ISA Dialog Manager

C-Schnittstelle - Funktionen

A.06.03.b

Dieses Handbuch beschreibt alle Funktionen der C-Schnittstelle des ISA Dialog Managers. Es enthält die Funktionsdefinitionen mit ihren Parametern und Rückgabewerten.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Deutschland

Darstellungskonventionen

DM wird in diesem Handbuch synonym zu "Dialog Manager" verwendet.

Die Bezeichnung UNIX schließt generell alle unterstützten UNIX-Derivate ein - außer in den explizit angegebenen Fällen.

Dort wo für geläufige englische Fachbegriffe keine gängigen deutschen Übersetzungen existieren, wird zur Vermeidung von Unklarheiten der englische Begriff verwendet.

< >	muss durch einen entsprechenden Wert ersetzt werden
color	Schlüsselwort ("keyword")
.bgc	Attribut
{ }	optional (0 oder einmal)
[ ]	optional (0 oder n-mal)
<A> | <B>	entweder <A> oder <B>

Beschreibungsmodus

Alle Schlüsselwörter sind fett und unterstrichen, z.B.

variable **integer** **function**

Indizierung von Attributen

Syntax für indizierte Attribute:

[I]

[I,J] bzw. [row,column]

Identifikatoren

Identifikatoren müssen mit einem Großbuchstaben oder einem "Unterstrich" ('_') beginnen. Die weiteren Zeichen können Groß-, Kleinbuchstaben, Zahlen oder Unterstriche sein.

Der Bindestrich ('-') ist für die Benennung von Identifikatoren als Zeichen ***nicht*** zugelassen!

Die maximale Länge eines Identifikators beträgt *31* Zeichen.

Beschreibung der zugelassenen Identifikatoren in Backus-Naur-Form

<Identifikator> ::= <erstes Zeichen>{<Zeichen>}

<erstes Zeichen> ::= _ | <Großbuchstabe>

<Zeichen> ::= _ | <Kleinbuchstabe> | <Großbuchstabe> | <Ziffer>

<Ziffer> ::= 1 | 2 | 3 | … 9 | 0

<Kleinbuchstabe> ::= a | b | c | … x | y | z

<Großbuchstabe> ::= A | B | C | … X | Y | Z

Inhalt

1 Einführung

Dieses Handbuch beschreibt die Anwendungsprogramm-Schnittstelle (gewöhnlich als "API-Application Interface" bezeichnet), die vom Dialog Manager (DM) für in C geschriebene Anwendungsprogramme angeboten wird.

Dieses Handbuch beschreibt die Implementierung und den Gebrauch der DM-API zu C-Programmen, die mit dem auf dem jeweiligen System üblichen C-Compiler geschrieben sind.

Neben bestimmten Kontrollfunktionen, die die umfassende Bedienung des DM beeinflussen, bietet das API die grundlegende Funktionalität, die in der Regelsprache des Anwendungsprogramms verfügbar ist.

Diese Handbuch behandelt nur die Funktionen, die vom Dialog Manager bereitgestellt werden, um auf Werte zuzugreifen bzw. Werte zu ändern. Der prinzipielle Aufbau eines C-Programms muss dem Handbuch "C-Schnittstelle Prinzipieller Aufbau und Datentypen" entnommen werden.

2 Schnittstellen-Funktionen

In diesem Kapitel werden die Funktionen der DM-Schnittstelle beschrieben. Nach einer kurzen Zusammenfassung (Kapitel "Übersicht über die Funktionen) werden die verschiedenen Funktionstypen erläutert (Kapitel "Zugriffsfunktionen" und "Fehlerverarbeitung"). Im Kapitel "Funktionen in alphabetischer Reihenfolge" finden Sie eine vollständige Liste mit den detailliert beschriebenen Funktionen.

Beim Beschreiben der Parameter werden folgende Zeichen verwendet:

-> Eingabeparameter

<- Ausgabeparameter

<-> Ein- und Ausgabeparameter

2.1 Übersicht über die Funktionen

Funktionsname	Kurzbeschreibung
AppFinish	Beenden-Routine (verteilte Anwendung).
AppInit	Startroutine (verteilte Anwendung).
AppMain	Hauptroutine der Anwendung.
DM_ApplyFormat	Formatierung eines Strings.
DM_BindCallBacks	Übergeben von Funktionen-Tabelle.
DM_BindFunctions	Übergeben von Funktionen-Tabelle.
DM_BootStrap	Erste Initialisierung von DM.
DM_CallFunction	Aufruf Funktion in beliebigem Anwendungsteil (verteilte Anwendung).
DM_CallMethod	Aufruf einer Methode eines Objekts.
DM_Calloc	Allokierung von Speicher.
DM_CallRule	Aufruf von Regeln mit Parametern.
DM_Control	Aktion auslösen.
DM_ControlEx	Aktion auslösen, erweitert um zusätzliche Aktion.
DM_CreateObject	Erzeugen von Objekten.

Funktionsname	Kurzbeschreibung
DM_DataChanged	Signalisieren, dass sich an einem Datenmodell der Wert eines Attributs geändert hat.
DM_Destroy	Löschen beliebiger DM-Objekte.
DM_DialogPathToID	Umwandlung von externen Objektnamen in interne DM_ID. **Veraltet, ersetzt durch DM_ParsePath.**
DM_DispatchHandler	Benutzerdefinierte Funktion zum Verarbeiten von *XEvents* auf X-Ebene (nur IDM FÜR MOTIF).
DM_DumpState	IDM Zustandsinformationen in Log-/Tracedatei schreiben.
DM_ErrMsgText	Zum Errorcode gehörender Text.
DM_ErrorHandler	Benutzerdefinierte Funktion zur Behandlung von Fehlern die vom Regelinterpreter erkannt werden.
DM_EventLoop	Starten der Dialogabarbeitung.
DM_ExceptionHandler	Benutzerdefinierte Funktion zur Behandlung von „asserts“ des IDM.
DM_Execute	Starten eines anderen Programms.
DM_FatalAppError	Fehlerbehandlung.
DM_FmtDefaultProc	Format.
DM_Free	Freigeben von Speicherplatz.
DM_FreeContent	Löschen von Objektinhalt (z.B. Listbox).
DM_FreeVectorValue	Freigabe von allokiertem Speicher.
DM_GetArgv	Wandelt die Argumente des Programmaufrufs in die intern verwendete Codepage um.
DM_GetContent	Abfrage von Objektinhalt (z.B. bei Listbox).
DM_GetMultiValue	Erfragen mehrerer DM-Attribute in einem Aufruf.
DM_GetToolkitData	Erfragen von Toolkit-Daten.
DM_GetToolkitDataEx	Erfragen von Toolkit-Daten mit zusätzlichen Optionen.
DM_GetValue	Erfragen von DM-Attributen.
DM_GetValueIndex	Erfragen von DM-Attributen (Angabe von Index-Datentyp).

Funktionsname	Kurzbeschreibung
DM_GetVectorValue	Abfrage vektorieller DM-Attribute.
DM_IndexReturn	Sichere Funktionsrückgabe von lokalen Index-Werten an den Aufrufer.
DM_Initialize	Initialisieren von DM.
DM_InitMSW	Parsen der Kommandozeile für Windows
DM_InputHandler	Abfrage zusätzlicher Eingabequellen.
DM_InstallNlsHandler	Holt Text aus externem Textkatalog.
DM_InstallWSINetHandler	Registrierung der benutzerdefinierten Funktionen für den Aufruf von Verschlüsselungssoftware (Java WSI)
DM_LoadDialog	Laden von Dialog in DM.
DM_LoadProfile	Laden benutzerspezifischer Einstellungen.
DM_Malloc	Allokierung von Speicher.
DM_NetHandler	Benutzerdefinierte Funktion zur Manipulation von Daten, die der DDM über das Netzwerk sendet.
DM_OpenBox	Öffnen einer Messagebox oder Dialogbox (Fenster mit *.dialogbox = true*).
DM_ParsePath	Sucht in einem beliebigen Dialog nach einem Objekt und gibt dessen Objekt-ID zurück.
DM_PathToID	Umwandlung von externem Objektnamen in interne DM-ID. **Veraltet, ersetzt durch DM_ParsePath.**
DM_PictureHandler	Benutzerdefinierte Funktion zum Laden von Bildern in vom IDM nicht unterstützten Grafikformaten (Grafik-Handler).
DM_PictureReaderHandler	Registrierung von benutzerdefinierten Funktionen zum Laden von Bildern (Grafik-Handler).
DM_ProposeInputHandlerArgs	Abfragen einer noch nicht vergebenen Nachrichtennummer (nur IDM FÜR WINDOWS).
DM_QueryBox	Öffnen von Messagebox.
DM_QueryError	Erfragen von Fehler.

Funktionsname	Kurzbeschreibung
DM_QueueExtEvent	Stellt ein externes Ereignis, das an ein Objekt geschickt werden soll, in die Ereignis-Warteschlange (Queue).
DM_Realloc	Allokierung von Speicher.
DM_ResetMultiValue	Zurücksetzen von DM-Objekten auf Modell-Wert in einem Aufruf.
DM_ResetValue	Zurücksetzen von DM-Objekten auf Modell- oder Default-Wert.
DM_ResetValueIndex	Zurücksetzen von Tablefieldattributen auf Wert des entsprechenden Defaultattributs.
DM_SaveProfile	Speichert die aktuellen Werte konfigurierbarer Records und Variablen in einer Konfigurationsdatei.
DM_SendEvent	Stellt ein externes Ereignis, das an ein Objekt geschickt werden soll, in die Ereignis-Warteschlange (Queue).
DM_SendMethod	Asynchroner Methodenaufruf.
DM_SetContent	Setzen von Objektinhalt (z.B. Listbox).
DM_SetMultiValue	Verändern von mehreren DM-Objektattributen in einem Aufruf.
DM_SetToolkitData	Setzen von speziellen Fenstersystemdaten.
DM_SetValue	Verändern von DM-Objektattributen.
DM_SetValueIndex	Verändern von DM-Attributen (Angabe von Index-Datentyp).
DM_SetVectorValue	Verändern vektorieller DM-Attribute.
DM_ShutDown	Reguläres Beenden des DM.
DM_StartDialog	Starten der eigentlichen Dialoganwendung.
DM_StopDialog	Beenden eines Dialogs.
DM_StrCreate	Text mit einer vorgegebenen Kodierung erzeugen.
DM_Strdup	Duplizieren eines Strings
DM_StringChange	Änderung eines vom IDM gemanagten Strings.
DM_StringInit	Umwandlung eines Strings in einen vom IDM gemanagten String.
DM_StringReturn	Sichere Funktionsrückgabe von lokalen Strings an den Aufrufer.
DM_TraceMessage	Schreiben von Protokollmeldungen der Anwendung in die DM Protokolldatei.

Funktionsname	Kurzbeschreibung
DM_ValueChange	Ersetzen des Gesamtwerts oder eines Elementwerts in einer Sammlung.
DM_ValueCount	Liefert die Anzahl der Werte, den Indextyp oder den höchsten Indexwert einer Sammlung zurück.
DM_ValueGet	Holt aus einer Sammlung einen einzelnen Elementwert an einem definierten Index.
DM_ValueIndex	Ermitteln des zu einer Position gehörenden Index einer Sammlung.
DM_ValueInit	Umwandeln einer Wertereferenz in eine vom IDM gemanagte lokale oder globale Wertereferenz.
DM_ValueReturn	Sichere Funktionsrückgabe von lokalen **DM_Value**-Werten an den Aufrufer.
DM_WaitForInput	Warten auf ein spezielles Ereignis
YiRegisterUserEventMonitor	Installation von Ereignis-Monitoren (nur IDM FÜR WINDOWS). **Verwendung nicht empfohlen!**

2.2 Initialisierung und Starten des Dialog Managers

Die im Folgenden aufgeführten Funktionen sind zum Initialisieren und Ausführen eines Dialogs notwendig. Folgende Funktionen müssen Sie in Ihrem Hauptprogramm einbauen:

- DM_BindCallBacks
  Diese Funktion übergibt alle Adressen Ihrer Funktionen an den Dialog Manager, so dass diese Funktionen vom Dialog Manager aufgerufen werden können.
- DM_BindFunctions
  Diese Funktion übergibt alle Adressen Ihrer Funktionen an den Dialog Manager, so dass diese Funktionen vom Dialog Manager aufgerufen werden können. Diese Funktion DM_BindFunctions wird dann benötigt, wenn die Funktion in Modulen definiert sind.
- DM_EventLoop
  Diese Funktion startet die Abarbeitung im Dialog Manager. Ohne diesen Aufruf wäre keine Interaktion mit der Benutzeroberfläche möglich.
- DM_Initialize
  Diese Funktion übergibt die Parameter von der Kommandozeile an den Dialog Manager und initialisiert den Dialog Manager.
- DM_LoadDialog

Diese Funktion lädt einen Dialog in den Dialog Manager. Diese Dialogbeschreibung kann eine ASCII-Datei oder eine Binärdatei sein.

» DM_LoadProfile
Diese Funktion lädt die benutzerabhängige Konfigurationsdatei, mit deren Hilfe speziell gekennzeichnete Variablen zu verändern sind.

» DM_StartDialog
Diese Funktion startet den Dialog. Sie initialisiert das Fenstersystem und macht die Fenster sichtbar, die im File als sichtbar definiert sind. Sie führt auch die Dialog-Start-Regel aus.

2.3 Zugriffsfunktionen

Mit Hilfe der folgenden Funktionen können Sie die Objekte des Dialog Managers und deren Attribute manipulieren. Sie sollten immer nur mit diesen Funktionen auf die Objektstrukturen zugreifen.

2.3.1 Zugriff auf Dialog Manager Identifikatoren

Um die einzelnen Objekte identifizieren zu können, werden die Namen benötigt, die sie den Objekten in den Dialogen und Modulen gegeben haben. Mit Hilfe dieses Namens können Sie vom Dialog Manager die interne ID des Objekts erfragen. Mit dieser ID können Sie dann dem Dialog Manager mitteilen, welches Attribut von welchem Objekt Sie erfragen bzw. verändern möchten.

» DM_DialogPathToID **(veraltet, ersetzt durch DM_ParsePath)**
Diese Funktion gibt den Dialog Manager Identifikator eines Objekts zurück. Diese Funktion kann jedoch mehr als einen Dialog verarbeiten.

» DM_ParsePath
Diese Funktion sucht in einem beliebigen Dialog nach einem Objekt und gibt dessen Objekt-ID zurück.

» DM_PathToID **(veraltet, ersetzt durch DM_ParsePath)**
Diese Funktion gibt den Dialog Manager Identifikator des Objekts zurück. Diese ID muss für alle anderen Aufrufe an den Dialog Manager benutzt werden.

2.3.2 Zugriff auf Objektattribute

Um auf irgendein Objektattribut zugreifen zu können, brauchen Sie den Objekt-Identifikator, den Datentyp und den Attribut-Identifikator.

» DM_DataChanged
Mit dieser Funktion kann signalisiert werden, dass sich an einem bestimmten Datenmodell (Model-Komponente) der Wert des angegebenen Attributs (Model-Attribut) geändert hat.

» DM_GetMultiValue
Diese Funktion gibt in einem Aufruf den Wert von mehreren gewünschten Attributen zurück.

» DM_GetValue
Diese Funktion gibt den Wert des gewünschten Attributes von jedem Objekt zurück.

- DM_GetValueIndex
  Diese Funktion gibt den Wert des gewünschten Attributes vom gegebenen Objekt zurück. Im Gegensatz zu DM_GetValue arbeitet diese Funktion mit zwei Indices.
- DM_ResetMultiValue
  Diese Funktion setzt in einem Aufruf mehrere Attribute auf den Wert des Objektmodells oder -default zurück.
- DM_ResetValue
  Diese Funktion setzt das Attribut auf den Wert des Objektmodells oder -default zurück.
- DM_ResetValueIndex
  Diese Funktion setzt das Tablefield-Attribut auf den Wert des entsprechenden Defaultattributs (mit Index 0) zurück.
- DM_SetMultiValue
  Diese Funktion verändert in einem Aufruf mehrere Attribute zu dem übergebenen Wert.
- DM_SetValue
  Diese Funktion verändert das Attribut zu dem übergebenen Wert.
- DM_SetValueIndex
  Diese Funktion verändert das Attribut zu dem übergebenen Wert. Im Gegensatz zu DM_SetValue arbeitet diese Funktion mit zwei Indices.

2.3.3 Behandlung vektorieller Attribute

Diese Funktionen dienen zur Behandlung sog. "vektorieller Attribute", d.h. Attribute, die mehrmals bei einem Objekt auftauchen.

- DM_FreeVectorValue
  Mit dieser Funktion kann allokierter Speicherplatz wieder freigegeben werden.
- DM_GetVectorValue
  Diese Funktion kann mehrere Attributwerte zurückgeben.
- DM_SetVectorValue
  Mit dieser Funktion können mehrere Attributwerte gesetzt werden.

2.3.4 Behandlung komplexer vektorieller Attribute

Um den Inhalt eines Tablefields oder einer Listbox effizient zu verarbeiten, können Sie folgende Funktionen benutzen:

- DM_FreeContent
  Diese Funktion gibt den belegten Speicherplatz für einen Objektinhalt frei, welcher durch einen Aufruf von DM_GetContent zurückgegeben wurde.
- DM_GetContent
  Diese Funktion gibt den gegenwärtigen Inhalt eines Objekts zurück.
- DM_SetContent
  Diese Funktion ersetzt den gegenwärtigen Inhalt des Objekts durch einen neuen Inhalt.

2.3.5 Erzeugen und Zerstören von Objekten

Objekte können mit den beiden folgenden Funktionen dynamisch erzeugt und zerstört werden.

- DM_CreateObject
  Diese Funktion erzeugt ein Objekt einer spezifizierten Klasse und gibt seinen Identifikator zurück. Nachdem das Objekt erfolgreich erzeugt wurde, kann darauf von einer DM-Funktion zugegriffen werden.
- DM_Destroy
  Diese Funktion zerstört jedes Objekt. Nach einem Aufruf dieser Funktion, kann keine DM-Funktion mehr auf diese Objekte zugreifen.

2.3.6 Speicherverwaltungsfunktionen

Mit Hilfe der im folgenden aufgeführten Speicherverwaltungsfunktionen können Speicherbereiche auf portable Art allokiert und wieder freigegeben werden, ohne auf die für das jeweilige Betriebssystem optimalen Funktionen zurückgreifen zu müssen. Werden diese Funktionen in einer Anwendung eingesetzt, müssen Sie auf jeden Fall darauf achten, dass der Speicher, der mit den hier beschriebenen Funktionen allokiert wurde, auch nur mit diesen Funktionen wieder freigegeben werden darf. Prinzipiell können unterschiedliche Methoden der Speicherallokierung gemischt werden; ein einmal allokierter Speicher kann aber nur mit derselben Art von Funktion weiterbearbeitet werden.

- DM_Calloc
  Diese Funktion allokiert Speicher in vorgegebener Größe und vorgegebener Anzahl.
- DM_Free
  Diese Funktion gibt allokierten Speicher wieder frei.
- DM_Malloc
  Diese Funktion allokiert Speicher in einer vorgegebenen Größe.
- DM_Realloc
  Diese Funktion allokiert Speicher in einer vorgegebenen Größe.

