienste angefordert, als der Server innerhalb eines Zeitintervalls bearbeiten kann. Dieser Fehler kann nicht auftreten, wenn als Timeout DMSAPI_WAIT_FOREVER angegeben wurde.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

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 Empfangen/Dekodieren von Daten

Applikationen haben im DMS-API drei Möglichkeiten um Nachrichten von der Serverstation zu empfangen:

- In der Prozedur, die die Anfrage stellt, kann synchron auf die Antwort gewartet werden. Hierbei stellt die Applikation den Speicher zur Verfügung in den die Nachricht kodiert wird. Die Nachrichten für die zyklischen Variablenlisten müssen weiter empfangen werden.
- Die Applikation kann die Antworten über die DMSAPI_Receive-Funktion aktiv abholen. Hierbei stellt die Applikation den Speicher zur Verfügung in den die Nachricht kodiert wird.
- Die Applikation kann sich über Callback-Funktion über eintreffende Nachrichten informieren lassen. Der Speicher in der die Nachricht kodiert ist, gehört dem DMS-API. Nach dem Beenden der Callback-Funktion ist der Speicher ungültig.

3.7.1 Strukturdefinitionen

Werden Nachrichten von der Serverstation empfangen, wird die Applikation benachrichtigt. Sie erhält Daten für folgende eintreffende Nachrichten.

==>	Verbindungsaufbau/abbau:	Verbindungsstruktur
==>	Variablendienste:	Variablenstruktur
==>	InformationReport:	InformationReportstruktur, die weitergehende Strukturdefinition ist

applikationsabbhängig Freelance definiert die Strukturen für Kurven und Störablaufprotokolle

==> Alarmdienste: Alarmstruktur

==> Alarmquittierungsdienst: Alarmquittierungsstruktur

==> Downloaddienste: Downloadstruktur

==> ProgramInvokationdieste: ProgramInvokationstruktur

==> Versionsänderungen: Versionsstruktur

Im Kapitel DMS-Utilities gibt es für diese Strukturen eine globale Funktion, die zu Debugzwecken den Inhalt dieser Strukturen ausgibt:

DMSAPI_DumpRecData

Datentyp	Komponente	Wertebereich
DMS_RC	rc	
DMS_CONN_HANDLE	ConnHandle: Handle der Verbindung	
DMS_INT32	BuffLen: Länge des folgenden Buffers, die Länge muss bei synchronen Aufrufen bzw. beim synchronen Empfangen von Daten über die DMSAPI-Receivefunktion groß ge-nug sein.	

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 über alle Receivestrukturen	

Verbindungsstruktur

Beim Verbindungsaufbau kann der ReturnCode folgende Werte annehmen:

- E_DMSAPI_OK: Alles in Ordnung
- E_DMSAPI_INVALID_STATION_TYPE: Falscher Stationstyp
- E_DMSAPI_INVALID_STATION_NO: Falsche Stationsnummer
- E_DMSAPI_NO_OS: Kein Betriebssystem

Beim Verbindungsabbbau kann der ReturnCode nur folgenden Wert annehmen:

- E_DMSAPI_ABORT: Verbindung abgebrochen

Datentyp	Komponente	Wertebereich
DMS_INT32	lBoardType: Typ des CPU-Board	DMS_CPU_UNKNOWN DMS_CPU_DCP02 DMS_CPU_DCP10 DMS_CPU_PC DMS_CPU_HK80
DMS_INT32	lOSType: Typ des Betriebssystems	DMS_OSVERSION_EPROM DMS_OS-VERSION_MSR DMS_OSVERSI-ON_GWY

DMS_INT32	lGwySubType Untertyp des Gateways	DMS_SUBTYPE_UNKN_GWY DMS_SUBTYPE_P_GWY DMS_SUBTYPE_DDE_GWY
DMS_UINT32	ulIPAddress	
DMS_INT32	OwnStationNo: Eigene Stationsnummer wird nur benötigt, falls Station als Serverstation fungiert.	
DMS_INT32	lRedFlag	DMS_STATION_PRIMARY DMS_STATION_SECONDARY

Variablenstruktur

Die Variablenstruktur wird mit folgenden Returncodes aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung
- E_DMSAPI_ABORT: Verbindung abgebrochen

Datentyp	Komponente	Wertebereich
DMS_HANDLE	D MSHandle	
DMS_INT16	MaxNoOfVar Maximale Anzahl der Variablen	
DMS_INT16	ActNoOfVar Anzahl der belegten Variablen	
DMS_INT16	FreeBytes Anzahl der freien Bytes in Liste	
DMS_VAR_ELEM	DMSVarElem Eine Tabelle mit MaxNoOfVar Einträgen von denen ActNoOfVar gültig sind.	

Die Struktur des DMS_VAR_ELEMS

Datentyp	Komponente	Wertebereich
DMS_OBJPATH	DMS-Adressierung: ObjNumber CmpNumber	
DMS_CHAR	VarName	Variablenname oder NULL
DMS_UINT32	ValueSize	
DMS_VAR_RC	VarRc: ReturnCode für diese 1 Variable	
DMS_VAR_TYPE	VarType: Typ der Variablen	DMS_VAR_TYPE_WORD16 DMS_VAR_TYPE_WORD32
DMS_VAR_STATUS	VarStatus	DMS_NOT_VALID DMS_CHANGED DMS_NOT_CHANGED DMS_DELETED
DMS_VALUE	VarValue	Falls die Variable ungültig bzw. gelöscht ist wird hier ein NullPointer übergeben

Infomationreportstruktur

Die Variablenstruktur wird nur mit folgendem Returncode aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung

Datentyp	**Komponente**	**Wertebereich**
DMS_INT32	IRId	MSR supports: - DMSAPI_CP_ID - DMSAPI_SAP_ID
DMS_INT32	Buffer (application-dependent)	

Die Struktur für DMSAPI_CP_ID und DMSAPI_CP_ID

Datentyp	Komponente	Wertebereich
DMS_INT16	NoOfVar Anzahl der Variable-nelemente	
DMS_OBJNO	ObjNumber	Objektnummer des Bau-steins
DMS_VAR_TYPE	VarType[6]	Datentypen der Variablen
DMS_INT16	ContentLen	Länge der Daten
DMS_DT	StartEreigniszeit für Ar-chivierung	
DMS_BYTE	Daten: - NoOfVar Zeitstempel - Daten	DMS_DT

Alarmstruktur

Die Alarmstruktur wird nur mit folgendem Returncode aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung

Datentyp	Komponente	Wertebereich
DMS_INT32	NoOfAlarm Anzahl der Alarmele-mente	
DMS_ALARM_EL EM	DMSAlarmElem	

Die Struktur der DMS_ALARM_ELEM

Datentyp	**Komponente**	**Wertebereich**
DMS_APATH	DMS-Adressierung: • ObjNumber • AlarmNumber	
DMS_DT	Alarm zeit	
DMS_WORD16	Alarm Type	
DMS_ASTATCurr	CurrAlarmStatus	
DMS_ASTATPrev	PrevAlarmStatus	
DMS_BOOLEAN	NotificationLost	
DMS_VALUE	lpMsgValue	