2.3.7 Dienstleistungsfunktionen (Utilities)

- DM_CallMethod
  Mit Hilfe dieser Funktion kann eine Methode eines Objekts aus dem Programm heraus aufgerufen werden.
- DM_CallRule
  Mit Hilfe dieser Funktion können benannte Regeln mit Parametern von der Anwendung aufgerufen werden.
- DM_Control
  Durch diese Funktion kann die Anwendung den Bildschirm neu aufbauen, das gegenwärtig benutzte Codeset umschalten oder die Tastatur sperren.
- DM_ControlEx

Mit dieser Funktion können generelle Einstellungen im ISA DIALOG MANAGER geändert oder Aktionen ausgelöst werden.

» DM_DumpState
Mit dieser Funktion werden IDM Zustandsinformationen in die Log- bzw. Tracedatei herausgeschrieben.

» DM_Execute
Diese Funktion startet ein anderes Programm.

» DM_FmtDefaultProc
Besitzt ein editierbarer Text ein Format, so wird diese Funktion für alle anfallenden Aufgaben (wie z.B. Setzen des Formates, Eingabeüberprüfung, Navigation usw.) aufgerufen.

» DM_IndexReturn
Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen Index-Werten (**DM_Index**) an den Aufrufer.

» DM_OpenBox
Diese Funktion öffnet eine Messagebox oder eine Dialogbox (Fenster mit gesetztem Attribut *.dialogbox = true*).

» DM_ProposeInputHandlerArgs
Mit dieser Funktion kann eine noch nicht vergebene Nachrichtennummer für DM_InputHandler abgefragt werden (nur IDM FÜR WINDOWS).

» DM_QueryBox
Mit Hilfe dieser Funktion können Messageboxes geöffnet werden.

» DM_QueueExtEvent
Mit Hilfe dieser Funktion kann ein Ereignis in eine Queue gestellt werden und dadurch eine Regel ausgeführt werden, die einem externen Ereignis zugeordnet ist.

» DM_SaveProfile
Diese Funktion schreibt die aktuellen Werte aller konfigurierbaren ***Record***-Instanzen und **globalen Variablen** eines Dialogs oder Moduls in eine Konfigurationsdatei (profile).

» DM_SendEvent
Mit Hilfe dieser Funktion kann ein Ereignis in eine Queue gestellt werden und dadurch eine Regel ausgeführt werden, die einem externen Ereignis zugeordnet ist.
Im Unterschied zu **DM_QueueExtEvent** wird das externe Ereignis durch eine **DM_Value**-Struktur definiert, wodurch die Verwendung von ***Message***-Ressourcen ermöglicht wird.

» DM_SendMethod
Mit dieser Funktion wird ein Methodenaufruf in die Ereignis-Warteschlange gestellt und asynchron aus der Ereignisschleife heraus ausgeführt.

» DM_TraceMessage
Durch diese Funktion kann die Anwendung Strings in das Tracefile schreiben.

» DM_ValueReturn
Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen **DM_Value**-Werten an den Aufrufer.

Siehe auch

Kapitel „Einbinden benutzerdefinierter Funktionen (Handler)“

2.3.8 Spezielle Funktionen

Diese Funktionen werden normalerweise nicht benutzt, d.h. nur in Ausnahmefällen, da sie automatisch vom Dialog Manager über das Modul **startup.o** bzw. **startup.obj** aufgerufen werden. Nur wenn dieses Modul (durch ein eigenes) ersetzt wird, müssen diese Funktionen aufgerufen werden.

- DM_BootStrap
  Damit kann der Dialog Manager initialisiert werden. **DM_BootStrap** muss die erste Funktion sein, die im Dialog Manager aufgerufen wird.
- DM_InitMSW
  Mit Hilfe dieser Funktion kann die Kommandozeile eines Windows Programms in seine einzelnen Bestandteile zerlegt werden.
- DM_ShutDown
  Mit Hilfe dieser Funktion kann der Dialog Manager regulär beendet werden.

2.3.9 Anbindung des Fenstersystems

Für die Anbindung der Anwendung an das Fenstersystem stehen folgende Funktionen zur Verfügung:

- DM_GetToolkitData
  Diese Funktion gibt die fenstersystemspezifischen Daten eines Attributs zurück.
- DM_SetToolkitData
  Mit Hilfe dieser Funktion können fenstersystemspezifische Daten eines Attributs überschrieben werden.

Siehe auch

Kapitel „Einbinden benutzerdefinierter Funktionen (Handler)“

2.4 Fehlerverarbeitung

Wenn ein Funktionsaufruf an den Dialog Manager FALSE zurückliefert, kann die Anwendung den Fehler mit Hilfe der Funktionen erfragen, die in diesem Kapitel beschrieben werden:

- DM_ErrMsgText
  Diese Funktion gibt den Textstring an einen Fehlercode zurück.
- DM_FatalAppError
  Diese Funktion sollte aufgerufen werden, wenn die Anwendung einen schwerwiegenden Fehler entdeckt und nicht weiterarbeiten kann. Sie beendet jegliches Arbeiten im Dialog Manager.

» DM_QueryError
Diese Funktion gibt die Anzahl von aktuellen Fehlern zusammen mit den Fehlercodes zurück.

Siehe auch

Kapitel „Einbinden benutzerdefinierter Funktionen (Handler)“

2.4.1 Informationen im Fehlercode

Der Fehlercode ist eine Kombination aus drei Informationen:

» dem Grad des Fehlers,

» dem Fehlercode selbst,

» dem Modul, in dem der Fehler auftritt.

Um den Grad eines Fehlers aus der Fehlercode-Variablen zu extrahieren, kann das Makro

```
DM_ExtractSev(ERRNO)
```

benutzt werden.

Die folgenden Fehlergrade sind möglich:

DM_SeveritySuccess	es liegt kein Fehler vor
DM_SeverityWarning	Warnung
DM_SeverityError	Fehler
DM_SeverityFatal	schwerwiegender Fehler, der Dialog Manager kann nicht weiterarbeiten
DM_SeverityProgErr	Programmfehler

Um das Modul zu extrahieren, das den Fehler verursacht hat, kann das Makro

```
DM_ExtractModule(ERRNO)
```

benutzt werden.

Die folgenden Module sind möglich:

DM_ModuleIDM	der Fehler ist innerhalb des Dialog Managers
DM_ModuleUnix	der Fehler taucht bei einem Funktionsaufruf an das Betriebssystem UNIX auf
DM_ModuleMpe	der Fehler taucht bei einem Funktionsaufruf an das Betriebssystem MPE auf
DM_ModuleVms	der Fehler taucht bei einem Funktionsaufruf an das Betriebssystem VMS auf

Um den Fehler selbst zu extrahieren, kann das Makro

```
DM_ExtractErrno(ERRNO)
```

benutzt werden.

Die möglichen Fehlercodes sind in der Datei **IDMuser.h** definiert.

2.5 Bearbeitung von Sammlungen

- DM_ValueChange
  Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung.
- DM_ValueCount
  Liefert die Anzahl der Werte in einer Sammlung (ohne die Standardwerte) zurück. Wenn gewünscht kann auch der Indextyp bzw. der höchste Indexwert zurückgeliefert werden.
- DM_ValueGet
  Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört.
- DM_ValueIndex
  Mit dieser Funktion kann der zu einer Position zugehörige Index einer Sammlung ermittelt werden.
- DM_ValueInit
  Mit dieser Funktion kann eine Wertereferenz in eine vom IDM gemanagte lokale oder globale Wertereferenz umgewandelt werden. Dadurch ist die weitere Manipulation des Wertes durch **DM_Value…()**-Funktionen möglich sowie die Rückgabe als Parameter bzw. Rückgabewert.

2.6 String-Funktionen

- DM_StrCreate
  Mit dieser Funktion kann ein Text in einer vorgegebenen Kodierung erzeugt werden.
- DM_Strdup
  Durch diese Funktion können Strings dupliziert werden.
- DM_StringChange
  Diese Funktion dient zur Änderung eines vom IDM gemanagten Strings.
- DM_StringInit
  Mit dieser Funktion kann ein String in einen vom IDM gemanagten lokalen oder globalen bzw. statischen String umgewandelt werden.
- DM_StringReturn
  Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen Strings an den Aufrufer.

2.7 Einbinden benutzerdefinierter Funktionen (Handler)

Die C-Schnittstelle des IDM bietet die Möglichkeit, eigene „Handler“ einzubinden, um individuell auf Ereignisse oder Fehler reagieren zu können bzw. eigene Funktionalität zu ergänzen. Für diese vom Benutzer zu implementierenden Funktionen gibt der IDM den Datentyp und die Parameter vor und stellt Funktionen zur Verfügung, mit denen die „Handler“ beim IDM angemeldet, abgemeldet, aktiviert

und deaktiviert werden können. Die benutzerdefinierten „Handler“ werden dann vom IDM in den jeweiligen Situationen (Ereignisse, Fehler…) aufgerufen, für die sie vorgesehen sind.

- DM_DispatchHandler
  Einrichten von benutzerdefinierten Funktionen zum Verarbeiten von *XEvents* auf X-Ebene (nur IDM FÜR MOTIF).
- DM_ErrorHandler
  Mit dieser Funktion können Handler eingerichtet werden, die beim Auftreten eines Fehlers, welcher vom Regelinterpreter erkannt wird, aufgerufen werden.
- DM_ExceptionHandler
  Einrichten von benutzerdefinierten Funktionen zur Behandlung von „asserts“ des IDM.
- DM_InputHandler
  Einrichten von benutzerdefinierten Funktionen zum Verarbeiten zusätzlicher Ereignisse (Nachrichten) des Fenstersystems.
- DM_InstallNlsHandler
  Mit dieser Funktion kann eine benutzerdefinierte Funktion für die Verwaltung externer Textkataloge installiert werden.
- DM_InstallWSINetHandler
  Registrierung der benutzerdefinierten Funktionen für den Aufruf von Verschlüsselungssoftware (Java WSI).
- DM_NetHandler
  Installation einer benutzerdefinierten Funktion zur Manipulation der Daten, die vom DDM über das Netzwerk verschickt werden, z.B. zur Verschlüsselung der Daten.
- DM_PictureHandler
  Benutzerdefinierte Funktion (Grafik-Handler, GFX-Handler) zum Laden von Bildern in Grafikformaten, die vom IDM nicht unterstützt werden.
- DM_PictureReaderHandler
  Einrichten von Grafik-Handlern (GFX-Handlern) zum Laden von Bildern in Grafikformaten, die der IDM nicht unterstützt.
- YiRegisterUserEventMonitor **Verwendung nicht empfohlen!**
  Installation von Ereignis-Monitoren, mit denen die Ereignisschleife des IDM unterbrochen werden kann (nur IDM FÜR WINDOWS).

2.8 Behandlung von String-Parametern

Werden über die Interface-Funktionen des Dialog Managers Strings von der Anwendung aus dem Dialog abgeholt, so kopiert der Dialog Manager den entsprechenden String in einen temporären Puffer, aus dem die Anwendung dann den entsprechenden String entnehmen kann. In diesen Puffer darf nicht hineingeschrieben werden, da der Anwendung unbekannt ist, in welcher Größe der Speicherplatz für den String allokiert worden ist. Beim nächsten Aufruf an den Dialog Manager, der als Ergebnis wieder einen String liefert, wird dieser temporäre Puffer überschrieben, es sei denn, diese Funktion wurde mit der Option *DMF_DontFreeLastStrings* aufgerufen. Nur mit dieser Option wird der

Wert des Strings nicht überschrieben. Ist diese Option nicht gesetzt, so kann auf den davor erhaltenen String nicht mehr zugegriffen werden.

Beispiele

```
DM_GetValue (Obj, AT_text, ...);
  // => String wird in temporären Puffer kopiert
DM_GetValue (Obj1, AT_text, ...);
  // => String wird kopiert, alter String ist nicht mehr gültig!

DM_GetValue(Obj, AT_text, ...);
  // => String wird in temporären Puffer kopiert
DM_GetValue(Obj1, AT_text, ...., DMF_DontFreeLastStrings);
  // => Text des Objektes wird kopiert und alter String ist noch gültig
DM_GetValueIndex(Obj2, AT_content, ...);
  // => String wird kopiert, beide davor erfragten Strings werden ungültig.
```

Hinweis

Da Records als Funktionsparameter ebenfalls über **DM_GetValue** gefüllt werden, muss in Funktionen, welche Records als Parameter nutzen, immer die Option *DMF_DontFreeLastStrings* benutzt werden.

2.9 Absicherungen in der Programmschnittstelle

Die Programmschnittstelle bietet Funktionen, die von der Anwendung aufgerufen werden dürfen. Dabei darf nicht jede DM-Funktion zu jedem beliebigen Zeitpunkt aufgerufen werden. So muss zum Beispiel **DM_Initialize** am Anfang aufgerufen werden, erst danach darf **DM_LoadDialog** aufgerufen werden. Für diese Konstellationen gibt es Absicherungen, die überprüfen, ob der Aufruf einer Funktion zu einem bestimmten Zeitpunkt erlaubt ist. Wenn die Funktion nicht aufgerufen werden darf, wird ein Fehlercode gesetzt (*DME_WrongRunState*).

2.9.1 Zustände des Dialog Managers

Der Dialog Manager durchläuft während des Programmablaufs verschiedene Zustände. Für jede DM-Funktion gilt, dass sie nur in bestimmten Zuständen aufgerufen werden darf. Einige DM-Funktionen bewirken, dass der Dialog Manager den Zustand wechselt oder temporär einen anderen Zustand annimmt.

U	uninitialisiert (Anfangszustand)
B	booted
I	initialisiert
M	in der Hauptschleife (main loop)

W Fenstersystem nahe Funktion ist gerade aktiv

F Aufruf einer Format-Funktion

Q innerhalb von **DM_QueueExtEvent**

S gestoppt (Endzustand)

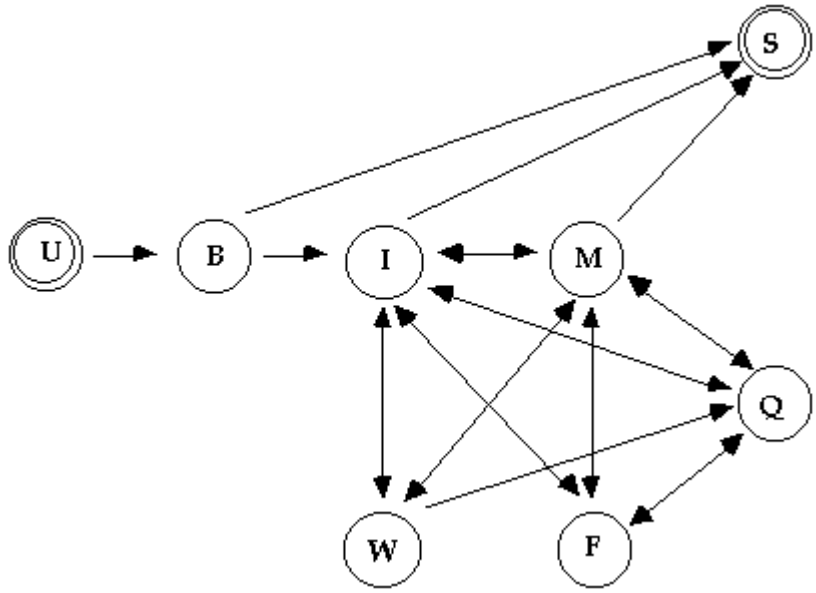

Abbildung 1: *Zustandsübergänge im Dialog Manager*

2.9.2 Übergänge zwischen Zuständen

Die folgenden Funktionen bewirken einen Übergang von einem Zustand in einen anderen:

Funktion	zuvor	während	danach
DM_BootStrap	U	–	U, B
DM_EventLoop	I	M	I
DM_Initialize	B	–	B, I
DM_QueueExtEvent	I, M, W, F	Q	I, M, W, F (wie zuvor)
DM_SendEvent	I, M, W, F	Q	I, M, W, F (wie zuvor)
DM_SendMethod	I, M, W, F	Q	I, M, W, F (wie zuvor)
DM_ShutDown	B, I, M	–	S

2.9.3 Erlaubte Zustände für Funktionen

Der Aufruf der folgenden Funktionen ist nur in den jeweils markierten Zuständen erlaubt.

Funktion	U	B	I	M	W	F	Q	S
DM_ApplyFormat			X	X				
DM_BindCallBacks			X	X				
DM_BindFunctions			X	X				
DM_BootStrap	X							
DM_CallFunction			X	X				
DM_CallMethod			X	X				
DM_CallRule			X	X				
DM_Control			X	X				
DM_ControlEx			X	X				
DM_CreateObject			X	X				
DM_DataChanged			X	X	X	X		
DM_Destroy			X	X				
DM_DialogPathToID			X	X	X	X		
DM_ErrMsgText		X	X	X	X	X		
DM_EventLoop			X	X				
DM_FmtDefaultProc			X	X		X		
DM_FreeContent			X	X				
DM_FreeVectorValue			X	X				
DM_GetContent			X	X	X			
DM_GetMultiValue			X	X	X			
DM_GetToolkitData			X	X	X			
DM_GetValue			X	X	X			
DM_GetValueIndex			X	X	X			
DM_GetVectorValue			X	X	X			
DM_Initialize		X						

Funktion	U	B	I	M	W	F	Q	S
DM_InputHandler		X	X	X				
DM_InstallNlsHandler			X					
DM_LoadDialog			X	X				
DM_LoadProfile			X	X				
DM_OpenBox			X	X				
DM_ParsePath			X	X	X	X		
DM_PathToID			X	X	X	X		
DM_PictureReaderHandler		X	X	X				
DM_QueryBox			X	X				
DM_QueryError		X	X	X	X	X		
DM_QueueExtEvent			X	X	X	X		
DM_ResetMultiValue			X	X				
DM_ResetValue			X	X				
DM_ResetValueIndex			X	X				
DM_SaveProfile			X	X				
DM_SendEvent			X	X	X	X		
DM_SendMethod			X	X	X	X		
DM_SetContent			X	X				
DM_SetMultiValue			X	X				
DM_SetToolkitData			X	X	X			
DM_SetValue			X	X				
DM_SetValueIndex			X	X				
DM_SetVectorValue			X	X				
DM_ShutDown		X	X	X				
DM_StartDialog			X	X				

Funktion	U	B	I	M	W	F	Q	S
DM_StopDialog			X	X				
DM_TraceMessage		X	X	X	X	X		

Bei den folgenden Funktionen wird keine Überprüfung durchgeführt:

- DM_Calloc
- DM_FatalAppError
- DM_Free
- DM_Malloc
- DM_ProposeInputHandlerArgs
- DM_Realloc
- DM_Strdup
- DM_WaitForInput

2.9.4 SetValue aus Canvas-Funktionen

Die Funktionen

- DM_ResetValue
- DM_ResetValueIndex
- DM_SetValue
- DM_SetValueIndex

sind in Canvas-Funktionen nur zum Ändern von Variablen erlaubt.