Alarmquittierungsstruktur

Die Alarmstruktur wird mit folgenden Returncodes aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung
- E_DMSAPI_ABORT: Verbindung abgebrochen

Datentyp	**Komponente**	**Wertebereich**
DMS_INT32	NoOfAckAlarm Anzahl der quittierten Alarme	
DMS_ALARM_ACK_ELEM	DMSAlarmElem	

Die Struktur der DMS_ALARM_ELEM

Datentyp	Komponente	Wertebereich
DMS_APATH	DMS-Adressierung: ObjNumber AlarmNumber	
DMS_ASTAT	CurrAlarmStatus	
DMS_RC	rc: ReturnCode für einzelne Quittierung	

Downloadstruktur

Die Downloadstruktur wird mit folgenden Returncodes aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung
- W_DMSAPI_DL_IS_RUNNING: Download läuft noch
- E_DMSAPI_ABORT: Verbindung ist abgebrochen
- E_DMSAPI_DOWNLOAD_ABORT: Server hat Download abgebrochen
- E_DMSAPI_INVALID_CONF: Server konnte Domain nicht installieren/Löschen

Datentyp	Komponente	Wertebereich
DMS_INT32	DMSHandle	
DMS_INT32	Percent: Downloadstatus in Prozent	0 -100

ProgrammInvokationstruktur

Die "Programminvokation"-Struktur wird mit folgenden Returncodes aufgerufen:

- E_DMSAPI_OK: Alles in Ordnung

- E_DMSAPI_ABORT: Verbindung ist abgebrochen
- E_DMSAPI_INVALID_CONF: Server konnte PI nicht finden
- E_DMSAPI_PI_IN_USE: Server konnte PI nicht ausführen, da sie gerade ausgeführt wird

Datentyp	Komponente	Wertebereich
DMS_INT32	DMSHandle	

Versionsstruktur

Die Versionsstruktur wird nur mit folgendem Returncode aufgerufen:

E_DMSAPI_OK: Alles in Ordnung

Bei jeder Umkonfiguration wird jede Callback-Funktion benachrichtigt, für welche Ressource eine Umkonfiguration stattgefunden hat. Nach einer Umkonfigurierung hat das Namemanagement neue Werte. Eventuell sind die laufenden Variablenlisten ungültig, bzw. verweisen auf andere Variablen.

Datentyp	Komponente
DMS_INT32	RessourcenNo
DMS_CHAR	Projektname
DMS_INT32	MajorVersionNo
DMS_INT32	MinorVersionNo
DMS_INT32	PatchVersionNo
DMS_INT32	MaxObjN

3.7.2 Synchrone Dienste

Alle Dienste, die einen Dienst auf dem Server ausführen und damit diesen zum Senden einer Antwort veranlassen, können synchron aufgerufen werden. Der Dienst kommt erst zurück, wenn die Antwort da ist. Der Antwortbuffer wird von der Appli-

kation zur Verfügung gestellt. Die Antwort wird in der Übergabestruktur zurückgegeben.

DMSAPI_Receive (ReceiveTimeOut,&RecStruct);

Alle Dienste, die einen Dienst auf dem Server ausführen und damit diesen zum Senden einer Antwort veranlassen, können asynchron aufgerufen werden. Der Dienst kommt sofort nach dem Senden zurück. Ist für den Dienst keine Callback-Funktion installiert, (CBId ist auf DMS_NO_CALLBACK gesetzt) muss die Antwort über die DMSAPI_Receivefunktion abgeholt werden. Der Antwortbuffer wird von der Applikation zur Verfügung gestellt. Die Antwort wird in der Übergabestruktur zurückgegeben.

3.7.3 DMSAPI_RegisterCltCB

SYNTAX

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

Diese Prozedur registriert eine Callbackfunktion mit einer bestimmten CallbackId. Die registrierte Callbackfunktion wird beim Empfangen von Daten aufgerufen. Beim Verbindungsaufbau und Abbau werden alle registrierten Callback-Funktionen aufgerufen.

Das Registrieren der verschiedenen Callback-Funktionen sollte / muss vor dem Aufruf des DMSAPI_Init stattfinden. Direkt nach dem Initialisieren ist das DMS aktiv und kann ab dem Zeitpunkt mit Clientstationen verbunden werden.

Diese Verbindungen werden den Applikationen auch über die Verbindungsstruktur in den Callback-Funktionen angezeigt.

Parameter:

- nCBId: CallbackId, für welche die folgende Prozedur installiert wird.

- CallBackProc: Callbackfunktion, die beim Empfangen von Daten für die CBID aufgerufen wird. Wird als Parameter NULL übergeben, wird die Callback-Funktion deinstalliert.

typedef DMS_RC (* DMS_REC_DATA_PROC) (DMS_REC_-DATA *DmsRec);

Mögliche Returnwerte:

E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_DUPLICATE_CALLBACK	Für die angegebene CallbackId ist schon eine Funktion installiert.
E_DMSAPI_NO_RESOURCE	Die maximale Anzahl der Callbackfuntkionen ist registriert.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

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

Diese Prozedur registriert eine Callbackfunktion und gibt eine freie CallbackId zurück. Die registrierte Callbackfunktion wird beim Empfangen von Daten aufgerufen. Beim Verbindungsaufbau und Abbau werden alle registrierten Callback-Funktionen aufgerufen.

Das Registrieren der verschiedenen Callback-Funktionen sollte / muss vor dem Aufruf des DMSAPI_Init stattfinden. Direkt nach dem Initialisieren ist das DMS aktiv und kann ab dem Zeitpunkt mit Clientstationen verbunden werden.

Diese Verbindungen werden den Applikationen auch über die Verbindungsstruktur in den Callback-Funktionen angezeigt.

Parameter:

- CBId: CallbackIdentifikation, für welche die folgende Prozedur installiert wird.

- Fnc: Callbackfunktion, die beim Empfangen von Daten für die CBId aufgerufen wird. Wird als Parameter NULL übergeben, wird die Callback-Funktion deinstalliert.

Mögliche Returnwerte:

E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CALLBACK	Die angegebene Callback-Funktion ist nicht installiert.
E_DMSAPI_DUPLICATE_CALLBACK	Für die angegebene CallbackId ist schon eine Funktion installiert.
E_DMSAPI_NO_RESOURCE	Die maximale Anzahl der Callback-funktionen ist registriert.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

3.7.4 Callback function (&RecStruct)

Wird der DMS-Dienst für eine Callback-Funktion aufgerufen, wird die Antwort an die Callback-Funktion übergeben. Nach Beendigung der Callbackfunktion ist der Datenbereich ungültig. Gibt die Callbackfunktion einen ReturnCode ungleich Null zurück wird die Verbindung eingerissen und neu aufgebaut.

In Multitasking (bzw. Multithreading)- Umgebungen kann die Callback-Funktion aus verschiedenen Task-Kontexten "gleichzeitig" aufgerufen werden. Hier kann in der Funktion auch auf das Eintreffen von Ereignissen gewartet werden. Dieses Ereignis darf aber nicht der Empfang von weiteren Daten auf dieser Verbindung sein. Nur die anderen Verbindungen werden weiter bedient.

Bei Verbindungsauf- und abbauen werden immer alle Callback-Funktionen aufgerufen. Auch die Verbindung zu Freelance Engineering wird dann angezeigt.

Bei jeder Konfigurationsänderung werden ebenso alle CallbackFunktionen aufgerufen. Hier muss die Applikation selber entscheiden, was getan werden muss. Z.B. welche Dienste gingen gerade schief, weil die Konfiguration nicht gestimmt hat und können nun wiederholt werden. Oder auch welche Variablen müssen neu gelesen werden, weil sich die internen Freelance Adressinformationen geändert haben. Konfigurationsänderungen können durch Sperren des Objektverzeichnisses verhindert

werden. In diesem Fall meldet Freelance Engineering, dass sich die Downloads nicht durchführen lassen.

4 Namensverwaltung

Das Namemanagement steht nur auf Stationen zur Verfügung, die in Freelance Engineering als Gateway konfiguriert und geladen wurden. Das Empfangen, der von Freelance Engineering zu ladenden Dateien (Domains), geschieht über die Serverprozeduren des Domainmangements. Die Domains sind binärkodiert.

Das Namemangement gilt nur für eine Ressource, d.h. wird das DMSAPI für mehrere Ressourcen gleichzeitig betrieben haben die Ressourcen unterschiedliche Adressräume.

z.B:

Resource 1 verwaltet Projekt1
Resource 2 verwaltet Projekt2

Die Umkonfiguration von Freelance Engineering kann verhindert werden, wenn das Objektverzeichnis gesperrt wird. Hier gibt es die Funktionen: DMSAPI_LockOV und DMSAPI_UnlockOV.

Das Namemanagement verfügt über folgende Informationen:

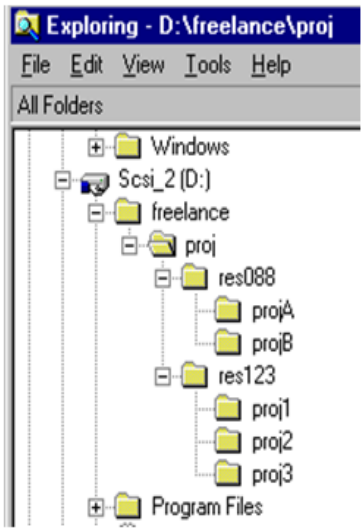
- Versionsinformation: Freelance Engineering überprüft, ob die Version des angeschlossen Gateways mit der konfigurierten Version übereinstimmt.
- Stationsinformation über die IPAdressen, Versionen, u.a. aller Stationen eines Freelance projektes
- VariablenInformation über alle Variablen, für die in Freelance Engineering ein Lese- oder Schreibzugriff konfiguriert wurde
- MSR-Stelleninformationen über alle MSR-Stellen, für die in Freelance Engineering ein Lese- oder Schreibzugriff konfiguriert wurde
- Informationen über Freelance Objektklassen, mit allen ComponentenNamen für die adressierbaren Variablen eines Bausteins.

4.1 Dateiverzeichnis

Vor dem Initialisieren des DMSAPI kann die Applikation ein Gatewayverzeichnis bestimmen unter dem ein Directorybaum angelegt wird. Unter dem Verzeichnis wird für jede initalisierte Ressource ein eigenes Directory angelegt. Dieses Directory bekommt den Namen Res<OwnResNo>. In diesem Verzeichnis wird eine Datei namens "Proj.dom" abgelegt, in dem das zuletzt für diese Resource benutzte Projekt steht. Unter dem Verzeichnis wird für jedes Projekt ein eigenes Directory mit dem Namen des Projektes angelegt. In diesem Verzeichnis werden durch den Download von Freelance Engineering folgende Dateien abgelegt:

- vers.dom:Datei mit den Informationen über die eigene Projektversion
- ov.dom:Datei mit den Informationen über die Größe des Objektverzeichnis und den Ladezustand der einzelnen Objekte
- stat.dom:Datei mit den Informationen über die verschiedenen Ressourcen
- version.dom:Datei mit den Informationen über die Versionsinformation auf den einzel nen Ressourcen
- tag.dom:Datei mit den Informationen über die MSR-Stellen
- var.dom:Datei mit den Informationen über die Variablen
- fb<Num>.dom:Datei mit den Informationen für die einzelnen Objektklassen, wie Funktionsbausteine, SFC und benutzerdefinierten Strukturen

Im Beispiel ist im Windows Explorer ein Gatewayverzeichnis auf Laufwerk D: unter freelance\proj angelegt. Unter diesem Verzeichnis wurden 2 Gateways mit den Ressourcenummern 88 und 123 angelegt. Die Ressource 88 wurde von Freelance Engineering mit den Projekten projA und projB geladen, die Ressource 123 von einem anderen Freelance Engineering mit den Projekten proj1, proi2 und proj3.



ap018gr.bmp

4.1.1 DMSAPI_SetProjectDir

SYNTAX

DMS_RC DMSAPI_SetProjectDir(

DMS_CHAR * szProjectDir

);

Wird das DMS-API auf einem Rechner mit Festplatte betrieben und auf diesem die Gateway-Konfiguration hinuntergeladen, kann das GatewayVerzeichnis durch die Funktion DMSAPI_SetProjectDir gesetzt werden. Durch dieses Setzen werden automatisch alle Freelance Domains unter diesem Verzeichnis installiert. Ändert sich durch den Aufruf dieser Funktion das Projektverzeichnis, wird automatisch die Verbindung zu allen Client-Stationen (in der Regel Freelance Engineering) unterbrochen. Das Projektdirectory wird für alle Ressourcen auf das gleiche Verzeichnis gesetzt. Solange das Projektverzeichnis nicht gesetzt ist kann Freelance Engineering keine Konfiguration laden.