3 Funktionen in alphabetischer Reihenfolge

3.1 AppMain

Diese Funktion ist die Hauptfunktion der Anwendung. Diese wird vom DM sofort nach dem Programmstart aufgerufen und kann dann die entsprechenden Aktionen einleiten. Sie erhält dieselben Parameter wie die normale Main-Funktion eines C-Programms.

```
int DML_c DM_CALLBACK AppMain
(
  int argc,
  char far * far *argv
)
```

Parameter

-> int argc

In diesem Parameter wird die Anzahl der Kommandozeilen-Argumente übergeben, die beim Programmstart angegeben worden ist.

-> char far * far * argv

In diesem Parameter werden die auf der Kommandozeile angegebenen Argumente übergeben.

Beispiel

```
int DML_c DM_CALLBACK AppMain (argc, argv)
int argc;
char far *far *argv;
{
    DM_ID dialogID;

    if (DM_Initialize (&argc, argv, 0) == FALSE)
        return (1);
    if (!(dialogID = DM_LoadDialog ("dialogname", 0)))
        return (1);

    if (!DM_BindCallBacks((DM_FuncMap *) 0, 0, dialogID, 0))
        DM_TraceMessage ("Function binding incomplete", 0);

    DM_StartDialog (dialogID, 0);
    DM_EventLoop (0);
    return (0);
}
```

Die Main-Funktion muss durch die Funktionen

- AppInit
- AppFinish

ersetzt werden, wenn eine nicht zum Dialog gelinkte Anwendung in einer verteilten Umgebung gestartet bzw. beendet werden soll.

Siehe auch

Objekt Application

3.2 Hauptprogramm im verteilten Dialog Manager

3.2.1 AppInit

Diese Funktion ist die Main-Funktion einer verteilten DM-Anwendung. Mit Hilfe dieser Funktion wird die Anwendung gestartet. Sie kann dann die notwendigen Initialisierungsschritte durchführen.

```
int DML_c DM_CALLBACK AppInit
(
  DM_ID applID,
  DM_ID dialogID,
  int argc,
  char far * far *argv
)
```

Parameter

-> DM_ID applID

Identifikator der Anwendung, die gestartet wurde.

-> DM_ID dialogID

Identifikator des Dialogs, zu dem die Applikation gehört.

-> int argc

Anzahl der übergebenen Kommandozeilen-Parameter.

-> char far * far *argv

Liste der Kommandozeilen-Parameter.

Rückgabewert

Gibt diese Funktion 0 zurück, bedeutet das, dass die Anwendung erfolgreich initialisiert und gestartet werden konnte.

Ein Rückgabewert ungleich 0 bedeutet, dass die Anwendung nicht erfolgreich initialisiert und gestartet werden konnte und der Rückgabewert den Fehlercode enthält.

Beispiel

```
/*
* setup the function table of functions which should be passed
* to the dialog manager
*/
#define ApplFuncCount
    (sizeof(ApplFuncMap)/ sizeof(ApplFuncMap[0]))

static DM_FuncMap ApplFuncMap[ ] = {
    { "FillListbox",(DM_EntryFunc) FillListbox},
    { "FillContinue",(DM_EntryFunc) FillContinue},
```

```
    { "QueryListbox",(DM_EntryFunc) QueryListbox},
    { "CheckFilename",(DM_EntryFunc) CheckFilename}
};

int DML_c DM_CALLBACK AppInit__4(
(DM_ID appl),
(DM_ID dialog),
(int argc)
(char far * far * argv))
{

    if (!DM_BindFunctions(ApplFuncMap, ApplFuncCount,
        appl, 0, DMF_silent))
     DM_TraceMessage ("There are some functions missing", 0);

    return 0;
}
```

3.2.2 AppFinish

Mit Hilfe dieser Funktion wird die Anwendung beendet. Sie sollte dann die notwendigen Schritte durchführen, damit die Anwendung korrekt beendet werden kann.

```
int DML_c DM_CALLBACK AppFinish
(
  DM_ID applID,
  DM_ID dialogID
)
```

Parameter

-> DM_ID applID

Identifikator der Anwendung, die beendet wurde.

-> DM_ID dialogID

Identifikator des Dialogs, zu dem die Applikation gehört.

Rückgabewert

Gibt diese Funktion 0 zurück, bedeutet das, dass die Anwendung erfolgreich beendet werden konnte. Ein Rückgabewert ungleich 0 bedeutet, dass die Anwendung nicht erfolgreich beendet werden konnte und der Rückgabewert den Fehlercode enthält.

Beispiel

```
int DML_c DM_CALLBACK AppFinish (applID, dialogID)
DM_ID applID;
DM_ID dialogID;
```

```
{
    return 0;
}
```

3.3 DM_ApplyFormat

Mit Hilfe dieser Funktion kann ein String so formatiert werden, wie er mit einem gegebenen Format dargestellt werden würde. Normalerweise sind Formate nur in Eingabefeldern und in Tablefields verfügbar, d.h nur hier kann der Ausgabetext mit Hilfe von definierten Formaten oder Formatfunktionen formatiert werden. Mit Hilfe dieser Funktion können jetzt beliebige Texte formatiert werden.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_String DML_default DM_EXPORT DM_ApplyFormat
(
  DM_ID formatOrFunc,
  DM_String formatString,
  DM_String string,
  DM_Options options
)
```

Parameter

-> DM_ID formatOrFunc

In diesem Parameter wird die ID des Formates oder der Formatfunktion angegeben, die den String formatieren soll. Wenn hier die Null-ID angegeben wird, so muss im nächsten Parameter ein gültiger Formatstring angegeben werden.

-> DM_String formatString

In diesem Parameter kann ein beliebiger gültiger Formatstring angegeben werden, mit dessen Hilfe der String formatiert werden soll. Dieser Parameter wird nur beachtet, wenn der Parameter "formatOrFunc" mit der ID einer Formatfunktion oder der ID 0 angegeben worden ist. Ist also im Parameter "formatOrFunc" die ID eines Formates angegeben, so wird dieser Parameter ignoriert.

-> DM_String string

In diesem Parameter wird der String angegeben, der mit Hilfe des angegebenen Formates formatiert werden soll.

-> DM_Options options

Dieser Parameter wird zur Zeit nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

Die Funktion liefert den auf das angegebene Format formatierten String zurück, also genau der String, der in einem Eingabefeld mit dem angegebenen Format dargestellt werden würde.

Beispiel

Von einer C-Funktion aus soll ein String in einem vorgegebenen Format formatiert werden.

```
void UseDMFormat __0()
{
  DM_String format = "NN.NN.NNNN";
  DM_String string = "12121995";

  DM_TraceMessage("Ergebnis DM_ApplyFormat: %s",DMF_Printf,
    DM_ApplyFormat((DM_ID) 0, format, string, 0));

}
```

In der Trace-Datei erscheint dann der formatierte String wie folgt:

```
"12.12.1995"
```

Siehe auch

Eingebaute Funktion applyformat im Handbuch „Regelsprache“

3.4 DM_BindCallBacks

Nach dem Laden des Dialogs kann mit Hilfe dieser Funktion dem DM eine Tabelle von Funktionen übergeben werden. Mit Hilfe dieser Tabelle ruft der DM dann die im Dialog angegebenen Funktionen auf.

```
DM_Boolean DML_default DM_EXPORT DM_BindCallBacks
(
  DM_FuncMap funcmap,
  DM_UInt funccount,
  DM_ID dialogID,
  DM_Options options
)
```

Parameter

-> DM_FuncMap funcmap

Dieser Parameter ist die Tabelle der Funktionen, die direkt vom DM aus aufgerufen werden können. Der Aufbau dieser Tabelle ist im Kapitel über die Datenstrukturen beschrieben.

-> DM_UInt funccount

Dieser Parameter gibt die Größe der übergebenen Funktionstabelle an.

-> DM_ID dialogID

Dieser Parameter ist die ID des Dialogs, für den diese Funktionstabelle gelten soll. Diese ID haben Sie von der Funktion DM_LoadDialog als Rückgabewert erhalten.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_Silent	Diese Option bedeutet, dass keine Fehlermeldungen an den Benutzer ausgegeben werden sollen. Ansonsten wird bei Fehlermeldungen zwischen „überflüssigen Funktionen“ und „fehlenden Funktionen“ unterschieden.
DMF_Verbose	Diese Option bedeutet, dass Fehlermeldungen an den Benutzer ausgegeben werden sollen.

Beispiel

Dem DM soll eine Tabelle mit drei Funktionen übergeben werden.

Damit die Funktionstabelle mit den Adressen dieser Funktionen initialisiert werden kann, müssen diese Funktionen vor der Definition der Tabelle zumindest deklariert werden.

Deklaration in der Header-Datei

```
DM_boolean DML_c DM_ENTRY ReadZip __((DM_String));
```

```
char* DML_c DM_ENTRY QueryZip __((DM_String));
DM_boolean DML_c DM_ENTRY WriteFile __((DM_String, DM_String,
       DM_String));
```

Definition der Tabelle im C-Programm

```
#define FuncCount (sizeof(FuncMap)/ sizeof(FuncMap[0]))

static DM_FuncMap far FuncMap[ ] = {
{ "ReadZip",(DM_EntryFunc) ReadZip},
{ "QueryZip",(DM_EntryFunc) QueryZip},
{ "WriteFile",(DM_EntryFunc) WriteFile},
};

/* Install table of application functions */

{

  if (!DM_BindCallBacks(FuncMap, FuncCount, dialogID, 0))
    DM_TraceMessage ("There are some functions missing.", 0);
  return 0;
}
```

3.5 DM_BindFunctions

Nach dem Laden des Dialogs kann mit Hilfe dieser Funktion dem DM eine Tabelle von Funktionen übergeben werden. Diese Funktion ist von der Arbeitsweise her sehr ähnlich zu der Funktion **DM_BindCallBacks**, jedoch mit dem Unterschied, dass diese Funktion benutzt werden muss, wenn der zugehörige Dialog modular aufgebaut ist und Funktionen in Modulen definiert worden sind.

Mit Hilfe dieser Funktionen können Funktionen, die in noch nicht geladenen Modulen definiert sind, an den Dialog gebunden werden, so dass nach dem Laden des Moduls diese Funktionen sofort aufgerufen werden können.

```
DM_Boolean DML_default DM_EXPORT DM_BindFunctions
(
  DM_FuncMap funcmap,
  DM_UInt funccount,
  DM_ID objID,
  DM_ID moduleID,
  DM_Options options
)
```

Parameter

-> DM_FuncMap funcmap

Dieser Parameter ist die Tabelle der Funktionen, die direkt vom DM aus aufgerufen werden können. Der Aufbau dieser Tabelle ist im Kapitel über die Datenstrukturen beschrieben. Diese Tabelle kann über die Option **+writefuncmap** automatisch vom Simulationsprogramm generiert werden.

-> DM_UInt funccount

Dieser Parameter gibt die Größe der übergebenen Funktionstabelle an.

-> DM_ID objID

Dieser Parameter ist die ID des Objektes, an das diese Tabelle gebunden werden soll. Diese ID kann entweder ein Dialog, ein Modul oder eine Applikation sein.

-> DM_ID moduleID

Dieser Parameter ist die ID des Moduls, das seine Funktionen sofort aus dieser Tabelle versorgen soll. Diese ID muss nur dann angegeben werden, wenn ein Modul nachgeladen worden ist und dann erst die Funktionen an die übergeordnete Instanz gebunden werden. Im Normalfall ist hier also die ID 0 anzugeben.

-> DM_Options options

Hier sind zur Zeit folgende Werte möglich:

Option	Bedeutung
DMF_ReplaceFunctions	Diese Option bedeutet, dass eine vielleicht vorhandene Funktionstabelle bei dem angegebenen Objekt komplett durch die neue Tabelle ersetzt werden soll. Ist diese Option nicht angegeben, so wird die neue Tabelle am Ende einer bestehenden Tabelle des Objekts angefügt.

Beispiel

Aus einem Modul wird über die Option **+writefuncmap** (siehe auch Kapitel „Startoptionen im Simulationsprogramm“ im Handbuch „Entwicklungsumgebung“) eine C-Datei generiert. Dabei hat die Dialogdatei folgendes Aussehen:

```
module ModFuncDate

application TimeServer
{
  !! Get the current date from the server to synchronize the
  !! clients
  !! example: CurrentDate( Year, Month, Day) returns true
  !! if successful and fills the three variables with the
  !! appropriate values
  function boolean CurrenDate( integer Year output,
    integer Month output, integer Day output);

  !! Get the current time, see function CurrentDate
  function boolean CurrentTime( integer Hour output,
    integer Minute output, integer Second output);
}
```

Das generierte C-Programm sieht wie folgt aus:

```
#include "IDMuser.h"
#include "dateappl.h"

#define FuncCount_TimeServer (sizeof(FuncMap_TimeServer) / sizeof(FuncMap_
TimeServer[0]))

static DM_FuncMap FuncMap_TimeServer[] =
{
/*
**  Get the current date from the server to synchronize the
**  clients
**  example: CurrentDate( Year, Month, Day) returns true
**  if successful and fills the three variables with the
**  appropriate values
*/
  { "CurrenDate", (DM_EntryFunc) CurrenDate },
```

```
/*
**  Get the current time, see function CurrentDate
*/
  { "CurrentTime", (DM_EntryFunc) CurrentTime },
}

DM_Boolean DML_default BindFunctions_TimeServer __3(
    (DM_ID, dialogID),
    (DM_ID, moduleID),
    (DM_Options, options))
{
return (DM_BindFunctions (FuncMap_TimeServer,
    FuncCount_TimeServer,dialogID,moduleID,options))
}
```

3.6 DM_BootStrap

Diese Funktion ist in startup.o/startup.obj enthalten und wird daher normalerweise nicht benutzt.

Mit Hilfe dieser Funktion wird der Dialog Manager intern initialisiert, die Argumente der Kommandozeile gesichert sowie die ersten Aktionen ausgeführt

- -IDMerrfile
- -IDMtracefile

Diese Funktion muss die erste Funktion im Dialog Manager sein, die von der Anwendung aufgerufen wird. Normalerweise erfolgt dieser Aufruf durch das Main-Programm im Dialog Manager, das danach das eigentliche Hauptprogramm der Anwendung **AppMain** aufruft. Aus diesem Grund darf diese Funktion nur dann aufgerufen werden, wenn das **Main** des Dialog Managers durch ein eigenes ersetzt wird.

```
int DML_default DM_EXPORT DM_BootStrap
(
  int *argcp,
  char far * far * far *argvp
)
```

Parameter

<-> int *argcp

In diesem Parameter wird die Adresse der Anzahl der Parameter übergeben. Diese Parameteranzahl kann dann von DM_BootStrap verändert werden, falls dort schon Argumente verarbeitet werden.

<-> char far * far * far *argvp

Dieser Parameter ist ein Pointer auf die Argumente der Kommandozeile. Aus dieser Kommandozeile werden dann alle von DM_BootStrap verarbeiteten Argumente entfernt.

Rückgabewert

0	Beim Initialisieren ist kein Fehler aufgetreten, der Dialog Manager kann ganz normal gestartet werden.
!=0	Beim Initialisieren ist ein Fehler aufgetreten. Das Programm darf nicht fortgesetzt werden.

Beispiel

Auszug aus der Datei "startup.c", über die Dialog Manager Programme gestartet werden:

```
int cdecl main __2(
(int, argc),
(char far * far *, argv))
{
  register int status;
```

```
  static char running = 0;

  if ((status = running++) == 0)
  {
    if ((status = DM_BootStrap(&argc, &argv)) == 0)
    {
      DM_InitOptions(&argc, argv, 0);
      status = AppMain (argc, argv);
    }
  }
  return (status);
}
```

3.7 DM_CallFunction

Um beliebige, dem Dialog Manager bekannte Funktionen in anderen Teilen der Anwendung aufrufen zu können, muss die Funktion DM_CallFunction benutzt werden. Diese Funktion ruft dann die entsprechende Funktion in einem beliebigen Anwendungsteil auf.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_CallFunction
(
  DM_ID funcID,
  DM_UInt argcount,
  DM_Value *argvec,
  DM_Value *result,
  DM_Options options
)
```

Parameter

-> DM_ID funcID

Dieses ist der Identifikator der Funktion, die aufgerufen werden soll.

-> DM_UInt argcount

Dieser Parameter gibt die Anzahl der Parameter an, mit der diese Funktion aufgerufen werden soll. Die Belegung der Parameter kann dann dem entsprechenden Element im Parameter argvec entnommen werden. Dabei muss argcount exakt den Wert haben, die dem aktuellen Belegungsgrad des argvec entspricht. Maximal ist 16 zulässig.

<-> DM_Value *argvec

Dieser Parameter ist ein Array von DM_Value-Werten, die als Parameter für den Funktionsaufruf genommen werden sollen. Die Länge dieses Vektors muss dabei unbedingt dem Wert in argcount entsprechen.

<- DM_Value *result

In diesem Parameter wird der Rückgabewert der Funktion zurückgegeben, falls der Funktionsaufruf durchgeführt werden konnte.

-> DM_Options options

Unbenutzt. Muss 0 sein.

Rückgabewert

TRUE	Funktion wurde erfolgreich aufgerufen.

FALSE	Funktion konnte nicht aufgerufen werden.

Beispiel

Eine Funktion, die im Dialog definiert ist, soll mit einem Integer-Parameter aufgerufen werden.

```
void DML_default DM_ENTRY CALLFUNC __((DM_ID Funktion))
{
  DM_Value argv;
  DM_Value retval;

  argv.type = DT_integer;
  argv.value.integer = 888;
  if (DM_CallFunction (Funktion, 1, &argv, &retval, 0))
    if (retval.type == DT_integer)
        printf(Erhaltener Wert: %ld", retval.value.integer);

}
```

Siehe auch

Objekt Application

3.8 DM_CallMethod

Mit Hilfe dieser Funktion können Sie Methoden der Objekte von der Anwendung aus aufrufen. Dabei wird nicht unterschieden, ob es sich um eine vordefinierte oder eingebaute Methode handelt. Die Belegung der Parameter ist dabei abhängig von der Methode und dem Objekt, bei dem diese Methode aufgerufen werden soll.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_boolean DML_default DM_EXPORT DM_CallMethod
(
  DM_ID object,
  DM_Method method,
  DM_UInt argc,
  DM_Value *argvc,
  DM_Value *retval,
  DM_Options options
)
```

Parameter

-> DM_ID object

Dieser Parameter bezeichnet das Objekt, dessen Methode aufgerufen werden soll.

-> DM_Method method

Dieser Parameter bezeichnet die Methode, die bei dem Objekt aufgerufen werden soll. Abhängig von diesem Wert werden die nachfolgenden Parameter dieser Funktion belegt.

-> DM_UInt argc

In diesem Parameter steht die Anzahl der gültigen Methodenparameter.

-> DM_Value *argv

Array von DM_Value-Strukturen, die die von der aufgerufenen Methode gültigen Parameter enthält.

Die Länge dieses Parameters muss unbedingt der beim Parameter *argc* angegebenen Anzahl entsprechen. Die maximale Länge dieses Arrays ist *16*.

<- DM_Value *retval

Pointer auf eine DM_Value-Struktur, die von dieser Funktion als Rückgabewert benutzt wird.

Abhängig von der aufgerufenen Methode wird dabei ein Element in dieser Struktur gesetzt.

-> DM_Options options

Hier können Sie als eine Option *DMF_DontFreeLastStrings* angeben, um den Speicherplatz der Stringparameter weiter gültig zu halten.

Rückgabewert

TRUE	Methode wurde erfolgreich ausgeführt.
FALSE	Methode wurde nicht erfolgreich ausgeführt.

Die verfügbaren eingebauten Methoden können Sie der „Methodenreferenz“ entnehmen. Die notwendigen Konstanten für die eingebauten Methoden sind in der **IDMuser.h** definiert.

Objekte

Alle verfügbaren Objekte

Beispiel

Entladen eines Tablefields in einer Nachlade-Funktion.

```
void DM_CALLBACK ContFunc (DM_ContentArgs* args)
{
    DM_Value methArgs[2];
    DM_Value retval;
    DM_Value first, last;
    ushort ldStart, ldCount;
    ushort visStart;

    ldStart = args->loadfirst - args->header;
    ldCount = args->loadlast - args->loadfirst + 1;

    visStart = args->visfirst - args->header;

    if (visStart > 20)
    {
        /*
        *  Beim Löschen sind zwei Parameter für die
        *  Methode anzugeben. Es muss der Start- und der
        *  Ende-Index angegeben werden, zwischen denen
        *  gelöscht werden soll. Beide sind vom Datentyp her
        *  integer.
        */
        methArgs[0].type = DT_integer;
        methArgs[1].type = DT_integer;
        methArgs[0].value.integer = args->header + 1;
        methArgs[1].value.integer = visStart - 20;
        DM_CallMethod(args->object, MT_clear, 2, methArgs,
```

```
                &retval, 0);
        }
    }
```

3.9 DM_Calloc

Mit Hilfe dieser Funktion können Sie Speicherplatz allokieren. Diese erfolgt dabei abhängig vom zugrundeliegenden Betriebssystem mit den dort verfügbaren Funktionen. Über diese Funktion allokierter Speicher darf nur mit der Funktion DM_Free freigegeben werden bzw. mit DM_Realloc verändert werden.

```
DM_Pointer DML_default DM_EXPORT DM_Calloc
(
  DM_UInt4 nelem,
  DM_UInt4 elsize
)
```

Parameter

-> DM_UInt4 nelem

Dieser Parameter gibt die Anzahl der Elemente an, die allokiert werden soll.

-> DM_UInt4 elsize

Dieser Parameter gibt die Größe eines zu allokierenden Elements an.

Rückgabewert

Pointer auf den allokierten Speicherbereich. Wenn der Speicher nicht allokiert werden konnte, wird der *NULL*-Pointer zurückgegeben.

Im Gegensatz zu DM_Malloc wird der Speicher auf 0 initialisiert.

Beispiel

Für die DM_Content-Struktur soll für 10 Elemente Speicher allokiert werden.

```
DM_Content *content;

if ((content = DM_Calloc (10, sizeof (DM_Content))
{
    ...
}
```

3.10 DM_CallRule

Mit Hilfe dieser Funktion können benannte Regeln mit Parametern von der Anwendung aufgerufen werden.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_CallRule
(
  DM_ID ruleID,
  DM_ID objectID,
  DM_UInt argc,
  DM_Value *argv,
  DM_Value *retval,
  DM_Options options
)
```

Parameter

-> DM_ID ruleID

Dieser Parameter bezeichnet die Regel, die ausgeführt werden soll.

-> DM_ID objectID

Dieser Parameter entspricht dem `this` in der entsprechenden Regel, d.h. in der Regel wird für `this` das Objekt eingesetzt, das Sie hier angeben.

-> DM_UInt argc

In diesem Parameter wird die Anzahl der Regelparameter angegeben.

-> DM_Value *argv

In diesem Parameter werden die Parameter der Regeln als Feld von DM_Value-Strukturen angegeben. Die Länge dieses Parameters muss unbedingt gleich der beim argc-Parameter angegebenen Anzahl sein. Es sind maximal 16 Parameter erlaubt.

<- DM_Value *retval

Pointer auf eine DM_Value-Struktur, die als Rückgabewert ausgefüllt wird.

-> DM_Options options

Dieser Parameter wird z. Zt. noch nicht benutzt. Muss 0 sein.

Rückgabewert

TRUE	Regel konnte ausgeführt werden.
FALSE	Regel konnte nicht ausgeführt werden.

Beispiel

Aufruf einer Regel im Dialog. Die Regel ist dabei wie folgt definiert:

```
rule integer Callrule (integer I input)
{
    I := I + atoi(Et.content);
    Et.content := itoa(I);
    print I;
    return(I);
}
```

Das zugehörige C-Programm sieht wie folgt aus:

```
void DML_default DM_ENTRY CALLRULE __((DM_ID Regel))
{
  DM_Value argv;
  DM_Value retval;

  argv.type = DT_integer;
  argv.value.integer = 888;
  if (DM_CallRule (Regel, Regel, 1, &argv, &retval, 0))
    if (retval.type == DT_integer)
        printf(Erhaltener Wert: %ld", retval.value.integer);

}
```

3.11 DM_Control

Mit dieser Funktion können generelle Einstellungen im ISA DIALOG MANAGER geändert oder Aktionen ausgelöst werden.

```
DM_boolean DML_default DM_EXPORT DM_Control
(
  DM_ID      objectID,
  DM_UInt    action,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Objekt für das die gewünschte Aktion ausgeführt werden soll.

-> DM_UInt action

Aktion, die der IDM ausführen soll. Dazu sind verschiedene Konstanten in der Include-Datei **IDMuser.h** definiert. Diese Konstanten sind in der Tabelle unten erläutert.

-> DM_Options options

Enthält bei Bedarf ein Argument für die in *action* angegebene Aktion (siehe Tabelle unten).

Rückgabewert

DM_TRUE	Die Aktion wurde erfolgreich ausgeführt.
DM_FALSE	Die Aktion konnte nicht ausgeführt werden.

Die nachfolgende Tabelle zeigt die gültigen Belegungen der einzelnen Parameter und erläutert ihre Bedeutung. Wenn bei der Aktion nichts anderes gesagt wird, ist als *objectID* immer *0* anzugeben.

action	options	Bedeutung
DMF_UpdateScreen	*0*	Es sollen alle internen SetVal-Aufrufe auf den Bildschirm gebracht werden. In diesem Fall muss der erste Parameter mit dem Dialog belegt sein.

action	options	Bedeutung
DMF_UIAutomationMode		Mit dieser Aktion kann die spezielle UI Automation Unterstützung des IDM für seine besonderen Objekte abgeschaltet werden. Weiterhin aktiv bleibt aber die UI Automation Unterstützung von Microsoft für die Standard Controls. Standardmäßig ist die Unterstützung von UI Automation aktiv. Die Umschaltung muss vor dem Aufruf von **DM_Initialize** und nach dem Bootstrapping erfolgen.
	0	Deaktiviert den UI Automation Support des IDM.
	1	Aktiviert den UI Automation Support des IDM.
DMF_PCREBinding	*0*	Deaktiviert die Anbindung an die PCRE-Bibliothek, Reguläre Ausdrücke sind somit nicht mehr möglich.
	1	Anbindung an statisch im Executable vorhandene PCRE-Funktionen (Anbindungsart E).
	2	Nur dynamische Anbindung von PCRE-Bibliotheken relativ zur Applikation (Anbindungsart A).
	3	Bindung zu PCRE-Bibliotheken relativ zur Applikation oder aus dem System (Anbindungsreihenfolge A – S).
	4	Bindung mit Vorrang für Funktionen im Executable (Anbindungsreihenfolge E – A – S), dies ist für selbstgebaute IDM-Applikationen der Standardfall.
	5	Bindung analog zu *4* aber in umgekehrter Reihenfolge, also Vorrang für die im System installierte PCRE-Bibliothek (Anbindungsreihenfolge S – A – E).
	Siehe auch Kapitel „PCRE-Bibliothek zur Unterstützung Regulärer Ausdrücke“ bei der eingebauten Funktion regex	
DMF_SignalMode	*0*	Die Signale werden über die Funktion **signal** abgefangen.
	1	Die Signale werden über die Funktion **sigaction** abgefangen.

action	options	Bedeutung
DMF_SetSearchPath	*0*	Mit dieser Aktion wird der Suchpfad für IDM-Dateien (Dialog-, Modul-, Interface- und Binärdateien) umgesetzt. Die Semikolon-separierten Verzeichnisse sind im *data*-Parameter als String-Pointer zu übergeben. **Siehe auch** Startoption **-IDMsearchpath**
DMF_SetUsepathModifier	*0*	Mit dieser Aktion wird der Konverter gesteuert, mit dem Use-Pfade zu Dateipfaden gewandelt werden. Die Steuerung erfolgt über einen String als *data*-Parameter. **Wertebereich** » *""* – Leerstring » *"L"* – Umwandlung in Kleinbuchstaben » *"F"* – Umwandlung des ersten Buchstabens eines Pfadteiles in Kleinbuchstaben » *"U"* – Umwandlung in Großbuchstaben » *"u"* – Umwandlung in Großbuchstaben, außer der Dateiendung

action	options	Bedeutung
DMF_SetCodePage		Mit dieser Aktion lässt sich die Codepage von Strings setzen, die zwischen Anwendung und IDM übergeben werden. Normalerweise erwartet und liefert der IDM Strings ISO 8859-1 codiert. Mit dieser Aktion kann eine andere Zeichencodierung für Strings festgelegt werden. Ab IDM-Version A.06.01.d ist es möglich, im Parameter *objectID* ein ***Application***-Objekt anzugeben. Damit wird die applikationsspezifische Codepage umgesetzt, welche für die Verarbeitung von Strings nötig ist. Das Umsetzen einer applikationsspezifischen Codepage innerhalb einer der Funktion der entsprechenden Applikation hat eine sofortige Wirkung. Der Aufruf auf einer DDM-Serverseite unterstützt allerdings kein Umsetzen der Applikationscodepage sondern wirkt sich ohnehin nur auf die Netzwerk-Applikation aus.
DMF_SetFormatCodePage		Definiert die Codepage, in der Formatfunktionen Strings interpretieren und zurückgeben.
Folgende Optionen gelten für DMF_SetCodePage und DMF_SetFormatCodePage		
	CP_ascii	ASCII Zeichencodierung.
	CP_iso8859	Westeuropäische Zeichencodierung Latin-1 nach ISO 8859-1.
	CP_cp437	Englische Zeichencodierung gemäß IBM-Codepage 437 (MS-DOS).
	CP_cp850	Westeuropäische Zeichencodierung gemäß IBM-Codepage 850 (MS-DOS).
	CP_iso6937	Westeuropäische Codierung mit variabler Länge nach ISO 6937.
	CP_winansi	MICROSOFT WINDOWS Zeichencodierung.
	CP_dec169	Zeichencodierung gemäß DEC-Codepage 169.

action	options	Bedeutung
	CP_roman8	8-Bit Zeichencodierung gemäß HP-Codepage Roman-8.
	CP_utf8	8-Bit Unicode-Codierung mit variabler Länge, entspricht im Bereich *0 – 127* der ASCII-Codierung.
	CP_utf16 CP_utf16b CP_utf16l	16-Bit Unicode-Codierung mit variabler Länge von 2 bis 4 Byte. Zwei Varianten: » BE – Big Endian: höherwertige(s) Byte(s) zuerst. » LE – Little Endian: niederwertige(s) Byte(s) zuerst. UTF-16 ohne Angabe der Byte-Reihenfolge entspricht unter Microsoft Windows der LE-Variante, auf Unix-Systemen dagegen der BE-Variante.
	CP_cp1252	Westeuropäische Zeichencodierung gemäß Microsoft Windows Codepage 1252.
	CP_acp	Aktuell von einer Anwendung verwendete ANSI-Codepage unter Microsoft Windows. Nur unter Microsoft Windows verwendbar.
	CP_hp15	Westeuropäische 16-Bit Zeichencodierung von HP-Systemen.
	CP_jap15	Japanische 16-Bit Zeichencodierung von HP-Systemen.
	CP_roc15	Vereinfachte chinesische 16-Bit Zeichencodierung (Kurzzeichen) von HP-Systemen.
	CP_prc15	Traditionelle chinesische 16-Bit Zeichencodierung (Langzeichen) von HP-Systemen.
	CP_ucp	Benutzerdefinierte Codepage ("User Code Page"). Konvertierung in eine beliebige Codepage mit **iconv ()**.

Anmerkungen

» Ein Umschalten zu einer Codepage ist gültig, bis die Codepage wieder gesetzt wird. Alle Strings müssen in dieser Codepage übergeben werden und alle Strings, die der IDM an die Anwendung

übergibt, werden in diese Codepage übersetzt.

» Die Funktion **DM_Control** darf nicht aus einer Canvas Callback-Funktion aufgerufen werden.

Beispiel

Setzen der Applikations-Codepage auf „Roman 8“.

```
DM_TraceMessage ("This application will use roman-8 codepage",  0);
DM_Control (0, DMF_SetCodePage, CP_roman8);
```

Siehe auch

Funktion DM_ControlEx

3.12 DM_ControlEx

Mit dieser Funktion können generelle Einstellungen im ISA DIALOG MANAGER geändert oder Aktionen ausgelöst werden.

Gegenüber der Funktion DM_Control bietet diese Funktion die zusätzliche Aktion *DMF_SetUserCodePage*. Durch *DMF_SetUserCodePage* kann ein Codepage-Systemstring für die **iconv** Wandlung gesetzt werden, der nur dann aktiv wird, wenn eine IDM Codepage auf **CP_ucp** gesetzt ist. Eine Setzung auf **CP_ucp** erfolgt bspw. durch die Aktion *DMF_SetCodePage*. Bei Benutzung dieser Codepage wird dann mittels der auf UNIX/LINUX vorhandenen **iconv**-Routinen eine Codepage-Konvertierung von Zeichen/Strings durchgeführt. Der Benutzer kann die Codepage frei wählen. Da die **iconv**-Funktionalität **nicht** auf MS-Windows oder VMS vorhanden ist, wird dies auf diesen Plattformen nicht unterstützt (Default-Konvertierung nach ?-Zeichen).
Von/zu welcher Codepage eigentlich konvertiert wird kann der Anwender selbst festlegen. Notwendig ist nur, das **iconv** die Konvertierung von/zu **UTF-8** unterstützt! Defaultmäßig ist die User-Codepage "8859-1" gesetzt. Mittels der Aktion DMF_SetUserCodePage kann der Anwendungsprogrammierer die Codepage in der seine Strings sind selbst setzen.

```
DM_boolean DML_default DM_EXPORT DM_ControlEx
(
  DM_ID      objectID,
  DM_UInt    action,
  DM_Pointer data,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Objekt für das die gewünschte Aktion ausgeführt werden soll.

-> DM_UInt action

Aktion, die der IDM ausführen soll. Dazu sind verschiedene Konstanten in der Include-Datei **IDMuser.h** definiert. Diese Konstanten sind in der Tabelle unten erläutert.

-> DM_Pointer data

Parameter zur Übergabe beliebiger Daten an die Funktion.

Bei der Aktion *DMF_SetUserCodePage* muss in diesem Parameter ein Pointer auf einen String mitgegeben werden, der Codepage-Code enthält.

-> DM_Options options

Enthält bei Bedarf ein Argument für die in *action* angegebene Aktion (siehe Tabelle unten).

Rückgabewert

DM_TRUE Die Aktion wurde erfolgreich ausgeführt.

DM_FALSE Die Aktion konnte nicht ausgeführt werden.

Die nachfolgende Tabelle zeigt die gültigen Belegungen der einzelnen Parameter und erläutert ihre Bedeutung. Wenn bei der Aktion nichts anderes gesagt wird, ist als *objectID* immer *0* anzugeben.

action	options	Bedeutung
DMF_UpdateScreen	*0*	Es sollen alle internen SetVal-Aufrufe auf den Bildschirm gebracht werden. In diesem Fall muss der erste Parameter mit dem Dialog belegt sein.
DMF_UIAutomationMode		Mit dieser Aktion kann die spezielle UI Automation Unterstützung des IDM für seine besonderen Objekte abgeschaltet werden. Weiterhin aktiv bleibt aber die UI Automation Unterstützung von Microsoft für die Standard Controls. Standardmäßig ist die Unterstützung von UI Automation aktiv. Die Umschaltung muss vor dem Aufruf von **DM_Initialize** und nach dem Bootstrapping erfolgen.
	0	Deaktiviert den UI Automation Support des IDM.
	1	Aktiviert den UI Automation Support des IDM.

action	options	Bedeutung
DMF_PCREBinding	*0*	Deaktiviert die Anbindung an die PCRE-Bibliothek, Reguläre Ausdrücke sind somit nicht mehr möglich.
	1	Anbindung an statisch im Executable vorhandene PCRE-Funktionen (Anbindungsart E).
	2	Nur dynamische Anbindung von PCRE-Bibliotheken relativ zur Applikation (Anbindungsart A).
	3	Bindung zu PCRE-Bibliotheken relativ zur Applikation oder aus dem System (Anbindungsreihenfolge A – S).
	4	Bindung mit Vorrang für Funktionen im Executable (Anbindungsreihenfolge E – A – S), dies ist für selbstgebaute IDM-Applikationen der Standardfall.
	5	Bindung analog zu *4* aber in umgekehrter Reihenfolge, also Vorrang für die im System installierte PCRE-Bibliothek (Anbindungsreihenfolge S – A – E).
	Siehe auch Kapitel „PCRE-Bibliothek zur Unterstützung Regulärer Ausdrücke“ bei der eingebauten Funktion regex	
DMF_SignalMode	*0*	Die Signale werden über die Funktion **signal** abgefangen.
	1	Die Signale werden über die Funktion **sigaction** abgefangen.
DMF_SetSearchPath	*0*	Mit dieser Aktion wird der Suchpfad für IDM-Dateien (Dialog-, Modul-, Interface- und Binärdateien) umgesetzt. Die Semikolon-separierten Verzeichnisse sind im *data*-Parameter als String-Pointer zu übergeben. **Siehe auch** Startoption **-IDMsearchpath**

action	options	Bedeutung
DMF_SetUsepathModifier	*0*	Mit dieser Aktion wird der Konverter gesteuert, mit dem Use-Pfade zu Dateipfaden gewandelt werden. Die Steuerung erfolgt über einen String als *data*-Parameter. **Wertebereich** » *""* – Leerstring » *"L"* – Umwandlung in Kleinbuchstaben » *"F"* – Umwandlung des ersten Buchstabens eines Pfadteiles in Kleinbuchstaben » *"U"* – Umwandlung in Großbuchstaben » *"u"* – Umwandlung in Großbuchstaben, außer der Dateiendung
DMF_SetCodePage		Mit dieser Aktion lässt sich die Codepage von Strings setzen, die zwischen Anwendung und IDM übergeben werden. Normalerweise erwartet und liefert der IDM Strings ISO 8859-1 codiert. Mit dieser Aktion kann eine andere Zeichencodierung für Strings festgelegt werden. Ab IDM-Version A.06.01.d ist es möglich, im Parameter *objectID* ein ***Application***-Objekt anzugeben. Damit wird die applikationsspezifische Codepage umgesetzt, welche für die Verarbeitung von Strings nötig ist. Das Umsetzen einer applikationsspezifischen Codepage innerhalb einer der Funktion der entsprechenden Applikation hat eine sofortige Wirkung. Der Aufruf auf einer DDM-Serverseite unterstützt allerdings kein Umsetzen der Applikationscodepage sondern wirkt sich ohnehin nur auf die Netzwerk-Applikation aus.
DMF_SetFormatCodePage		Definiert die Codepage, in der Formatfunktionen Strings interpretieren und zurückgeben.