Parameter:

- szProjectDir: gültiges Verzeichnis

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INVALID_DIR	auf Directory kann nicht zugegriffen werden
E_DMSAPI_CONFIGURING	Freelance Engineering lädt gerade Konfiguration
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.1.2 DMSAPI_ChangeProject

SYNTAX

```
DMSAPI_ChangeProject(
        DMS_RES_NOOwnResNo      /* eigene Ressourcenummer */,
        DMS_CHAR *szProjectName         /* Projektname */
)
```

Wird das DMS-API auf einem Rechner mit Festplatte betrieben und auf diesem die Gateway-Konfiguration hinuntergeladen, kann das GatewayVerzeichnis durch die Funktion DMSAPI_ChangeProject gesetzt werden. Durch dieses Setzen werden automatisch alle Freelance Domains unter diesem Verzeichnis installiert. Ändert sich durch den Aufruf dieser Funktion das Projektverzeichnis, wird automatisch die Verbindung zu allen Client-Stationen (in der Regel Freelance Engineering) unterbrochen. Ist das angegebene Projekt nicht vorhanden, liefern die Funktionen des Namemangements nichts zurück. Freelance Engineering setzt beim Initialisieren

und beim Laden der ganzen Station selbständig das Projektverzeichnis auf den Projektnamen von Freelance Engineering.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- szProjektname: Der Projektname muss ein gültiger Dateiname sein

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_CONFIGURING	Freelance Engineering lädt gerade Konfiguration
E_DMSAPI_INVALID_DIR	auf Directory kann nicht zugegriffen werden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.2 Projektinformation

4.2.1 DMSAPI_GetProjectInfo

SYNTAX

```
DMS_RC DMSAPI_GetProjectInfo(
                DMS_RES_NO        OwnResNo        /* Eigene Ressourcen. . */,
                DMS_VERSION_DATA* lpVersionData   /* Versionsdata */
);
```

Es stehen die folgenden Informationen zur Verfügung, wenn das DMSAPI über die Funktion DMSAPI_Init initialisiert wurde. Bei jedem Download von Freelance Engineering erhalten alle angemeldeten Callbackfunktionen diese Informationen automatisch.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- lpVersionData:

```
typedef struct DMS_VERSION_DATA {
        DMS_CHAR        *ProjName;          /* Projektname */
        DMS_WORD16wMajorVersion;            /* MajorversionsNummer*/
        DMS_WORD16 wMinorVersion;           /* MinorversionsNummer */
        DMS_WORD16 wPatchVersion;           /* PatchversionsNummer */
} DMS_VERSION_DATA;
```

Der Projektname kann durch die Funktion DMSAPI_ChangeProject geändert werden. Unterscheidet sich der aktuelle Projektname von dem Projekt das Freelance Engineering berarbeitet und wird von Freelance Engineering aus die Gatewaystation geladen erhält die Gatewaystation automatisch Projektname und folgende Versionsnummern von Freelance Engineering.

Die Majorversionsnummer ändert sich bei jedem "Laden der ganzen Station" von Freelance Engineering. Der alte Wert wird inkrementiert.

Die Minorversionsnummer ändert sich mit jedem einzelnen Objekt, das von Freelance Engineering hinuntergeladen wird. Der alte Wert wird inkrementiert.

Die Patchversionsnummer ändert sich auf dem Gateway nicht. Sie bleibt konstant auf 0.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.3 Sperren des "Namemanagement"

4.3.1 DMSAPI_LockOV

SYNTAX

DMS_RC DMSAPI_LockOV(

DMS_RES_NO OwnResNo /* Eigene Ressourcen. */

);

Soll über einen bestimmten Zeitraum eine Umkonfiguration von Freelance Engineering verhindert werden, kann diese über das Sperren des Objektverzeichnisses verhindert werden. Freelance Engineering zeigt dann an, dass auf dieses Gateway keine Konfiguration geladen werden kann.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_ALREADY_DONE	Objektverzeichnis ist schon gesperrt
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.3.2 DMSAPI_UnlockOV

SYNTAX

DMSAPI_UnlockOV

DMS_RES_NO OwnResNo /* Eigene Ressourcen. */

);

Soll nach einer Sperrung des Objektverzeichnisses die Umkonfiguration durch Freelance Engineering wieder zugelassen werden ist diese Prozedur aufzurufen.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_ALREADY_DONE	Sperrung des OV ist schon aufgehoben
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.4 Stationsinformation

Über die ServerProzeduren des Domainmangements werden folgende 2 Binärdomains empfangen:

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
....					

Table 1.

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

Der Zugriff auf diese Stationsinformation wird über folgende 2 Prozeduren gesteuert:

- DMSAPI_GetFirstResourceInfo
- DMSAPI_GetNextResourceInfo

!!!

Bei der Benutzung dieser Funktionen auf Multitaskingbetriebssystemen ist in der Applikation auf entsprechende Verriegelungsmechanismen zu achten. D.h. werden aus 2 Tasks abwechselnd die GetNext-Funktion aufgerufen, bekommen beide Tasks nicht alle Elemente aus der Stationsdomains sondern abwechselnd die nächsten Elemente aus der Domain.

!!!

Nach Umkonfiguration dieser Stationsdomain durch Freelance Engineering muss immer erneute die Prozedur ein GetFirst aufgerufen werden. Die Getnext-Routine liefert sonst einen Fehler zurück.

4.4.1 DMSAPI_GetFirstResourceInfo

SYNTAX

```
DMS_RC DMSAPI_GetFirstResourceInfo(
                DMS_RES_NO          OwnResNo     /* Eigene ResNum */,
                DMS_UINT32          *lpulNoOfRes  /* Anzahl der Ressourcen */,
                DMS_UINT32          ResNameLen     /* max. Länge Ressname */,
                DMS_CHAR            *lpResName   /* Ressname */,
                DMS_NAME_RESOURCE_DATA*lpResInfo     /* RessInfo */
);
```

Die Prozedur gibt die Informationen, über die erste Ressource, innerhalb der Ressourcendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- lpulNoOfRes: Anzahl der vorhandenen Ressourcen wird zurückgegegeben
- ResNameLen: Länge des folgenden Buffers (DMS_MAX_RESNAME_LEN)
- lpResName:Buffer für Ressourcenname
- lpResInfo: Ressourceinformation

```
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;
```

Der ResType kann folgende Werte annehmen:

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

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine Station im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.4.2 DMSAPI_GetNextResourceInfo

SYNTAX

```
DMS_RC DMSAPI_GetNextResourceInfo(
        DMS_RES_NO          OwnResNo      /* Eigene RessNummer */,
        DMS_UINT32          ResNameLen    /* max. Länge Ressname */,
        DMS_CHAR            *lpResName              /* Ressname */,
        DMS_NAME_RESOURCE_DATA*lpResInfo  /* RessInfo */
);
```

Die Prozedur gibt die Informationen, über die weiteren Ressourcen, innerhalb der Ressourcendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ResNameLen: Länge des folgenden Buffers (DMS_MAX_RESNAME_LEN)
- lpResName:Buffer für Ressourcenname
- lpResInfo: Ressourceinformation

```
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;
```

Der ResType kann folgende Werte annehmen:

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

Parameter:

- OwnResourceNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- StationNameLen: Länge des folgenden Buffers
- lpStationName:Buffer für Stationsname
- lpStationInfo: Stationsinformation

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_NO_ACCESS	Funktion: Getfirst wurde nicht aufgerufen
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine weiteren Stationen im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.5 Variableninformation

Über die ServerProzeduren des Domainmangements wird folgende Binärdomain empfangen:

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
...					

Der Zugriff auf diese Variableninformation wird über folgende 2 Prozeduren gesteuert:

- DMSAPI_GetFirstVarInfo
- DMSAPI_GetNextVarInfo

!!!