action	options	Bedeutung
DMF_SetUserCodePage		Stellt den Zeichencode für **iconv** ein, und beeinflusst damit indirekt die IDM Codepage **CP_ucp**. Diese wird mittels **DMF_SetCodePage** aktiviert. (Nur auf Plattformen, die iconv unterstützen). Der Codepage-Code wird im Parameter *data* übergeben (Pointer auf einen String der Codepage-Code enthält).
Folgende Optionen gelten für DMF_SetCodePage und DMF_SetFormatCodePage		
	CP_ascii	ASCII Zeichencodierung.
	CP_iso8859	Westeuropäische Zeichencodierung Latin-1 nach ISO 8859-1.
	CP_cp437	Englische Zeichencodierung gemäß IBM-Codepage 437 (MS-DOS).
	CP_cp850	Westeuropäische Zeichencodierung gemäß IBM-Codepage 850 (MS-DOS).
	CP_iso6937	Westeuropäische Codierung mit variabler Länge nach ISO 6937.
	CP_winansi	MICROSOFT WINDOWS Zeichencodierung.
	CP_dec169	Zeichencodierung gemäß DEC-Codepage 169.
	CP_roman8	8-Bit Zeichencodierung gemäß HP-Codepage Roman-8.
	CP_utf8	8-Bit Unicode-Codierung mit variabler Länge, entspricht im Bereich *0 – 127* der ASCII-Codierung.
	CP_utf16 CP_utf16b CP_utf16l	16-Bit Unicode-Codierung mit variabler Länge von 2 bis 4 Byte. Zwei Varianten: » BE – Big Endian: höherwertige(s) Byte(s) zuerst. » LE – Little Endian: niederwertige(s) Byte(s) zuerst. UTF-16 ohne Angabe der Byte-Reihenfolge entspricht unter Microsoft Windows der LE-Variante, auf Unix-Systemen dagegen der BE-Variante.

action	options	Bedeutung
	CP_cp1252	Westeuropäische Zeichencodierung gemäß Microsoft Windows Codepage 1252.
	CP_acp	Aktuell von einer Anwendung verwendete ANSI-Codepage unter Microsoft Windows. Nur unter Microsoft Windows verwendbar.
	CP_hp15	Westeuropäische 16-Bit Zeichencodierung von HP-Systemen.
	CP_jap15	Japanische 16-Bit Zeichencodierung von HP-Systemen.
	CP_roc15	Vereinfachte chinesische 16-Bit Zeichencodierung (Kurzzeichen) von HP-Systemen.
	CP_prc15	Traditionelle chinesische 16-Bit Zeichencodierung (Langzeichen) von HP-Systemen.
	CP_ucp	Benutzerdefinierte Codepage ("User Code Page"). Konvertierung in eine beliebige Codepage mit **iconv ()**.

Anmerkungen

» Ein Umschalten zu einer Codepage ist gültig, bis die Codepage wieder gesetzt wird. Alle Strings müssen in dieser Codepage übergeben werden und alle Strings, die der IDM an die Anwendung übergibt, werden in diese Codepage übersetzt.

» Die Funktion **DM_ControlEx** darf nicht aus einer Canvas Callback-Funktion aufgerufen werden.

Beispiel

Setzen der Applikations-Codepage unter Verwendung von CP_ucp und DMF_SetUserCodePage.

```
/* Applikations-Codepage auf CP_ucp setzen und CP1250 verwenden. */
...
DM_Control((DM_ID)0, DMF_SetCodePage, CP_ucp);
DM_ControlEx((DM_ID)0, DMF_SetUserCodePage, "CP1250", 0)
...
```

Siehe auch

Funktion DM_Control

3.13 DM_CreateObject

Jede Instanz oder jedes Modell innerhalb eines Dialogs kann mit dieser Funktion generiert werden. Der erste Parameter beschreibt den Typ des zu generierenden Objekts (Pushbutton, Fenster etc.); der zweite Parameter beschreibt den Vater des neuen Objekts, der letzte beschreibt die Art (Instanz/Modell) des neuen Objekts.

```
DM_ID DML_default DM_EXPORT DM_CreateObject
(
  DM_ID classtagOrModel,
  DM_ID parentID,
  DM_Options options
)
```

Parameter

-> DM_ID classtagOrModel

Dieser Parameter beschreibt die Art des neuen Objekts. Die notwendigen Definitionen werden von der **IDMuser.h** miteinbezogen.

Die folgenden Definitionen werden akzeptiert:

- DM_ClassCanvas
- DM_ClassCheck
- DM_ClassControl
- DM_ClassEditext
- DM_ClassGroupbox
- DM_ClassImage
- DM_ClassImport
- DM_ClassListbox
- DM_ClassMenubox
- DM_ClassMenuitem
- DM_ClassMenusep
- DM_ClassMessagebox
- DM_ClassModule
- DM_ClassNotebook
- DM_ClassNotepage
- DM_ClassPoptext
- DM_ClassProgressbar
- DM_ClassPush
- DM_ClassRadio

- DM_ClassRecord
- DM_ClassRect
- DM_ClassScroll
- DM_ClassSpinbox
- DM_ClassSplitbox
- DM_ClassStatext
- DM_ClassStatusbar
- DM_ClassTablefield
- DM_ClassTreeview
- DM_ClaasToolbar
- DM_ClassTimer
- DM_ClassWindow

-> DM_ID parentID

In diesem Parameter wird der Vater des neu zu generierenden Objektes angegeben. Für das Anlegen eines dialog Objekts muss dieser Parameter *NULL* sein.

-> DM_Options options

Für diese Funktion sind verschiedene Optionen möglich, die auch mit einem "oder" (|) verknüpft angegeben werden können.

Option	Bedeutung
DMF_CreateModel	Ist diese Option gesetzt, so soll mit Hilfe dieser Funktion ein neues Modell generiert werden.
DMF_CreateInvisible	Ist diese Option gesetzt, soll das neu generierte Objekt unabhängig von der Definition in der angegebenen Vorlage unsichtbar generiert werden.
DMF_InheritFromModel	Diese Option muss genau dann gesetzt sein, wenn in dem Parameter *classtagOrModel* die DM_ID eines Modells angegeben ist. Damit wird der Funktion angegeben, dass von diesem Modell eine neue Instanz generierter werden soll. Wenn es sich dabei um ein hierarchisches Modell handelt, so werden auch dessen Kinder in das neue Objekt generiert.

Rückgabewert

ID != 0 Objekt konnte generiert werden.

ID = 0 Objekt konnte nicht generiert werden.

Beispiel

Von einer C-Funktion aus soll eine neue Instanz eines Fenster-Modells generiert werden.

```
void DML_default DM_ENTRY CreateNewInstance
(
    DM_ID Modell,
    DM_ID DialogID
)
{
  DM_ID NewObject;
  DM_Value data;

  if ((NewObject = DM_CreateObject (Modell, DialogID,
    DMF_InheritFromModel | DMF_CreateInvisible)) != (DM_ID) 0)
  {
    /* Setzen der Daten im neu generierten Fenster */

    /* Abschließend Fenster sichtbar machen */
    data.type = DT_boolean;
    data.value.boolean = TRUE;
    DM_SetValue(NewObject, AT_visible, 0, &data, 0);
  }
}
```

Siehe auch

Eingebaute Funktion create() im Handbuch „Regelsprache“

Methode :create()

3.14 DM_DataChanged

Mit dieser Funktion kann signalisiert werden, dass sich an einem bestimmten Datenmodell (Model-Komponente) der Wert des angegebenen Attributs (Model-Attribut) geändert hat. Diese Änderung wird als *datachanged*-Ereignis in die Ereignisverarbeitung gestellt. Erst bei der weiteren Ereignisverarbeitung werden dann die gekoppelten Präsentationsobjekte (View-Komponenten) veranlasst, die neuen Datenwerte zu holen und zur Anzeige zu bringen.

```
DM_Boolean DML_default DM_EXPORT DM_DataChanged
(
  DM_ID        object,
  DM_Attribute attribute,
  DM_Value  *  index,
  DM_Options   options
)
```

Parameter

-> DM_ID object

Dieses ist die Objekt-ID des Datenmodell-Objekts an dem sich ein Datenwert geändert hat.

-> DM_Attribute attribute

Dieser Parameter bezeichnet das Attribut des Datenmodells, das sich geändert hat.

-> DM_Value *index

Dieser Parameter definiert den Index des geänderten Attributs. Wird hier *NULL* angegeben ist dies gleichbedeutend mit der Änderung aller Einzelwerte.

-> DM_Options options

Option	Bedeutung
0	Es findet kein Tracing dieser Funktion statt.
DMF_Verbose	Aktiviert das Tracing dieser Funktion.

Rückgabewert

DM_TRUE Änderung konnte erfolgreich als *datachanged*-Ereignis gespeichert werden.

DM_FALSE Es konnte kein Ereignis erzeugt werden.

Beispiel

Dialogdatei

```
dialog D
function datafunc FuncData();
function void     Reverse(integer Idx);

window Wi
{
  .title "Datafunc demo";
  .width 200;  .height 300;

  edittext Et
  {
    .datamodel FuncData;
    .dataget .text;
    .dataset .text;
    .xauto 0;
    .xright 80;
  }

  pushbutton PbAdd
```

```
  {
    .text "Add";
    .xauto -1;
    .width 80;

    on select
    {
      this.window:apply();
    }
  }

  listbox Lb
  {
    .xauto 0;  .yauto 0;
    .ytop 30;  .ybottom 30;
    .datamodel FuncData;
    .dataget .content;

    on select
    {
      PbReverse.sensitive := true;
    }
  }

  pushbutton PbReverse
  {
    .xauto 0;  .yauto -1;
    .text "Reverse element";
    .sensitive false;

    on select
    {
      Reverse(Lb.activeitem);
    }
  }

  on close { exit(); }
}
```

C-Datei

```
#ifdef VMS
#  define EXITOK    1
#  define EXITERROR 0
#else
#  define EXITOK    0
#  define EXITERROR 1
#endif
```

```
#include <IDMuser.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "datafuncfm.h"

#define DIALOGFILE "~:datafunc.dlg"

static DM_ID     data_id = (DM_ID)0;
static DM_Value  data_vec;
static DM_String data_str;

void DML_default DM_CALLBACK FuncData __1((DM_DataArgs *, args))
{
  if (!data_id)
    data_id = args->object;
  switch (args->task)
  {
  case MT_get:
    switch(args->attribute)
    {
    case AT_text:
      args->retval.type = DT_string;
      args->retval.value.string = data_str;
      break;
    case AT_content:
      if (args->index.type == DT_void)
      {
        args->retval = data_vec;
      }
      else if (args->index.type == DT_integer)
      {
        DM_ValueGet(&data_vec, &args->index, &args->retval, 0);
      }
      break;
    default:
      break;
    }
    break;
  case MT_set:
    switch(args->attribute)
    {
    case AT_text:
      if (args->data.type == DT_string)
      {
```

```
        if (DM_ValueChange(&data_vec, NULL, &args->data, DMF_AppendValue))
          DM_DataChanged(data_id, AT_content, NULL, DMF_Verbose);
      }
      break;
    default:
      break;
    }
    break;
  default:
    break;
  }
}

void DML_default DM_ENTRY Reverse __1((DM_Integer, Idx))
{
  DM_Value  index, data;
  char     *cp,    ch;
  size_t    len,   i;

  DM_ValueInit(&data, DT_void, NULL, 0);
  index.type = DT_integer;
  index.value.integer = Idx;
  if (DM_ValueGet(&data_vec, &index, &data, 0) && data.type == DT_string)
  {
    if (DM_StringChange(&data_str, data.value.string, 0))
      DM_DataChanged(data_id, AT_text, NULL, DMF_Verbose);

    /* reverse the string */
    cp = data.value.string;
    if (cp)
    {
      len = strlen(cp);
      if (len>2)
      {
        len--;
        for (i=0; len>0 && i<len/2; i++)
        {
          ch = cp[i];
          cp[i] = cp[len-i];
          cp[len-i] = ch;
        }
      }
    }
    if (DM_ValueChange(&data_vec, &index, &data, 0) && data_id)
      DM_DataChanged(data_id, AT_content, &index, DMF_Verbose);
  }
}
```

```
int DML_c AppMain __2((int, argc), (char **,argv))
{
  DM_ID dialogID;
  DM_Value data;

  /* initialize the Dialog Manager */
  if (!DM_Initialize (&argc, argv, 0))
  {
    DM_TraceMessage("Could not initialize.", 0);
    return (1);
  }

  /* load the dialog file */
  switch(argc)
  {
  case 1:
    dialogID = DM_LoadDialog (DIALOGFILE,0);
    break;
  case 2:
    dialogID = DM_LoadDialog (argv[1],0);
    break;
  default:
    DM_TraceMessage("Too many arguments.", 0);
    return(EXITERROR);
    break;
  }
  if (!dialogID)
  {
    DM_TraceMessage("Could not load dialog.", 0);
    return(EXITERROR);
  }

  data.type = DT_type;
  data.value.type = DT_string;
  DM_ValueInit(&data_vec, DT_vector, &data, DMF_StaticValue);

  data.type = DT_string;
  data.value.string = "^ Enter a string";
  DM_ValueChange(&data_vec, NULL, &data, DMF_AppendValue);
  data.value.string = "and press 'Add'";
  DM_ValueChange(&data_vec, NULL, &data, DMF_AppendValue);

  DM_StringInit(&data_str, DMF_StaticValue);
  DM_StringChange(&data_str, "Change me!", 0);

  /* install table of application functions */
```

```
  if (!BindFunctions_D (dialogID, dialogID, 0))
    DM_TraceMessage ("There are some functions missing.", 0);

  /* start the dialog and enter event loop */
  if (DM_StartDialog (dialogID, 0))
    DM_EventLoop (0);
  else
    return (EXITERROR);

  return (EXITOK);
}
```

3.15 DM_Destroy

Mit Hilfe dieser Funktion können beliebige Objekte oder Modelle innerhalb des Dialogs gelöscht werden. Dabei werden alle Kinder des Objekts mit gelöscht. Mit Hilfe des zweiten Parameters wird gesteuert, was gelöscht werden soll.

```
DM_Boolean DML_default DM_EXPORT DM_Destroy
(
  DM_ID objectID,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

In diesem Parameter wird das Objekt angegeben, das gelöscht werden soll.

-> DM_Options options

Über diesen Parameter wird gesteuert, welches Verhalten beim Löschen gefordert ist. Dabei gibt es folgende Möglichkeiten:

Option	Bedeutung
DMF_ForceDestroy	Wenn als Option *DMF_ForceDestroy* angegeben wird, werden das Objekt gelöscht und alle Regelteile, die dieses Objekt benutzen, abgeändert, sodass die entsprechenden Anweisungen entfernt werden. Falls es sich bei dem zu löschenden Objekt um ein Modell handelt, wird ohne den Parameter *DMF_ForceDestroy* nur die erneute Referenzierung des Modells verboten, aber es bleibt weiterhin als Modell bestehen. Wird *DMF_ForceDestroy* angegeben, so wird bei allen Objekten, die dieses Modell benutzen, dieses Modell entfernt und sie übernehmen wieder die Werte des nächsthöheren Modells bzw. Defaults.

Rückgabewert

TRUE	Objekt konnte gelöscht werden.
FALSE	Objekt konnte nicht gelöscht werden.

DM_Destroy() ruft die :clean()-Methode des zu zerstörenden Objekts auf.

Beispiel

Zerstören eines Objektes von einer C-Funktion aus.

```
DM_Boolean DML_default DM_ENTRY DestroyObject
(
    DM_ID ObjID
)
```

```
{
  return (DM_Destroy(ObjID, DMF_ForceDestroy));
}
```

Siehe auch

Eingebaute Funktion destroy() im Handbuch „Regelsprache“

Methode :destroy()

3.16 DM_DialogPathToID

Achtung

Die Funktion **DM_DialogPathToID** ist veraltet und wird nur noch aus Kompatibilitätsgründen unterstützt. An ihrer Stelle sollte DM_ParsePath verwendet werden.

Mit Hilfe dieser Funktion können Sie den Identifikator eines Objekts erfragen, wenn Sie mehr als einen Dialog geladen haben und das gesuchte Objekt sich nicht in dem zuerst geladenen Dialog befindet.

```
DM_ID DML_default DM_EXPORT DM_DialogPathToID
(
  DM_ID dialogid,
  DM_ID rootid,
  DM_String path
)
```

Parameter

-> DM_ID dialogid

Dies ist der Identifikator des Dialoges, in dem nach dem Objekt gesucht werden soll.

-> DM_ID rootid

Mit Hilfe dieses Parameters können Sie steuern, ab welchem Objekt der Dialog Manager die Suche nach dem von Ihnen gewünschten Objekt beginnt. Dabei gibt es folgende Möglichkeiten:

- *rootid = 0*
  Der Dialog Manager sucht in der gesamten Dialogdefinition nach dem angegebenen Objekt. Dies ist der Normalfall. Auf diese Art können auch die Identifikation von Regeln, Funktionen, Variablen und Ressourcen erfragt werden.
- *rootid != 0*
  Der Dialog Manager soll ab dem angegebenen Objekt nur auf der nächsttieferen Hierarchiestufe suchen; tiefere Hierarchiestufen werden nicht durchsucht.
  Diese Vorgehensweise ist nur dann angebracht, wenn in einem Dialog ein Objektname mehr als einmal auftritt.

-> DM_String path

Mit Hilfe dieses Pfades wird das gesuchte Objekt beschrieben. Dieser Pfad muss eindeutig ein Objekt beschreiben. Existiert der Objektname nur einmal innerhalb des Dialoges, so reicht die Angabe des Namens, um die gewünschte Referenz zu erhalten. Ist der Objektname nicht eindeutig, muss das Objekt über einen Pfad von Objektnamen getrennt durch einen Punkt beschrieben werden.

Rückgabewert

0	Das gesuchte Objekt wurde nicht gefunden oder der Name ist nicht eindeutig.
!= 0	Identifikator des gesuchten Objekts.

Anmerkung

Wenn *"setup"* als *path*angegeben wird und sowohl *dialogid* als auch *rootid* gleich *0* sind, dann wird das ***Setup***-Objekt zurückgegeben.

Beispiel

```
void DML_default DM_ENTRY OkButtonCallback __1((DM_ID, dialogID))
{
  DM_ID ID1;
  DM_ID ID2;

  /* Abfrage eines Objektes im Dialog global */
  ID1 = DM_DialogPathToID(dialogID, 0, "ErstesObjekt");

  /* Abfrage über einen Pfad */
  ID2 = DM_DialogPathToID(dialogID, 0, "ErstesObjekt.Kind1");
}
```

Siehe auch

Funktion DM_ParsePath

Eingebaute Funktion parsepath im Handbuch „Regelsprache“

3.17 DM_DispatchHandler

Mit Hilfe der Funktion **DM_DispatchHandler** kann beim IDM FÜR MOTIF eine benutzerdefinierte Funktion (Handler) registriert werden, die vor der Funktion **XtDispatchEvent** aufgerufen wird. Dies ermöglicht, *XEvents* auf X-Ebene (und nicht nur auf der X-Toolkit-Ebene) zu verarbeiten.