Bei der Benutzung dieser Funktionen auf Multitaskingbetriebssystemen ist in der Applikation auf entsprechende Verriegelungsmechanismen zu achten. D.h. werden aus 2 Tasks abwechselnd die GetNext-Funktion aufgerufen, bekommen beide Tasks

nicht alle Elemente aus der Variablendomain sondern abwechselnd die nächsten Elemente aus der Domain.

!!!

Nach Umkonfiguration dieser Stationsdomain durch Freelance Engineering muss immer erneut dieProzedur ein GetFirst aufgerufen werden. Die Getnext-Routine liefert sonst einen Fehler zurück.

4.5.1 DMSAPI_GetFirstVarInfo

SYNTAX

```
DMS_RC DMSAPI_GetFirstVarInfo(
        DMS_RES_NO          OwnResNo    /* Eigene RessourceNummer */,
        DMS_UINT32          *lpulNoOfVar    /* Anzahl der Var */,
        DMS_UINT32          VarNameLen      /* max. Laenge Varname */,
        DMS_CHAR            *lpVarName  /* Variablenname */,
        DMS_NAME_VAR_DATA*lpVarInfo     /* VariablenInfo */);
```

Die Prozedur gibt die Informationen, über die erste Variable, innerhalb der Variablendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- lpulNoOfVar: Anzahl der vorhandenen Variablen wird zurückgegeben
- VarNameLen: Länge des folgenden Buffers(DMS_MAX_VARNAME_LEN)
- lpVarName: Buffer für Variablenname
- lpVarInfo: Variableninformation

```
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 kann die folgenden Werte annehmen:DMS_READ_ONLY

DMS_READ_WRITE

VarType kann verschiedenen Werte annehmen. (siehe Anhang DMS-Variablentypen)

Opath ist die DMS-Adressierung auf dem Server:

typedef struct {

DMS_OBJNOObjNo;

DMS_CMPNOCmpNo}

DMS_OBJ_PATH;

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine Variablen im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.5.2 DMSAPI_GetNextVarInfo

SYNTAX

DMS_RC DMSAPI_GetNextVarInfo(

```
DMS_RES_NO          OwnResNo /*Eigene RessourceNummer */,
DMS_UINT32          VarNameLen  /* max. Länge Varname */,
DMS_CHAR                    *lpVarName    /*Variablenname */,
DMS_NAME_VAR_DATA*lpVarInfo      /*VariablenInfo */);
```

Die Prozedur gibt die Informationen, über die weiteren Variablen, innerhalb der Variablendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- VarNameLen: Länge des folgenden Buffers(DMS_MAX_VARNAME_LEN)
- lpVarName: Buffer für Variablenname
- lpVarInfo : Variableninformation

```
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 kann die folgenden Werte annehmen:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

VarType kann verschiedenen Werte annehmen. (siehe Anhang DMS-Variablentypen)

Opath ist die DMS-Adressierung auf dem Server:

```
typedef struct {
```

DMS_OBJNOObjNo;

DMS_CMPNOCmpNo}

DMS_OBJ_PATH;

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_NO_ACCESS	Funktion: Getfirst wurde nicht aufgerufen
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine weiteren Variablen im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.6 MSR-Stelleninformation

Über die ServerProzeduren des Domainmangements wird folgende Binärdomain empfangen:

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
.....					

Der Zugriff auf diese MSR-Stellen-Information wird über folgende 2 Prozeduren gesteuert:

- DMSAPI_GetFirstTagInfo
- DMSAPI_GetNextTagInfo

!!!

Bei der Benutzung dieser Funktionen auf Multitaskingbetriebssystemen ist in der Applikation auf entsprechende Verriegelungsmechanismen zu achten. D.h. werden aus zwei Tasks abwechselnd die GetNext-Funktion aufgerufen, bekommen beide Tasks nicht alle Elemente aus der MSR-Stellendomain sondern abwechselnd die nächsten aus der Domain.

!!!

Nach Umkonfiguration dieser MSR-Stellendomain durch Freelance Engineering muss immer erneut die Prozedur GetFirst aufgerufen werden. Die Getnext-Routine liefert sonst einen Fehler zurück.

DMSAPI_GetFirstTagInfo

SYNTAX

```
DMSAPI_GetFirstTagInfo(
        DMS_RES_NO        OwnResNo    /* Eigene RessourceNummer */,
        DMS_UINT32                *lpulNoOfTag/* Anzahl der Tags */,
        DMS_UINT32              TagNameLen/* max. Länge Tagname */,
        DMS_CHAR                 *lpTagName   /* MSR-Stellenname */,
        DMS_NAME_TAG_DATA*lpTagInfo     /* Taginfo*/
);
```

Die Prozedur gibt die Informationen, über die erste MSR-Stelle, innerhalb der MSR-Stellendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- lpulNoOfTag : Anzahl der vorhandenen MSR-Stellen wird zurückgegeben
- TagNameLen: Länge des folgenden Buffers(DMS_MAX_TAGNAME_LEN)
- lpTagName:Buffer für MSR-Stellenname

- lpTagInfo : MSR-Stelleninformation

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 kann die folgenden Werte annehmen:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine MSR-Stellen im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_GetNextTagInfo

SYNTAX

DMSAPI_GetNextTagInfo(

```
DMS_RES_NO      OwnResNo    /* Eigene RessourceNummer */,
DMS_UINT32      TagNameLen      /* max. Länge Tagname */,
DMS_CHAR            *lpTagName   /* MSR-Stellenname */,
```

```
                DMS_NAME_TAG_DATA*lpTagInfo     /* TagInfo*/
);
```

Die Prozedur gibt die Informationen, über alle weiteren MSR-Stellen, innerhalb der MSR-Stellendomain zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- 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 kann die folgenden Werte annehmen:

```
                DMS_READ_ONLY
                DMS_READ_WRITE
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.

E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_EMPTY_CONF	keine weiteren MSR-Stellen im Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_NO_ACCESS	Funktion: Getfirst wurde nicht aufgerufen
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

DMSAPI_GetTagByAddr

```
CGEXPORT DMS_RC DMSAPI_GetTagByAddr (
                DMS_RES_NO      OwnResNo    /* Eigene RessourceNummer */,
                DMS_RES_NO      ResNo       /* RessNummer */,
                DMS_OBJNO       ObjNo       /* ObjektPfad */,
                DMS_UINT32      TagNameLen  /* max. Länge Tagname */,
                DMS_CHAR        *lpTagName  /* MSR-Stellenname */,
                DMS_NAME_TAG_DATA*lpTagInfo /* Taginfo*/
);
```

Die Prozedur gibt die Informationen über einen "Tag", der über Ressourcenummer und Objektnummer adressiert wird, zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ResNo: RessourcenNummer der ServerStation.
- ObjNo: Objektnummer des gesuchten Objektes
- TagNameLen: Länge des folgenden Buffers(DMS_MAX_TAGNAME_LEN)
- lpTagName:Buffer für MSR-Stellenname
- lpTagInfo : MSR-Stelleninformation

```
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 kann die folgenden Werte annehmen:

DMS_READ_ONLY

DMS_READ_WRITE

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_INVALID_CONF	keine MSR-Stellen für die Adresse im Projekt vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.7 Objektklassen-Stelleninformation

Über die ServerProzeduren des Domainmangements werden mehrere Binärdomains empfangen:

Analogzähler: Objektklasse 267

Variablenname	Zugriff	DatenTyp	ComponentNr
Enable	RW	DIGI_BOOLEAN	1

Eingang	R	DIGI_FLOAT32	2
...			

BinärÜberwacher: Objektklasse 279

Variablenname	Zugriff	DatenTyp	ComponentNr
Enable	R	DIGI_BOOLEAN	1
Eingang	RW	DIGI_BOOLEAN	2
...			

Datenbaustein: Objektklasse 520 (Namen sind vom Anwender definiert)

Variablenname	Zugriff	DatenTyp	ComponentNr
Struct1	RW	Datenbaustein 510	1
Struct2	RW	Datenbaustein 510	n
...			

Datenbaustein: Objektklasse 510 (Namen sind vom Anwender definiert)

Variablenname	Write	DatenTyp	ComponentNr
Elem1	RW	DIGI_BOOLEAN	1
Elem2	RW	DIGI_FLOAT32	2
....			
Elemn	RW	DIGI_INT32	n

Im Beispiel handelt es sich bei dem Datenbaustein 520 um eine mehrstufige Adressierung:

Die mehrstufige Adressierung über Variablennamen lautet dann z.B.:

StructTst/Struct2/Elem2

Die KomponentenNummer berechnet sich dann:

KomponentenNummer von StructTst +

KomponentenNummer von Struct2 +

KomponentenNummer von StrucElem2 .

Es ist eine beliebig tiefe Schachtelung möglich. Rekursion muss ausgeschlossen sein.

- DMSAPI_GetFirstCmpOfObjClass
- DMSAPI_GetNextCmpOfObjClass

!!!

Bei der Benutzung dieser Funktionen auf Multitaskingbetriebssystemen ist in der Applikation auf entsprechende Verriegelungsmechanismen zu achten. D.h. werden aus zwei Tasks abwechselnd die GetNext-Funktion aufgerufen, bekommen beide Tasks nicht alle Elemente aus der Objektlassendomains sondern abwechselnd die nächsten aus der Domain.

!!!

Nach Umkonfiguration dieser Stationsdomain durch Freelance Engineering muss immer ein erneutes GetFirst aufgerufen werden. Die Getnext-Routine liefert einen Fehler zurück.

Die verfügbaren Komponenten aller Bausteine sind im Handbuch "Freelance 200 Zusatzprogramme - Anhang" aufgelistet.

4.7.1 DMSAPI_GetFirstCmpOfObjClass

SYNTAX

```
DMS_RC DMSAPI_GetFirstCmpOfObjClass(
                DMS_RES_NO      OwnResNo /* Eigene RessourceNummer */,
                DMS_OBJNO       ObjClass    /* Objektklasse */,
```

```
                DMS_UINT32    *lpulNoOfCmp       /* Anzahl der Komponenten */,
                DMS_UINT32   CmpNameLen         /* max. Länge Komp.namen */,
                DMS_CHAR      *lpCmpName          /* Komponentenname */,
                DMS_NAME_OBJ_DATA     *lpObjInfo     /* ObjektInfo */
);
```

Die Prozedur gibt die Informationen, über die erste Komponente, innerhalb der angegebenen Objektklasse zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ObjClass: Objektnummer der gesucht Objektklasse
- lpulNoOfCmp : Anzahl der vorhandenen Komponenten wird zurückgegeben
- CmpNameLen: Länge des folgenden Buffers(DMS_MAX_COMPNAME_LEN)
- lpCmpName: Buffer für Komponentenname
- lpObjInfo: Information über die 1. Komponente der Objektklasse

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

dwAccessRights kann die folgenden Werte annehmen:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_EMPTY_CONF	keine Objektklasse dieses Typs vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.7.2 DMSAPI_GetNextCmpOfObjClass

SYNTAX

```
DMS_RC DMSAPI_GetNextCmpOfObjClass(
        DMS_RES_NO      OwnResNo /* Eigene RessourceNummer */,
        DMS_OBJNO       ObjClass    /* Objektklasse */,
        DMS_UINT32      CmpNameLen/* max. Länge Komp.namen */,
        DMS_CHAR            *lpCmpName /* Komponentenname */,
        DMS_NAME_OBJ_DATA*lpObjInfo     /* ObjektInfo */
);
```

Die Prozedur gibt die Informationen, über alle weiteren Komponenten, innerhalb der angegebenen Objektklasse zurück.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ObjClass: Objektnummer der gesucht Objektklasse
- CmpNameLen: Länge des folgenden Buffers(DMS_MAX_COMPNAME_LEN)

- lpCmpName:Buffer für Komponentenname
- lpObjInfo:Information über die 1. Komponente der Objektklasse

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

dwAccessRights kann die folgenden Werte annehmen:

```
DMS_READ_ONLY
DMS_READ_WRITE
```

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_EMPTY_CONF	keine weiteren Informationen über diese Objektklasse vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.8 Adressen-Konvertierung

Zusätzlich zu diesen Grundfunktionen gibt es die Möglichkeit "Variablennamen" in "Freelance -Objektpfad" zu wandeln und umgekehrt.

4.8.1 DMSAPI_GetVarNameByOPath

SYNTAX

```
DMS_RC DMSAPI_GetVarnameByOPath (
        DMS_RES_NO      OwnResNo    /* Eigene RessourceNummer */,
        DMS_RES_NO      ResNo       /* RessNummer */,
        DMS_OBJ_PATH     *lpOPath   /* ObjektPfad */,
        DMS_UINT32      VarNameLen  /* Max. . Länge Varname */,
        DMS_CHAR              *lpVarName   /* VariablenName */);
```

Die Prozedur wandelt einen Objektpfad in einen Variablennamen um.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- ResNo: RessourcenNummer der ServerStation.
- lpOPath: Objektpfad der gesuchten Variablen

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

- VarNameLen: Länge des folgenden Buffers (DMS_MAX_VARNAME_LEN)
- lpVarName:Buffer für Variablenname

Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.

E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_SMALL_RCV_BUFF	Übergebener Buffer ist zu klein.
E_DMSAPI_INVALID_CONF	keine Information über die Variable vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

4.8.2 DMSAPI_GetVarInfoByName

SYNTAX

```
DMS_RC DMSAPI_GetVarInfoByName(
        DMS_RES_NO      OwnResNo    /* Eigene RessourceNummer */,
        DMS_CHAR        *lpVarName  /* VariablenName */,
        DMS_RES_NO      *lpResNo        /* RessNummer */,
        DMS_OBJ_PATH    *lpOPath        /*ObjektPfad*/,
        DMS_VAR_TYPE    *lpVarType      /* DigiTyp */,
DMS_WORD32          * lpAccessRights        /* Access Rights */
);
```

Die Prozedur wandelt einen Variablennamen in einen "Objektpfad" um. Dabei muss es sich bei dem Variblennamen nicht um eine Variable handeln, die über die Funktionen GetFirstVar und GetNextVar erhalten werden, sondern es kann sich auch um eine aus "Tag" und Komponentennamen zusammengesetzte Variable handlen.