```
DM_Boolean DML_default DM_EXPORT DM_DispatchHandler
(
  DM_DispatchHandlerProc  funcp,
  DM_UInt     chain_pos,
  DM_UInt     operation,
  DM_Options  options
)
```

Parameter

-> DM_DispatchHandlerProc funcp

Funktionspointer zur benutzerdefinierten Handlerfunktion. Diese Funktion erhält das *XEvent* als Parameter.

Die übergebene Funktion muss wie folgt definiert sein:

```
DM_Boolean DML_default DM_EXPORT MyHandler
(
  XEvent * event
)
```

-> DM_UInt chain_pos

Dieser Parameter bestimmt, ob die Handlerfunktion an den Anfang (*DMF_InstallHead*) oder an das Ende (*DMF_InstallTail*) der Liste der Handlerfunktionen eingefügt wird.

chain_pos wird nur ausgewertet, wenn der Parameter *operation* auf *DMF_RegisterHandler* gesetzt ist.

-> DM_UInt operation

In diesem Parameter wird der Funktion die eigentlich auszuführende Operation mitgeteilt. Dazu sind folgende Konstanten definiert:

operation	Bedeutung
DMF_RegisterHandler	Mit Hilfe dieses Wertes wird ein Handler installiert.
DMF_WithdrawHandler	Mit Hilfe dieses Wertes wird ein vorher installierter Handler deinstalliert.
DMF_DisableHandler	Mit Hilfe dieses Wertes wird ein Handler vorübergehend deaktiviert.

operation	Bedeutung
DMF_EnableHandler	Mit Hilfe dieses Wertes wird ein deaktivierter Handler wieder aktiviert.

-> DM_Options options

Ist *options* auf den Wert *DMF_DontTrace* gesetzt, erfolgt keine Protokollierung des Aufrufs im Tracefile.

Rückgabewert

DM_TRUE	Das *XEvent* wurde vollständig verarbeitet. Es wird kein weiterer „DispatchHandler“ aufgerufen und auch nicht die Funktion **XtDispatchEvent**.
DM_FALSE	Der nächste registrierte „DispatchHandler“, bzw. als letztes die Funktion **XtDispatchEvent**, wird aufgerufen.

3.18 DM_DumpState

Mit dieser Funktion werden IDM Zustandsinformationen in die Log- bzw. Tracedatei herausgeschrieben.

Syntax

```
void DML_default DM_EXPORT DM_DumpState
(
  DM_Enum state,
  DM_Options options
)
```

Parameter

-> DM_Enum state

Dieser Parameter beeinflusst, welche Abschnitte der Zustandsinformationen herausgeschrieben werden.

Der Dumpstate ist eine Zustandsinformation von IDM-relevanten Informationen um die Fehleranalyse einer IDM-Applikation zu erleichtern.

Der Inhalt des Dumpstate ist in verschiedene Abschnitte unterteilt, die variabel und der Fehlersituation angepasst sind. Außerdem wird er von zuvor aufgetretenen Fehlern beeinflusst. Beispielsweise führt eine erfolglose Speicherallokierung bei der nächsten Dumpstate-Ausgabe dazu, dass Informationen bzgl. der Speichernutzung durch den IDM ausgegeben werden. Konnten keine IDM-Objekte oder Bezeichner mehr angelegt werden, wird die Auslastung von IDM-Objekten und Bezeichnern ausgegeben.

Die Dumpstate-Information ist immer zwischen *„*** DUMP STATE BEGIN ***“* und *„*** DUMP STATE END ***“* eingeschlossen und kann folgende Abschnitte aufweisen:

- PROCESS: Prozess- und Thread-Nummer, Datum/Uhrzeit.
- ERRORS: Kompletter Inhalt der gesetzten Fehlercodes.
- CALLSTACK: Aufruf-Stack – beinhaltet Regeln, DM-Schnittstellenfunktionen und die obersten, vom IDM aufgerufenen, Applikationsfunktionen.
- THISEVENTS und EVENT-QUEUE: Aktuell verarbeitete thisevent-Objekte und deren Werte sowie Ereignisse die noch in der Warteschlange stehen.
- USAGE: Anzahl angelegter Objekte, Module und Bezeichner, Speichergröße die vom Regelinterpreter und für die String-Übergabe genutzt wird.
- MEMORY: Speichernutzung die vom IDM erfassbar ist.
- SLOTS: Hinweise zu IDM-Objekten die nicht richtig freigegeben wurden.
- VISIBLE OBJECTS: Liste der sichtbaren Objekte mit ihren Werten.

Um die Ausgabe möglichst klein zu halten, erfolgt sie normalerweise in gekürzter Form. Grundsätzlich immer erfolgt eine Kürzung von IDM-Strings (in “…“) auf maximal 40 Zeichen. Ihre

Gesamtlänge wird in [ ] angehängt. Bytegrößen-Angaben erfolgen gekürzt auf Kilo-, Mega oder Gigabyte (k/m/g).

Folgende Werte sind möglich:

Wert (enum)	Bedeutung
DM_DUMP_all	Alle Abschnitte werden gekürzt herausgeschrieben. Entspricht der Ausgabe bei einem FATAL ERROR.
DM_DUMP_error	Die Abschnitte ERRORS, CALLSTACK und EVENTS werden gekürzt herausgeschrieben. Dies ist auch die normale Ausgabe im Falle eines EVAL ERRORS.
DM_DUMP_events	Die Abschnitte THISEVENTS und EVENT QUEUE werden ungekürzt herausgeschrieben.
DM_DUMP_full	Alle Abschnitte werden ungekürzt herausgeschrieben.
DM_DUMP_locked	Der Abschnitt SLOTS wird ungekürzt herausgeschrieben. Zusätzlich werden für gesperrte (locked) Objekte deren Attributwerte herausgeschrieben.
DM_DUMP_memory	Der Abschnitt MEMORY wird ungekürzt herausgeschrieben.
DM_DUMP_none	Nichts passiert (kein Herausschreiben).
DM_DUMP_process	Der Abschnitt PROCESS wird ungekürzt herausgeschrieben.
DM_DUMP_short	Alle Abschnitte (außer SLOTS) werden gekürzt herausgeschrieben.
DM_DUMP_slots	Der Abschnitt SLOTS wird ungekürzt herausgeschrieben.
DM_DUMP_stack	Der Abschnitt CALLSTACK wird ungekürzt herausgeschrieben.
DM_DUMP_usage	Der Abschnitt USAGE wird ungekürzt herausgeschrieben.
DM_DUMP_uservisible	Der Abschnitt VISIBLE OBJECTS wird ungekürzt für alle sichtbaren Toplevel-Objekte inklusive deren Kindobjekte, vordefinierten und benutzerdefinierten Attribute herausgeschrieben.
DM_DUMP_visible	Der Abschnitt VISIBLE OBJECTS wird ungekürzt herausgeschrieben.

Kombinationen aus mehreren Abschnitten sind nicht möglich.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Derzeit bitte nur 0 angeben.

Die Ausgabe von IDM Zustandsinformationen kann auch über die eingebaute Funktion dumpstate und die Startoptionen **-IDMdumpstate <enum>** und **-IDMdumpstateseverity <string>** erreicht werden.

Verfügbarkeit

Versionen A.05.01.g3, A.05.01.h sowie ab Version A.05.02.e

Siehe auch

Kapitel „Dumpstate (Zustandsinformationen)“ im Handbuch „Entwicklungsumgebung“

3.19 DM_ErrMsgText

Diese Funktion gibt den Fehlerstring zurück, der zu dem Fehlercode gehört.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_String DML_default DM_EXPORT DM_ErrMsgText
(
  DM_ErrorCode eno,
  DM_Options options
)
```

Parameter

-> DM_ErrorCode eno

Dies ist der Fehlercode, zu dem der Fehlertext zurückgegeben werden soll.

-> DM_Options options

Über verschiedene Optionen kann gesteuert werden, welche Informationen in den Fehlertext integriert werden. Dabei gibt es folgende Möglichkeiten, die natürlich mit einem "oder" verbunden kombiniert angegeben werden können.

Option	Bedeutung
DMF_IncludeIdent	Der Name des Teils, der den Fehler erzeugt hat, soll in dem Fehlertext enthalten sein. Dies kann das Betriebssystem, das Fenstersystem oder der Dialog Manager sein.
DMF_IncludeModule	Der Name des Moduls, in dem der Fehler aufgetreten ist, soll in der Fehlermeldung enthalten sein.
DMF_IncludeSeverity	Der Grad des Fehlers (Warnung, Fehler, fataler Fehler) soll in dem Fehlertext enthalten sein.
DMF_IncludeText	Diese Option besagt, dass der eigentliche Fehlertext in der Fehlermeldung enthalten sein soll.

Beispiel

```
/* error handling function for dialog manager errors */
static void QueryError ( )
{
  /* buffer for the errors occurred */
  DM_ErrorCode errorbuffer[32];
  register int i;
  /* number of errors */
```

```
  int errors;

  if ((errors = DM_QueryError(errorbuffer, 32, 0)))
    for (i = 0, i < errors; i++)
      DM_TraceMessage(DM_ErrMsgText(errorbuffer[i], 0), 0);
}
```

3.20 DM_ErrorHandler

Mit dieser Funktion kann eine Handlerfunktion eingerichtet werden, die beim Auftreten eines Fehlers, welcher vom Regelinterpreter erkannt wird, aufgerufen wird.

```
DM_Boolean DML_default DM_EXPORT DM_ErrorHandler
(
  DM_ErrorHandlerProc  funcp,
  DM_UInt     operation,
  DM_Options  options
)
```

Parameter

-> DM_ErrorHandlerProc funcp

Dieses ist ein Zeiger auf die Funktion, die als Fehlerhandler installiert werden soll.

-> DM_UInt operation

In diesem Parameter wird die Aktion angegeben, die ausgeführt werden soll. Dabei gibt es folgende Werte:

DMF_RegisterHandler	Handlerfunktion soll installiert werden
DMF_EnableHandler	Handlerfunktion soll wieder aufgerufen werden
DMF_DisableHandler	Handlerfunktion soll nicht mehr ausgerufen werden
DMF_WithdrawHandler	Handlerfunktion soll abgemeldet werden

-> DM_Options options

Zurzeit unbenutzt, muss *0* sein.

Rückgabewert

DM_TRUE	Aktion erfolgreich.
DM_FALSE	Aktion konnte nicht durchgeführt werden. Dies kann Auftreten, wenn durch die Aktion der Handler zweimal existieren würde, der Funktionspointer *NULL* ist oder der von der Aktion angesprochene Handler nicht gefunden werden konnte.

Ein nicht abgefangener Fehler im Regelcode der Anwendung (z.B. fehlerhafte Parametertypen, dynamischer Zugriff auf ein nicht vorhandenes Attribut oder relatives Kind, jeweils nicht über `fail()` abgefangen) wird vom Regelinterpreter, als *„EVAL ERROR“* in die Log- oder Tracedatei geschrieben. Meldungen in „[]“ die mit „W:“, „E:“ oder „I:“ anfangen wie Modul- bzw. Interface-Ladefehler gehören nicht zum Regelinterpreter und werden insofern auch nicht an den Fehlerhandler weitergeleitet.

Um es IDM-Anwendungen zu ermöglichen über diese Anwendungsfehler vorher informiert zu werden, dient der Fehlerhandler. Mehrere, nicht identische, Fehlerhandler sind möglich. Wobei der Aufruf in umgekehrter Reihenfolge des Anmeldens erfolgt. Nur die aktivierten Fehlerhandler werden aufgerufen und erhalten dabei die gleiche Informationsstruktur (**DM_ErrorInfo**) als Parameter mit. Diese Aufrufe werden in die Trace-Datei mitprotokolliert. Ein rekursiver Aufruf der Fehlerhandler wird unterbunden. Ein Aufruf der **DM_ErrorHandler-**Funktion innerhalb eines Fehlerhandlers darf keine neuen Handler installieren oder deinstallieren.

Es ist zu bedenken, dass der Fehlerhandler synchron aufgerufen wird und insofern Design und Kodierung mit besonderer Sorgfalt und unter Berücksichtigung der entsprechenden Konstellationen und Restriktionen erfolgen muss. So ist durchaus ein Aufruf der Handler vor dem Start des Dialoges möglich (z.B. wenn Fehler in **:init()**-Methoden vorliegen) wie auch bei unsachgemäßer Benutzung in einem ungeeigneten Runstate (z.B. erzwungener Aufruf einer fehlerhaften Regel durch **DM_CallRule** innerhalb einer Formatfunktion). Eine Vermeidung solcher Konstellationen ist sinnvoll. Auch sollte, wenn der IDM für die Darstellung einer Fehlerbox verwendet wird, ein eigenständiger, vor der Anwendung gestarteten, Fehlerdialog benutzt werden.

Die zu installierende Fehlerhandler-Funktion muss dabei wie folgt definiert sein:

```
typedef void (DML_default DM_CALLBACK *DM_ErrorHandlerProc) __((DM_ErrorInfo
*info));
```

Die mitgegebene Informationsstruktur gibt dabei Auskunft über den Fehler:

```
define  EITK_rule_engine  1

typedef struct {
  /* user info, not changeable */
  DM_UInt1      task;     /* task/component which detects the error */
  DM_ErrorCode  errcode;  /* error code */
  DM_ID         object;   /* this object or null-ID */
  DM_ID         rule;     /* rule where error occured */
  DM_String     file;     /* module/dialgo filename or NULL */
  DM_String     message;  /* error message or NULL*/
} DM_ErrorInfo;
```

Diese Informationen dürfen von keinem Fehlerhandler verändert werden.

Strings liegen grundsätzlich in der Applikations-Codepage vor.

Der *task*-Eintrag gibt Auskunft über die IDM-Komponente, die den Fehler meldet, und sollte wegen möglichen zukünftigen Erweiterungen immer überprüft werden. Momentan werden nur Fehler vom Regelinterpreter als *EITK_rule_engine* weitergegeben.

Die Einträge geben Auskunft über den Fehlercode, die Regel in der der Fehler aufgetreten ist sowie den dazugehörigen Fehlertext. Wenn möglich, wird das `this`-Objekt sowie der Dateiname des Moduls bzw. Dialogs, in dem sich die fehlerhafte Regel befindet, bereitgestellt.

Beispiel

```
#include <IDMuser.h>
void DML_default DM_CALLBACK ErrorHandler __1((DM_ErrorInfo *, info))
{
  if (info->task == EITK_rule_engine)
  {
    DM_TraceMessage("ERROR: errcode=%d message=\"%s\" rule=\"$I\"",
      DMF_LogFile|DMF_Printf,
      info->errcode, info->message,
      info->rule);
  }
}
int DML_c DM_CALLBACK AppMain (int argc, char **argv)
{
  DM_ID dialogID = (DM_ID)0;
  if (DM_Initialize (&argc, argv, 0) == FALSE)
    return (1);
  if (argc>1)
  {
    if (!(dialogID = DM_LoadDialog (argv[1], 0)))
      return (1);
    DM_ErrorHandler(ErrorHandler, DMF_RegisterHandler, 0);
    DM_StartDialog (dialogID, 0);
    DM_EventLoop (0);
    return (0);
  }
  return 1;
}
```

3.21 DM_EventLoop

Diese Funktion startet die Verarbeitung des Dialogs. Dadurch wird der Benutzer in die Lage versetzt, mit dem Dialog zu arbeiten. Erst nach diesem Funktionsaufruf werden vom DM Ereignisse verarbeitet, denn diese Funktion übernimmt die Bearbeitung des Dialogs.

```
DM_Boolean DML_default DM_EXPORT DM_EventLoop
(
  DM_Options options
)
```

Parameter

-> DM_Options options

Mit Hilfe dieses Parameters wird gesteuert, ob die Ereignisverarbeitung abgebrochen werden soll, wenn kein Ereignis mehr vorliegt oder ob die Bearbeitung bis zum Dialogende durchgeführt werden soll.

Option	Bedeutung
0	Das ist der Normalfall. Der Dialog Manager soll die Verarbeitung aller Ereignisse übernehmen und erst beim Ende des Programms aus dieser Funktion zurückkommen.
DMF_DontWait	Mit Hilfe dieser Konstanten wird der Dialog Manager informiert, dass er nicht in eine passive Warteschleife im Fenstersystem gehen soll, wenn kein Ereignis anliegt. Er soll dann sofort zu der Anwendung zurückkommen, damit diese irgendwelche Aktionen ausführen kann. Danach muss von der Anwendung die Funktion DM_EventLoop für weitere Verarbeitungen wieder aufgerufen werden.
DMF_WaitForEvent	Mit Hilfe dieser Konstanten wird der Dialog Manager informiert, dass er passiv auf ein Benutzerereignis im Fenstersystem warten kann. Nach dessen Verarbeitung soll er dann in die Anwendung zurückkommen, damit hier irgendwelche Aktionen durchgeführt werden können und anschließend wieder die Funktion DM_EventLoop aufgerufen werden kann.

Rückgabewert

TRUE	Die Ereignisverarbeitung wurde beendet da im Moment kein Ereignis mehr zur Verarbeitung ansteht; es sind aber noch Dialoge gestartet.
FALSE	Die Ereignisverarbeitung wurde beendet da das Dialogende erreicht worden ist (Standard).

Beispiel

Übergabe der Verarbeitung an den Dialog Manager im Hauptprogramm:

```
int DML_c DM_CALLBACK AppMain __2(
(int, argc),
(char far * far *, argv))
{
  DM_ID dialogID;
  static char * dialogfile = "format.dlg";

  DM_Initialize(&argc, argv, 0);
  dialogID = DM_LoadDialog (dialogfile, 0);

  if (!dialogID)
    return(1);

  (void) DM_BindCallBacks(funcmap, NFUNCS, dialogID, 0);

  DM_StartDialog(dialogID, 0);

  DM_EventLoop(0);

  return (0);
}
```

3.22 DM_ExceptionHandler

Mit Hilfe dieser Funktion kann ein Handler installiert werden, der mögliche "asserts" im Dialog Manager abfangen und eine geeignete Nachricht an den Benutzer ausgeben kann. Zusätzlich kann der Handler Datenbanken und Dateien schließen und so in einem definierten Zustand hinterlassen.

```
DM_Boolean DML_default DM_EXPORT DM_ExceptionHandler
(
  DM_ExceptionHandlerProc funcp,
  DM_UInt operation,
  DM_Options options
)
```

Parameter

-> DM_ExceptionHandlerProc funcp

Dieses ist ein Zeiger auf die Funktion, die als Exception-Handler installiert werden soll.

-> DM_Uint operation

In diesem Parameter wird die Aufgabe angegeben, die ausgeführt werden soll. Dabei gibt es folgende Werte:

DMF_RegisterHandler	Handlerfunktion soll installiert werden
DMF_EnableHandler	Handlerfunktion soll wieder aufgerufen werden
DMF_DisableHandler	Handlerfunktion soll nicht mehr ausgerufen werden
DMF_WithdrawHandler	Handlerfunktion soll abgemeldet werden

-> DM_Options options

Zur Zeit unbenutzt, muss 0 sein.

Rückgabewert

TRUE	Handlerfunktion erfolgreich installiert
FALSE	Handlerfunktion nicht installiert

Der Aufruf wird im logfile protokolliert. Der erste Aufruf wird mitgetraced. Folgende, rekursive Aufrufe werden **nicht** ins tracefile geschrieben da sonst Endlosschleifen entstehen können. Der ExceptionHandler wird nicht mehr aufgerufen bei einem assert in der assfail-Funktion. Das Trace/Logfile enthält file, line und assertion. Das Ende des Aufrufs wird ebenfalls mit protokolliert (mit Parameter DM_ExceptionInfo)

Sind mehrere Funktionen aktiv, dann werden diese nacheinander aufgerufen. Die Reihenfolge ist in umgekehrter Reihenfolge des Anmeldens. Die Funktionen arbeiten auf dem gleichen DM_ExceptionInfo. D.h. setzt eine Funktion eine message, dann kann sie durch eine weitere Funktion überschrieben werden. Ebenso showmessage.

Die zu installierende Funktion muss dabei wie folgt definiert sein:

```
typedef void (DML_default DM_CALLBACK *DM_ExceptionHandlerProc) __((DM_
ExceptionInfo *info));

typedef DM_ExceptionInfo
{
  // user info, not changeable
  DM_String   file;
  DM_Integer  line;
  DM_String   assertion;
  // output, changeable
  DM_String   message;      // default (char *)0
  DM_Boolean  showMessage;  // default TRUE
} DM_ExceptionInfo;
```

Die Einträge der Struktur entsprechen denen der assertion. Diese Werte sind nicht vom Benutzer änderbar.

message wird mit *NULL*-Pointer übergeben. Hier hat der Benutzer die Möglichkeit eine eigene Nachricht sich auszudenken, statt der Message "Contact your local ....". Ist message *NULL*-Pointer, dann wird weiterhin unsere Message ausgegeben.

showMessage gibt an, ob eine Nachricht ausgegeben werden soll.

3.23 DM_Execute

Mit Hilfe dieser Funktion kann aus dem Dialogskript ein anderes Programm gestartet werden. Je nach dem zugrundeliegendem Betriebssystem werden hier unterschiedliche Arten von zu startenden Programmen unterstützt.

```
DM_Boolean DML_default DM_EXPORT DM_Execute
(
  DM_String command,
  DM_String arguments,
  DM_Boolean synchronous,
  DM_Enum exetype,
  DM_Enum windowtype,
  DM_ID object,
  DM_Value * event,
  DM_Value * replydata,
  DM_Options options
)
```

Die Parameter entsprechen denen der Builtin-Funktion execute. Allerdings müssen von der C-Seite her alle Parameter mit einem Wert belegt sein.

Zusätzlich gibt es einen Parameter options, der zur Zeit unbenutzt ist und mit 0 belegt werden muss.

Siehe auch

Eingebaute Funktion execute im Handbuch „Regelsprache“

3.24 DM_FatalAppError

Diese Funktion muss aufgerufen werden, wenn die Anwendung auf einen Fehler gestoßen ist und nicht weitermachen kann. Diese Funktion beendet den Dialog Manager, schließt alle Dateien und Displays, und gibt die Kontrolle, wenn nötig, an die Anwendung zurück.

```
void DML_default DM_EXPORT DM_FatalAppError
(
  DM_String reason,
  DM_Int reaction,
  DM_Options options
)
```

Parameter

-> DM_String reason

Eine Meldung, die in das Tracefile eingetragen werden sollte. Das ist der Grund für die Beendung der Anwendung.

-> DM_Int reaction

Dies ist die Aktion, die vom Dialog Manager ausgeführt werden sollte. Die folgenden Werte sind gültig:

-1 Der Dialog Manager ruft die Funktion abort() auf, um einen Coredump zu erzeugen.

0 Der Dialog Manager schließt das Display und alle offenen Dateien und kehrt zur Anwendung zurück.

> 0 Der Dialog Manager schließt das Display und alle offenen Dateien und beendet sich mit dem angegebenen Wert.

Achtung

Beim Aufruf der Funktion **DM_FatalAppError** mit *reaction = 0* muss sichergestellt sein, dass die Kontrolle nicht mehr zum Dialog Manager zurückkehrt und dass auch keine DM-Funktion mehr aufgerufen wird.

-> DM_Options options

Unbenutzt, muss 0 sein.

Beispiel

Im eigentlichen Hauptprogramm des Dialog Managers in der Datei **startup.c** wird ein **DM_FatalAppError** aufgerufen, wenn das Hauptprogramm zum zweiten Mal innerhalb eines Prozesses aufgerufen wird.

```
int cdecl main __2(
(int, argc),
(char far * far *, argv))
{
```

```
  register int status;
  static char running = 0;

  if ((status = running++) == 0)
  {
    if ((status = DM_BootStrap(&argc, &argv)) == 0)
    {
      DM_InitOptions(&argc, argv, 0);

      status = AppMain (argc, argv);
      DM_ShutDown();
    }
    else
      DM_TraceMessage ("Bootstrap failed", DMF_LogFile);
  }
    else
      DM_FatalAppError ("Unexpected restart", -1, 0);
  return (status);
}
```

3.25 DM_FmtDefaultProc

Diese Funktion übernimmt alle anfallenden Aufgaben bei der Bearbeitung eines editierbaren Textes wie das Setzen des Formats, die Eingabeüberprüfung, die Navigation usw., sobald einem editierbaren Text ein Format gesetzt wird.

Normalerweise wird dann vom Dialog Manager automatisch diese Default-Format-Funktion aufgerufen, ohne dass die eigentliche Anwendung involviert wird.

Hat der editierbare Text aber eine eigene Format-Funktion, ruft der Dialog Manager anstatt der Default-Format-Funktion diese anwendungsspezifische Funktion auf. Diese muss dann die vom Dialog Manager geforderten Aufgaben erfüllen.

Dazu kann sie die Funktion DM_FmtDefaultProc mit einzelnen Tasks aufrufen, falls diese nicht anwendungsspezifisch geändert werden soll.

```
DM_boolean DML_default DM_EXPORT DM_FmtDefaultProc
(
  DM_FmtRequest *req,
  DM_FmtFormat *fmt,
  DM_FmtFormatDef **fmtDef,
  DM_FmtContent *cont,
  DM_FmtContentDef **contDef,
  DM_FmtDisplay *dpy
)
```

Parameter

-> DM_FmtRequest *req

Hier wird die Anforderung an die Formatierungsroutine übergeben. Dazu wird ein Strukturelement mit der Task belegt und je nach Aufgabe eine Union mit den erforderlichen Daten gefüllt (z.B. mit dem neuen Inhalt als String beim Setzen eines neuen Inhalts).

<-> DM_FmtFormat *fmt

Diese Struktur enthält Beschreibungsdaten über den Formatstring, die unabhängig von der Format-Funktion und der gewählten Formatart benötigt werden.

<-> DM_FmtFormatDef **fmtDef

Mit diesem Parameter wird eine Struktur durchgereicht, die die formatspezifischen Daten zu einem Formatstring enthält. Soll eine fremde Formatfunktion auf die Defaultformate zurückgreifen, muss sie hierfür in ihren privaten Daten einen Eintrag vorsehen und diese hier übergeben.

<-> DM_FmtContent *cont

Diese Struktur enthält Beschreibungsdaten über den zu formatierenden Inhaltsstring, die unabhängig von der Formatfunktion und der gewählten Formatart benötigt werden.

<-> DM_FmtContentDef **contDef

Mit diesem Parameter wird eine Struktur durchgereicht, die die formatspezifischen Daten zu einem Inhaltsstring enthält. Soll eine fremde Formatfunktion auf die Defaultformate zurückgreifen,

muss sie hierfür in ihren privaten Daten einen Eintrag vorsehen und diese hier übergeben.

<-> DM_FmtDisplay *dpy

Diese Struktur enthält Beschreibungsdaten über den formatierten Darstellungsstring, die unabhängig von der Formatfunktion und der gewählten Formatart benötigt werden.

Beispiel

Realisierung einer Formatfunktion, die nur wenige Aufgaben selber übernimmt. Die Funktion sorgt dafür, dass der Anwender nur ein Datum eingeben kann.

```
/*
** Abhängig von der Cursorpostion ist die Zahl, die eingegeben
** werden kann, damit sich ein gültiges Datum ergibt.
*/
char GetMax __1(
(DM_FmtDisplay far *, dpy))
{
  char max;

  switch (dpy->curpos)
  {
    case 0:
      max = '3';
    break;
    case 1:
      if (dpy->string[0] == '3')
        max = '1';
      else
        max = '9';
    break;
    case 3:
      max = '1';
    break;
    case 4:
      if (dpy->string[3] == '1')
        max = '2';
      else
        max = '9';
    break;
    default:
    max = '9';
    break;
  }
  return (max);
}

/*
```

```
** eigentliche Formatfunktion
**
*/
DM_Boolean DML_c DM_CALLBACK My_Formatter __6(
(DM_FmtRequest far *, req),
(DM_FmtFormat far *, fmt),
(FPTR *, fmtPriv),
(DM_FmtContent far *, cont),
(FPTR *, contPriv),
(DM_FmtDisplay far *, dpy))
{
  DM_Boolean retval;

  switch (req->task)
  {
    /*
    ** Nur Datum im Format dd.mm.yy soll eingebbar sein
    ** Diese Funktion ist nur rudimentär realisiert,
    ** BackSpace und DEL sollten auch abgefangen werden
    */
    case FMTK_modify:
    {
      char max = GetMax(dpy);

      if ((!req->targs.modify.strlength
      && (dpy->curpos == dpy->length))
      || ((req->targs.modify.strlength == 1)
        && (req->targs.modify.string[0] >= '0')
        && (req->targs.modify.string[0] <= max)))
      {
        retval = DM_FmtDefaultProc(req, fmt,
            (DM_FmtFormatDef **) (FPTR) fmtPriv, cont,
             (DM_FmtContentDef **) (FPTR) contPriv, dpy);
      }
      else
        retval = FALSE;
    }
    break;

    default:
    /*
    ** Aufruf der Default-Formatfunktion
    */
    retval = DM_FmtDefaultProc(req, fmt, (DM_FmtFormatDef **)
        (FPTR) fmtPriv,cont,
        (DM_FmtContentDef **) (FPTR) contPriv, dpy);
```

```
    break;
  }
  return (retval);
}
```

Siehe auch

Ressource format

3.26 DM_Free

Mit Hilfe dieser Funktion kann Speicherplatz wieder freigegeben werden, der über DM_Malloc oder DM_Realloc allokiert wurde.

```
void DML_default DM_EXPORT DM_Free
(
  DM_Pointer ptr
)
```

Parameter

-> DM_Pointer ptr

Pointer auf den Speicherbereich, der freigegeben werden soll.

Beispiel

```
char *string;
if ((string = (char *) DM_Malloc (5)))
{
  strcpy (string, "1234");
  ...
  DM_Free (string);
}
```

3.27 DM_FreeContent

Mit Hilfe dieser Funktion können Sie den von der Funktion DM_GetContent allokierten Speicherplatz wieder freigeben, d.h. jedem Aufruf von DM_GetContent muss DM_FreeContent folgen, nachdem Sie die Daten in Ihrer Anwendung verarbeitet haben.

```
void DML_default DM_EXPORT DM_FreeContent
(
  DM_Content *content
  DM_Options options
)
```

Parameter

-> DM_Content *content

Dieser Parameter ist ein Zeiger auf die Inhaltsstruktur eines Objekts (z.B. einer Listbox), dessen Speicherplatz freigegeben werden soll. Diesen Zeiger haben Sie von der Funktion DM_GetContent erhalten.

-> DM_Options options

Unbenutzt. Muss 0 sein.

Beispiel: Abfrage des Inhaltes eines Tablefields

```
void DML_default DM_ENTRY GetContent  __1(
(DM_ID, tablefieldID))
{
  int i;
  DM_UInt count;
  DM_Content *vec;
  DM_GetContent(tablefieldID, (DM_Value *) 0, (DM_Value *) 0,
     &vec, &count, 0);
  for (i = 0; i < count; i++)
  {
    printf("vec[%d].sensitive = %d\n",i, vec[i].sensitive);
    printf("vec[%d].string    = %s\n",i, vec[i].string);
  }
  DM_FreeContent(vec, 0);
}
```

Siehe auch

Objekte poptext, listbox, tablefield, treeview

3.28 DM_FreeVectorValue

Mit Hilfe dieser Funktion kann der vom Dialog Manager beim Aufruf von DM_GetVectorValue allokierte Speicherplatz wieder freigegeben werden.

```
void DML_default DM_EXPORT DM_FreeVectorValue
(
  DM_VectorValue *values,
  DM_Options options
)
```

Parameter

-> DM_VectorValue *values

In diesem Parameter muss der vom Dialog Manager erhaltene Vektor übergeben werden.

-> DM_Options options

Unbenutzt. Muss 0 sein.

Beispiel

Abfrage des Tablefield-Inhalts und Freigabe des Attributvektors.

```
DM_Boolean DML_default DM_ENTRY ReplaceName__3(
    (DM_ID, table),
    (char *, from),
    (char *, to))
{
  DM_VectorValue *oldData;
  DM_Value count;
  DM_Value lastidx;
  DM_boolean retval = true;

  if (!DM_GetValue(table, AT_rowcount, 0, &count, 0)
      || (count.type != DT_integer))
    return FALSE;

  lastidx.type = DT_index;
  lastidx.value.index.first = count.value.integer;
  lastidx.value.index.second = 2;

  if DM_GetVectorValue(table, AT_content, (DM_Value *) 0,
      &lastidx, &oldData, 0))
    return FALSE;

  DM_FreeVectorValue(oldData, 0);

  return retval;
}
```

Siehe auch

Objekte poptext, listbox, tablefield, treeview

Benutzerdefinierte Attribute

3.29 DM_GetArgv

Diese interne Funktion wandelt die Argumente des Programmaufrufs in die intern verwendete Codepage um. Sie darf nur genau einmal vor dem Aufruf von DM_BootStrap aufgerufen werden (siehe auch die mitgelieferte startup.c bzw. IDMuser.h).

Notwendig ist diese Funktion für die Unicode-Unterstützung

```
char ** DML_default DM_EXPORT DM_GetArgv
(
  int *argc,
  char ** argv,
  DM_Options options
)
```

Parameter

-> int *argc

In diesem Parameter wird ein Zeiger auf die Anzahl der Kommandozeilen-Argumente übergeben.

-> char ** argv

Vektor auf die Argumentliste.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Bitte geben Sie hier deshalb eine 0 an.

Rückgabewert

Die Argumentliste konvertiert in die vom Dialog Manager verwendete Applikationscodepage.

Anmerkung

Die Funktion ist nur für die Verwendung in der **startup.c** vorgesehen und ihr Aufruf darf auch nur dort erfolgen (siehe auch das mitgelieferte Modul **startup.c**).

3.30 DM_GetContent

Mit Hilfe dieser Funktion kann von der Anwendung aus der aktuelle Inhalt des Objekts (Listbox, Poptext oder Tablefield) mit einem einzigen Funktionsaufruf abgefragt werden. Auf diese Art und Weise kann also mit einem Zugriff der aktuelle Zustand (Inhalt und selektierte Einträge) erfragt werden.

```
DM_Boolean DML_default DM_EXPORT DM_GetContent
(
  DM_ID object,
  DM_Value *firstindex,
  DM_Value *lastindex,
  DM_Content **contentvec,
  DM_UInt *count,
  DM_Options options
)
```

Parameter

-> DM_ID object

Dieser Parameter bezeichnet das Objekt, dessen Inhalt erfragt werden soll.

-> DM_Value *firstindex

Steuert, welcher Bereich des Inhalts durch diese Funktion erfragt werden soll. In diesem Parameter wird der Startpunkt des Bereichs definiert. Dabei werden bei der Abfrage des Inhalts einer Listbox oder eines Poptexts der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Startwert belegt. Bei einem Tablefield hingegen wird der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Startwert belegt. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Startpunkt folgende Defaults:

listbox	integer = 1
poptext	integer = 1
tablefield	index.first = 1, index.second = 1
treeview	integer = 1

-> DM_Value *lastindex

Über diesem Parameter wird gesteuert, welcher Bereich des Inhalts durch diese Funktion erfragt werden soll. In diesem Parameter wird der Endpunkt des Bereichs definiert. Dabei werden bei der Abfrage des Inhalts einer Listbox oder eines Poptexts der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Endewert belegt. Bei einem Tablefield hingegen wird der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Endewert belegt. Hierbei wird in index.first die Zeile, in index.second die Spalte

eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Endpunkt folgende Defaults:

listbox	integer = Object.itemcount
poptext	integer = Object.itemcount
tablefield	index.first = object.rowcount, index.second = object.colcount
treeview	Integer = Object.itemcount

<- DM_Content **contentvec

In diesem Pointer wird der Zeiger auf die vom Dialog Manager gefüllte Struktur zurückgegeben. Diese Struktur darf dann auf keinen Fall von der Anwendung verändert werden.

<- DM_UInt *count

In diesem Parameter liefert der Dialog Manager zurück, wie viele Einträge durch die Angabe der Indizes erfasst werden und damit, wie groß die zurückgelieferte Struktur *contentvec* ist.

-> DM_Options options

Über diesen Parameter wird gesteuert, welche Informationen der Dialog Manager zurückliefern soll. Dabei gibt es die folgenden Möglichkeiten, die auch kombiniert (bitweises "oder") werden dürfen.

Option	Bedeutung
DMF_OmitActive	Diese Option besagt, dass in dem Vektor das Attribut *.active* nicht mit Werten versorgt werden soll, da sich die Anwendung für dieses Attribut nicht interessiert.
DMF_OmitSensitive	Diese Option besagt, dass in dem Vektor das Attribut *.sensitive* nicht mit Werten versorgt werden soll.
DMF_OmitStrings	Diese Option besagt, dass in dem Vektor das Attribut *.content* nicht mit Werten versorgt werden soll, da sich die Anwendung für dieses Attribut nicht interessiert. Wenn die Strings wirklich nicht benötigt werden, bringt das Setzen dieser Option wesentliche Performanz-Vorteile.
DMF_OmitUserData	Diese Option besagt, dass in dem Vektor das Attribut *.userdata* nicht mit Werten versorgt werden soll, da sich die Anwendung für dieses Attribut nicht interessiert. Wenn die Userdata wirklich nicht benötigt werden und das Objekt je Eintrag eine Userdata hat,, bringt das Setzen dieser Option wesentliche Performanz-Vorteile.

Hinweis

Den Inhaltsvektor, den Sie im Parameter *contentvec* erhalten, wird vom Dialog Manager allokiert und muss unbedingt über die Funktion DM_FreeContent wieder freigegeben werden.