Parameter:

- OwnResNo: RessourcenNummer der eigenen Station. Das DMSAPI_Init muss aufgerufen sein.
- lpVarName: Name der gesuchten Variable
- lpResNo: RessourcenNummer der ServerStation.
- lpOPath: Objektpfad der gesuchten Variablen

typedef struct {

DMS_OBJNO ObjNo;

DMS_CMPNOCmpNo;

} DMS_OBJ_PATH;

- lpVarType kann verschiedenen Werte annehmen. (siehe Anhang DMS-Variablentypen)
- lpAccessRights kann die folgenden Werte annehmen:

  DMS_READ_ONLY

  DMS_READ_WRITE

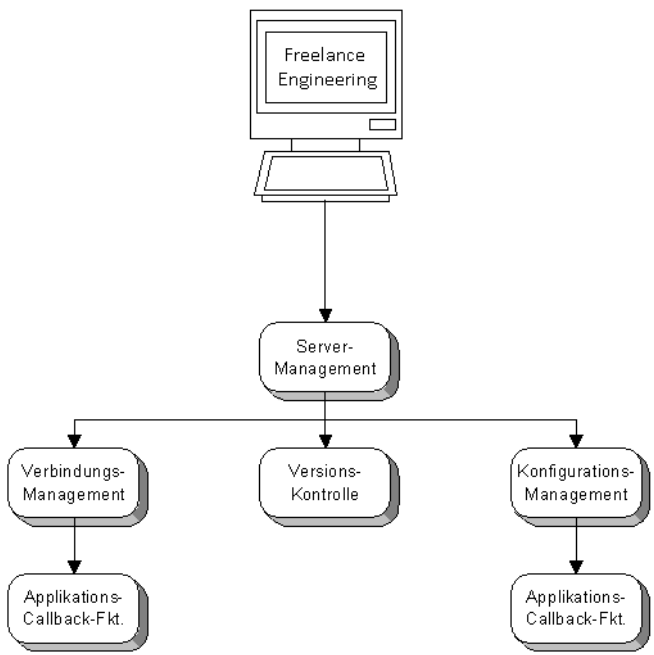
Mögliche Returnwerte:

E_DMSAPI_NOT_INIT	Die Funktion wurde aufgerufen, obwohl die DMS-Schicht für diese Ressourcennummer nicht initialisiert wurde.
E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_NO_CONF	kein Projekt vorhanden
E_DMSAPI_INVALID_CONF	keine Information über die Variable vorhanden
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

5 Server Management

Es gibt vordefinierte Funktionen über die das von Freelance Engineering benötigte ServerManagement ausgeführt werden kann:

- Ablegen der Konfigurationsdomains für das Namemanagement auf Platte
- Start/Stop des DMS`s während einer Umkonfiguration
- Auslesen der Versionsinformation.



6 DMS utilities

6.1 DMSAPI_GetStringByValue

SYNTAX

```
DMS_RC DMSAPI_GetStringByValue(
                DMS_UINT32      ulStrLen        /* Größe des Speichers auf den
                                                   Pointer referenziert*/,
   DMS_CHAR       *lpszString     /* Speicher fuer String */,
   DMS_VAR_TYPEVarType         /* VariablenTyp */,
   DMS_VALUE      *lpvVarValue    /* Variablenwert*/
);
```

Die Prozedur wandelt einen Freelance -Wert in einen druckbaren String um.

Parameter:

- ulStrlen: maximale Länge des Buffers
- lpszString:Buffer für String
- Vartype:Typ des Wertes
- lpvVarValue:Freelance-Wert

Mögliche Returnwerte:

E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_SMALL_RCV_BUFF	Buffer zu klein
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

6.2 DMSAPI_GetValueByString

SYNTAX

```
DMS_RC DMSAPI_GetValueByString(
   DMS_UINT32     ulValLen          /* Größe des Speichers auf den Pointer
                                       referenziert */,
   DMS_VALUE      *lpvVarValue      /* Speicher fuer Variablenwert */,
   DMS_VAR_TYPEVarType      /* VariablenTyp */,
   DMS_CHAR       *lpszString   /* Wert als String */
);
```

Die Prozedur wandelt einen eingelesen String in einen Freelance -Wert um.

Parameter:

- ulValLen: maximale Länge des Buffers
- lpvVarValue:Buffer für Value
- Vartype:Typ des Wertes
- lpvVarValue:Freelance -Wert als String

Mögliche Returnwerte:

E_DMSAPI_INVALID_ARG	Übergabeparameter sind fehlerhaft.
E_DMSAPI_SMALL_RCV_BUFF	Buffer zu klein
E_DMSAPI_INTERNAL_ERROR	Interner Fehler

6.3 DMSAPI_GetVarLen

SYNTAX

```
int DMSAPI_GetVarLen(
        DMS_VAR_TYPE VarType /* VariablenTyp */
);
```

Die Prozedur gibt für einen Freelance -Datentyp die Länge des Speichers zurück, die eine Variable dieses Typs in einer Variablenliste benötigt.

Parameter:

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

Mögliche Returnwerte:

Die Länge des Datentypen in Bytes oder -1 falls ein ungültiger Variablentyp übergeben wird.

6.4 DMSAPI_DumpRecData

SYNTAX

```
void DMSAPI_DumpRecData(
        DMS_REC_DATA * DmsRecData /* */
) ;
```

Die Prozedur gibt (Dump) die Struktur einer Receivestruktur auf StandardOutput.

Parameter:

- RecData: ReceiveData

Anhang A Variablen Typen und Fehler Codes

A.1 DMS-Variablentypen

Im DMSAPI bestehen Variablen immer aus Typ und Wert. Die Variablentypen können folgende Werte annehmen:

Define für Variablentyp	Wert für Variablentyp	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;
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_TY- PE_STRING32	0x10	DMS_STRING32 String32; typedef struct { DMS_WORD16 wMaxStringLen; DMS_CHAR Content[34]; } DMS_STRING32;

DMS_VAR_TY- PE_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-FehlerCodes

Define für Error	Wert für Error	Beschreibung des Fehlers
E_DMSAPI_OK	0x00000000	Kein Fehler
E_DMSAPI_NOT_INIT	0x00000001	Das DMSAPI ist für die angegebene Ressource nicht initialisiert
E_DMSAPI_INVALID_CONF	0x00000002	Keine Konfiguration für angegebene Namen vorhanden
E_DMSAPI_INVALID_ARG	0x00000003	Funktion wurde mit falschem Parameter aufgerufen
E_DMSAPI_SMALL_RCV_BUFF	0x00000004	Der übergebene Buffer ist zu klein
E_DMSAPI_EMPTY_CONF	0x00000005	Für die angegebene Namemanage¬ment-Klasse ist keine Information vorhanden
E_DMSAPI_INTERNAL_ERROR	0x00000006	Interner DMS-Fehler ist aufgetreten. Aus Sicherheitsgründen sollte die Applikation möglichst schnell und "datenschonend" verlassen werden.
E_DMSAPI_ACCESS_ERROR	0x00000007	Auf angegebene Station oder Variable kann nicht zugegriffen werden.
E_DMSAPI_NO_CONF	0x00000008	Für die angegebene Ressource ist keine Konfiguration vorhanden
E_DMSAPI_INVALID_DMS_HANDLE	0x00000009	Der übergebene DMS-Handle ist nicht gültig
E_DMSAPI_INVALID_-CONN_HANDLE	0x0000000a	Der übergebene ConnectionHandle ist nicht gültig
E_DMSAPI_NO_RESOURCE	0x0000000b	Das DMS hat zur Zeit keine Ressourcen. Unter Umständen können Ressourcen nicht zurückgegeben werden, weil Callbackfunktionen die Applikation blockieren.

E_DMSAPI_VARLIST_IN_USE	0x0000000c	Eine Variablenliste kann nicht verändert werden, solange ein Dienst noch nicht abgeschlossen ist.
E_DMSAPI_NO_CALLBACK	0x0000000d	Für den übergebenen CallbackId ist keine Callbackfunktion angege-ben.
E_DMSAPI_DUPLICATE_CALL-BACK	0x0000000e	Unter der angegebene CallbackId wurde schon eine Callback-Funktion angemeldet.
E_DMSAPI_INVALID_INDEX	0x00000000f	In der Variablenliste befindet sich unter dem angebenen Index keine gültige Variable.
E_DMSAPI_INVALID_VARTYPE	0x00000010	Der Wert des Variablentyps ist ungültig
E_DMSAPI_INVALID_VARMODE	0x00000011	Variablenliste wurde für einen Dienst kreiert und soll nun für einen anderen Dienst benutzt wer-den.
E_DMSAPI_NO_CONNECTION	0x00000012	Keine Verbindung zu der angegebenen Station
E_DMSAPI_ALREADY_INIT	0x00000013	Das DMSAPI wurde für diese Station schon initialisiert
E_DMSAPI_MAX_APPLICATION	0x00000014	Das DMSAPI kann nur für eine bestimmt Anzahl Ressourcen initialisiert werden.
E_DMSAPI_MAX_CONNECTION	0x00000015	Das DMSAPI kann nur zu einer bestimmten Anzahl von Ressourcen Verbindungen öffnen.
E_DMSAPI_TIMEOUT	0x00000016	Der Dienst konnte nicht innerhalb des angegebenen Timeouts ausgeführt werden.
E_DMSAPI_INVALID_DIR	0x00000017	Das angebene Verzeichnis existiert nicht.

Anhang B Applikationsschnittstelle Freelance Beispiele

B.1 DMSAPI-Beispiele

Die Beispiele werden bei der DMSAPI - Installation vom Setup in das angegebene Freelance-Verzeichnis

(z.B. c:\Freelance) unter

... \dmsapi\ - incluce

- lib

- samples

angelegt.

B.2 Variablendienste

B.2.1 Einfaches Lesen "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 everytime 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 Freelance Engineering
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,&nVL-
Handle))!=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_A-
SYNC))!=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_SYN-
CHRON,1000,3500,lpRecVar))
         !=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 Zyklisches Lesen "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 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: Configure from Freelance Engineering 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.lpVar-
List;
                                                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> <Var-
Name> .....");
       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,&nVL-
Handle))!=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 Einfaches Schreiben "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 * lpDms-
Rec)
{

/* 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 everytime 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*/
                                                                                suc-
cess++;
                                                                                }
```

```
                                                        }
                                                }
                        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 <iOwnStati-
onNo> ");
                        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, OwnDMSAPICall-
back);
        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
Freelance Engineering 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_GetNextVa-
rInfo %08lx \n", rc);
                                goto _LBL_FNC_XIT;
                        }
                }

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

                StationConnect = 0;
                if ((rc = DMSAPI_ConnectByNo(OwnStationNo, VarInfo.ResNo,
&ConnHandle,
                                                DMSAPI_STD_A-
SYNC)) != 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_SING-
LE_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_VLAddWrite-
VarByName:%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(nVL-
Handle)) != 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)
                                                    {
                                                                NoAns-
wer++;
if ((rc = DMSAPI_VLDelete(nVLHandle)) != E_DMSAPI_OK)

printf("Error VLDelete %lx\n", rc);
if ((rc = DMSAPI_VLCreate(ConnHandle,
                                                                DMSA-
PI_VL_SINGLE_WRITE, &nVLHandle)) != E_DMSAPI_OK)
                                                                {

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

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

                                                                DmsValueFlo-
at32 = (DMS_FLOAT32) 0.0;
                                    AddConst = (DMS_FLOAT32) 1.0;

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

```
                                                                    &Dms-
Value, &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(nVL-
Handle, j,
                                                                DIGI_FLO-
AT32, &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 Alarmdienste "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 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: Configure from Freelance Engineering 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_I-
NACTNACKED ||
                lpRecAL->lpAlarm[i].CurrAlarmStatus==DMS_ALARM_ACT_ACT-
NACKED ||
               lpRecAL->lpAlarm[i].CurrAlarmStatus==DMS_ALARM_INACT_AC-
TNACKED) {
                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,&Dms-
Handle,
                    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_DMSA-
PI_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 Namensverwaltung "name.c"

```
/*
*/
#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 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: Configure from Freelance Engineering 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,&Sta-
tInfo);
   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(OwnStati-
onNo,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,&Stati-
onNo,&Path,&Dtype,
              &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 Setzen der Zeit "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 Redundanzwechsel Primary - Secondary "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 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: Configure from Freelance Engineering 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 */
(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;
    }
(!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_TOG-
GLE);
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);
}
```

Anhang C DMS-API-Dateien

C.1 dmstyp.h

```
/*
COMMENT
*****************************************************************
*
                    DMS-API
Digimatik Message Specification Application Interface
Kommunication 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      Cont-
ent[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      Cont-
ent[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_FO-
REVER, 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_acknowled-
ged */
   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 messa-
ges */
   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
          Kommunication 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 DMS-
TYP.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 Alarm-
ListStruct */);

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

Kommunication 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)
```

Stichwortverzeichnis

—

www.abb.com/freelance
www.abb.com/controlsystems

—

Technische Änderungen der Produkte sowie Änderungen im Inhalt dieses Dokuments behalten wir uns jederzeit ohne Vorankündigung vor. Bei Bestellungen sind die jeweils vereinbarten Beschaffenheiten maßgebend. ABB übernimmt keinerlei Verantwortung für eventuelle Fehler oder Unvollständigkeiten in diesem Dokument.

3BDD012508-111 A