Beispiel

Abfrage der Zeilen 2 bis 5 in einer Listbox.

```
void DML_default DM_ENTRY GetContent __1((DM_ID, lb))
{
  int i;
  DM_Integer count;
  DM_Value first, last;
  DM_Content *vec;

  first.type = DT_integer;
  first.value.integer = 2;

  last.type = DT_integer;
  last.value.integer = 5;

  DM_GetContent(lb, &first, &last, &vec, &count, 0);
  for (i = 0; i < count; i++)
  {
    printf("vec[%d].sensitive = %d\n",i, vec[i].sensitive);
    printf("vec[%d].active    = %d\n",i, vec[i].active);
    printf("vec[%d].string    = %s\n",i, vec[i].string);
  }

  DM_FreeContent(vec, 0);
}
```

Siehe auch

Objekte listbox, poptext, tablefield, treeview

3.31 DM_GetMultiValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, mehrere Attribute von einem oder verschiedenen DM-Objekten in einem Funktionsaufruf zu erfragen. Diese Funktion sollte daher v.a. im Zusammenhang mit dem Verteilten Dialog Manager eingesetzt werden, da sie die Netzwerkbelastung deutlich reduziert.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_GetMultiValue
(
  DM_MultiValue * values,
  DM_UInt count,
  DM_ID dialogID,
  DM_String pathname,
  DM_Options options
)
```

Parameter

<-> DM_MultiValue *values

Liste von Attributen und Objekten, die erfragt werden sollen. Ist dabei das Element in der Struktur für das Objekt auf 0 gesetzt, wird das im Parameter *pathname* beschriebene Objekt genommen. Die Liste muss dabei mindestens die im Parameter *count* angegebene Länge haben.

-> DM_UInt count

Dieser Parameter bezeichnet die Länge des im Parameter *values* angegebenen Objekt-Attribut-Vektors.

-> DM_ID dialogID

Dieser Parameter beschreibt den Dialog, zu dem die angegebenen Objekte gehören. Muss nur dann angegeben werden, wenn die ID des Objektes dessen Attribute erfragt werden sollen, nicht bekannt ist und daher der Name des Objektes im Parameter *pathname* angegeben ist.

-> DM_String pathname

Dieser Parameter bezeichnet das Objekt, dessen Attribute erfragt werden sollen. Er ist nur dann belegt, wenn die interne ID des Objekts noch nicht bekannt ist.

-> DM_Options options

Unbenutzt - muss 0 sein.

Rückgabewert

TRUE	Attribute konnten erfolgreich erfragt werden.
FALSE	Mindestens ein Attribut konnte nicht erfragt werden.

Beispiel

Die nachfolgende C-Funktion soll Koordinaten bei verschiedenen Objekten abfragen.

```
void DML_default DM_ENTRY Get __3((DM_ID, o1),
                  (DM_ID, o2),
                  (DM_ID, o3))
{
  DM_MultiValue val[3];

  /* Setzen der jeweiligen Objekt-ID */
  val[0].object = o1;
  val[1].object = o2;
  val[2].object = o3;
/* Setzen des jeweiligen Index-Typs */
  val[0].index.type = DT_void;
  val[1].index.type = DT_void;
  val[2].index.type = DT_void;
/* Setzen des jeweiligen Attributes */
  val[0].attribute = AT_xleft;
  val[1].attribute = AT_width;
  val[2].attribute = AT_xright;

  DM_GetMultiValue(val, 3, dialogID, (char *)0, 0);

  if (val[0].data.type == DT_integer)
    printf("%d %d %d\n",
        (int) val[0].data.value.integer,
        (int) val[1].data.value.integer,
        (int) val[2].data.value.integer);
}
```

3.32 DM_GetToolkitData

Mit Hilfe dieser Funktionen können fenstersystemspezifische Daten abgefragt werden.

```
FPTR DML_default DM_EXPORT DM_GetToolkitData
(
  DM_ID objectID,
  DM_Attribute attr
)
```

Parameter

-> DM_ID objectID

Dieser Parameter ist der Identifikator des Objekts, dessen fenstersystemspezifischen Daten erfragt werden sollen.

-> DM_Attribute attr

Dieser Parameter gibt das Attribut an, das abgefragt werden soll.

Die gültigen Belegungen des Parameters *attr* sind fenstersystemabhängig und können den nachfolgenden Kapiteln entnommen werden.

Rückgabewert

Je nach Art des abgefragten Wertes liefert diese Funktion die entsprechenden auf *FPTR* konvertierten Werte.

Anmerkungen

Die Funktion **DM_GetToolkitData** existiert nicht für die Serverseite und sollte dort nicht aufgerufen werden!

3.32.1 Motif

Mit Hilfe dieser Funktion können die für X-Windows notwendigen Daten wie „window-id“, „widget“ und „color“ abgefragt werden. Die Bedeutungen dieser Datentypen werden im entsprechenden X-Windows-Handbuch erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	Bedeutung
AT_CanvasData	Dieser Wert liefert die benutzerspezifischen Daten einer Canvas. Diese Daten wurden von einer Canvas-Callback-Funktion gesetzt und beinhalten benutzerspezifische Daten (siehe auch Kapitel „Strukturen für Canvas-Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“).

Attribut	Bedeutung
AT_IsNull	Liefert einen Wert<>*0* wenn die Schriftart ein *UI_NULL_FONT* ist.
AT_Tile	Siehe auch AT_XTile
AT_XColor	Dieser Wert liefert die X-Windows-spezifische Struktur für die angegebene Farbe zurück. Der Rückgabewert ist dann vom Typ „Pixel“. Das angegebene Objekt muss eine Farbe sein.
AT_XColormap	Dieser Wert liefert die Colormap des Default-Screens zurück.
AT_XCursor	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Cursor. Der Rückgabewert ist dann vom Typ „Cursor“. Das angegebene Objekt muss ein Cursor sein.
AT_XDepth	Dieser Wert liefert die Farbtiefe des Default-Screens als *int*.
AT_XDisplay	Dieser Wert liefert die Anzeige des angegebenen Dialogs. Der Rückgabewert der Funktion ist dann vom Typ „Display *“.
AT_XFont	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XFontStruct *“, sofern zur Font-Definition passend. Andernfalls NULL.
AT_XFontSet	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XFontSet“, sofern zur Font-Definition passend. Andernfalls NULL.
AT_XmFontList	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XmFontList“, sofern zur Font-Definition passend. Andernfalls NULL.
	Liefert die Default-Schriftart die typischerweise vom IDM bei einem nicht gesetzten font-Attribut verwendet wird.

Attribut	Bedeutung
AT_XScreen	Dieser Wert liefert den Bildschirm zu dem angegebenen Dialog. Der Rückgabewert der Funktion ist dann vom Typ „Screen“.
AT_XShell	Dieser Wert liefert das Shell-Widget eines Objekts. Der Rückgabewert der Funktion ist ein Widget.
AT_XtAppContext	Liefert den Application Context. Returns the Application Context.
AT_XTile	Dieser Wert liefert die X-Windows-spezifische Struktur eines Musters „tile“ zurück. Der Rückgabewert der Funktion ist dabei davon abhängig, wie das Muster definiert worden ist. Wurde es in einer externen Datei im „gif“-Format hinterlegt, wird ein „XImage*“ zurückgeliefert; wurde es direkt in der Dialog Manager Datei definiert, wird ein „Pixmap“ zurückgegeben. Es wird empfohlen AT_XTile über die Funktion **DM_GetToolkitDataEx** abzufragen.
AT_XVisual	Dieser Wert liefert eine Visual Struktur für den Default-Screen.
AT_XWidget	Dieser Wert liefert das Widget zu dem angegebenen Objekt. Der Rückgabewert der Funktion ist dann vom Typ „Widget“.
AT_XWindow	Dieser Wert ist das zu gehörende Fenster. Der Rückgabewert der Funktion ist dann vom Typ „Window“.

Zu Beachten bei Multiscreendialogen

Der Aufruf mit AT_ XTile bzw. AT_XColor liefert immer nur das tile bzw. die color des default Screen zurück (siehe auch Kapitel „Multiscreen Ssupport untder Motif“ im Handbuch „Programmiertechniken“).

Beispiel

```
int DML_c AppMain (argc, argv)
int argc;
char far * far *argv;
{
  DM_ID dialogID;
  Widget toplevel;    /* Application shell erhalten von
                         XtInitialize(). */
```

```
static char * dialogfile = "xres.dlg";

/* Initialize the dialog manager */
DM_Initialize (&argc, argv, 0);

dialogID = DM_LoadDialog (dialogfile, 0);

if (!dialogID)
{
  DM_TraceMessage("%s: Could not load dialog \"%s\"",
      DMF_LogFile | DMF_InhibitTag | DMF_Printf,
      argv[0], dialogfile);
  return(1);
}


/* Abfrage der application shell */
if ((toplevel = (Widget) DM_GetToolkitData (dialogID,
  AT_XWidget)) != 0)
{
  DM_ID my_bgc;
  DM_ID my_font;
  DM_ID my_FontOfExit;

  /* weitere, eigene Statements */ }
```

3.32.2 Microsoft Windows

Mit Hilfe dieser Funktion können die für MICROSOFT WINDOWS notwendigen Daten erfragt werden, wie „window handle“, „instance handle“ und „color“.

Die Bedeutung dieser Datentypen wird im entsprechenden Microsoft Windows Handbuch erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	Objekt	Rückgabewert	Bedeutung
AT_CanvasData	canvas	FPTR	Über dieses Attribut können die benutzerspezifischen Daten eines Canvas-Objekts erfragt werden. Diese Daten wurden von DM_ SetToolkitData oder einer Canvas-Callback-Funktion gesetzt und beinhalten jegliche benutzerspezifischen Daten (Siehe auch Kapitel „Strukturen für Canvas-Funktionen“).
AT_ClipboardText	setup oder 0	DM_String	Dieses Attribut gibt den Textinhalt des Microsoft Windows Clipboards zurück. Der Rückgabewert wird gepuffert und ist bis zur erneuten Abfrage des Attributes gültig. Auch ein Setzen des Attributes macht den Puffer ungültig. Um den String ohne Änderung des Clipboards freizugeben, dient der Aufruf: `DM_SetToolkitData(0, AT_ ClipboardText, (FPTR) 0, 0);`
AT_Color	color	COLORREF	Siehe auch AT_XColor
	tile	HPALETTE	Siehe auch AT_XColor
AT_DataType	tile	int	Die Abfrage ist nur noch aus Kompatibilitätsgründen vorhanden. Es sollte mit DM_GetToolkitDataEx das Attribute AT_Tile mit gesetztem "data" Parameter verwendet werden. Dieses Attribut gibt den Typ des Musters zurück. Die Zuordnung ist bei AT_XTile beschrieben.
AT_DPI	IDM Objekte	int	Siehe auch AT_GetDPI
AT_Font	font	HFONT	Siehe auch AT_XFont
	IDM Objekte	HFONT	Siehe auch AT_XFont
	setup	HFONT	Siehe auch AT_XFont

Attribut	Objekt	Rückgabewert	Bedeutung
AT_GetDPI	setup oder 0	int	Dieses Attribut gibt den System DPI Wert zurück. Siehe Hinweis unten.
	IDM Objekte	int	Dieses Attribut gibt den DPI Wert des Objektes zurück. Dieser ist abhängig zu welchem Monitor das Objekt zugeordnet ist. Siehe Hinweis unten.
AT_IsNull	font oder color	int	Hiermit kann abgefragt werden, ob die Resource auf *NULL* definiert wurde (*UI_NULL_FONT* bzw. *UI_NULL_COLOR*). Die Resource wurde auf *NULL* definiert, wenn der Rückgabewert nicht 0 ist. Hereby it can be queried whether the resource was defined to *NULL* (*UI_NULL_FONT* or *UI_NULL_COLOR*). The resource has been defined to *NULL* if the return value is not *0*.
AT_maxsize	setup oder 0	int	Dieses Attribut gibt die Anzahl der noch freier WSI-ID's zurück. **Achtung:** Die Anzahl noch verfügbarer WSI-IDs sagt nichts darüber aus, wie viele Objekte tatsächlich noch sichtbar gemacht werden können! Es ist die maximale Obergrenze.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_Raster	dialog editbox groupbox layoutbox module notebook notepage spinbox splitbox statusbar tablefield toolbar Window	DWORD	Dieses Attribut gibt die Größe des am Objekt definierten Rasters in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
	font	DWORD	Dieses Attribut gibt die Größe der Schriftart in IDM Pixel, wie sie für die Rasterberechnung verwendet wird, zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Sroll-barDimension	groupbox notepage window	DWORD	Dieses Attribut gibt die Breite der senkrechten und die Höhe der waagrechten Scrollbar in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_Size	dialog module	DWORD	Dieses Attribut gibt die Größe des Arbeitsbereichs des primären Monitors in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
	font	DWORD	Dieses Attribut gibt die Größe der Schriftart in IDM Pixel zurück. Die Breite wird anhand des Referenzstrings berechnet, falls einer angegeben ist. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
	Übrige IDM Objekteaußer menubox, menuitem und menuseparator	DWORD	Dieses Attribut gibt die Größe des Objektes in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Tile	color	HBRUSH	Siehe auch AT_XTile
	tile	HANDLE	Die Abfrage ist nur noch aus Kompatibilitätsgründen vorhanden. Es sollte DM_GetToolkitDataEx mit gesetztem "data" Parameter verwendet werden. Siehe auch AT_XTile
AT_toolhelp	setup oder 0	HWND	Dieses Attribut gibt den Microsoft Windows Handle des tooltip Controls zurück, welches vom Dialog Manager zur Darstellung des *.tool-help* Attributs verwendet wird. Siehe auch „Beispiel zu AT_toolhelp am setup Objekt“ weiter unten.
AT_value	RTF edittext	DM_String	Dieses Attribut gibt den vollständigen Inhalt, also mit allen Formatierungsanweisungen usw., eines RTF-Eingabefeldes zurück.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_VSize	IDM Objekte außer menubox, menuitem und menuseparator	DWORD	Dieses Attribut gibt die virtuelle Größe des Objektes in IDM Pixel zurück. Gibt es keine virtuelle Größe, dann wird die reale Größe in IDM Pixel zurückgegeben. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Widget	USW	HWND	Siehe auch AT_XWidget
AT_WinHandle	dialog module setup oder 0	HINSTANCE	Dieses Attribut gibt den Microsoft Windows Handle der Anwendungsinstanz zurück.
	menubox	HMENU	Dieses Attribut gibt den Microsoft Windows Menu Handle zurück.
	menuitem menuseparator	HMENU	Dieses Attribut gibt den Microsoft Windows Menu Handle des umgebenden ***menubox*** Objekts zurück.
	Übrige IDM Objekte	HWND	Dieses Attribut gibt den Microsoft Windows Handle des Objekts zurück. Die Gruppierungsobjekte (***groupbox***, ***notepage***, ***window***, …) sind meist aus mehreren Microsoft Windows Objekten aufgebaut, bei diesen wird das Handle des „Client“-Fensters (das Fenster in dem die Kindobjekte angelegt werden) zurückgegeben.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_wsidata	cursor	HCURSOR	Siehe auch AT_XCursor
	font	HFONT	Siehe auch AT_XFont
	tile	HANDLE	Die Abfrage ist nur noch aus Kompatibilitätsgründen vorhanden. Es sollte DM_GetToolkitDataEx mit gesetztem "data" Parameter verwendet werden. Siehe auch AT_XTile
	Übrige IDM Objekte	HWND	Dieses Attribut gibt den Microsoft Windows Handle des äußeren Microsoft Windows Objektes zurück, das innere kann mit „AT_WinHandle“ erfragt werden.

AT_XColor	color	COLORREF	Dieses Attribut gibt die Microsoft Windows-spezifische Struktur für die angegebene Farbe zurück. Der Zugriff auf die Farbwerte erfolgt mit den entsprechenden Makros von Microsoft Windows:
			`COLORREF u1RGB = (COLORREF) (size_t) DM_GetToolkitDataEx (colorID, AT_XColor, (FPTR) 0, 0);`
			`BYTE ucRed = GetRValue (u1RGB);`
			`BYTE ucGreen = GetGValue (u1RGB);`
			`BYTE ucBlue = GetBValue (u1RGB);`

	tile	HPALETTE	Dieses Attribut gibt die Microsoft Windows Farbpalette, die von dem Muster verwendet wird, zurück.
AT_XCursor	cursor	HCURSOR	Dieses Attribut gibt den Microsoft Windows Cursor Handle zurück.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_XFont	font	HFONT	Dieses Attribut gibt den Microsoft Windows Font Handle zurück.
	IDM Objekte	HFONT	Dieses Attribut gibt den Microsoft Windows Font Handle der an diesem Objekt verwendeten Schriftart zurück.
	setup	HFONT	Es wird der Font-Handle der verwendeten Standardschriftart zurück geliefert.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_XTile	color	HBRUSH	Dieses Attribut gibt eine Microsoft Windows Brush der Farbe zurück. Diese Brush kann zum Füllen des Hintergrunds verwendet werden.

Attribut	Objekt	Rückgabewert	Bedeutung
	tile	HANDLE	Die Abfrage von AT_wsidata, AT_Tile und AT_XTile ohne Angabe des "data" Parameters ist nur noch aus Kompatibilitätsgründen vorhanden. Es sollte der "data" Parameter verwendet werden. Dieses Attribut gibt die Microsoft Windows spezifische Struktur für das Muster (tile) wie in Version A.06.03.a und früher zurück. Hierzu müssen GDI Objekte zusätzlich angelegt werden. Um die neuen Microsoft Direct2D Daten, die der IDM intern verwendet, zu erhalten, muss der "data" Parameter gesetzt werden. Der Typ des Microsoft Windows Handle ist abhängig von AT_DataType, wobei AT_DataType erst nach der Abfrage von AT_wsidata, AT_Tile oder AT_XTile abgefragt werden darf: » *DMF_TlkDataIsIcon*: HICON » *DMF_TlkDataIsWMF*: HMETAFILE » *DMF_TlkDataIsEMF*: HENHMETAFILE » Sonst: HBITMAP **WICHTIG**: Die zurückgelieferten Daten sollten nicht gespeichert werden, da diese automatisch freigegeben werden, wenn die tile Resource von keinem sichtbaren IDM Objekt mehr verwendet wird. **Hinweis**: Wenn mittels DM_SetToolkitData Daten gesetzt wurden, dann werden die gesetzten und nur die gesetzten Daten zurückgeliefert.

Attribut	Objekt	Rückgabewert	Bedeutung
AT_XWidget	USW	HWND	Dieses Attribut gibt den Microsoft Windows Handle des USW Objektes zurück.

Hinweis zum Objekt und Attribut

Das beim Aufruf angegebene Objekt muss im Allgemeinen sichtbar und damit im WSI angelegt sein, damit die zurückgegebenen Daten Sinn machen. Ressourcen werden in der Regel beim Aufruf angelegt. Wird ein Objekttyp angegeben, der beim betreffenden Attribut nicht erwähnt ist, wird in der Regel eine Fehlermeldung in die Logdatei geschrieben und „(FPTR) 0“ zurückgegeben.

Hinweis zum Zugriff auf den Rückgabewert

Der Rückgabewert der Funktion ist ein „FPTR“ bzw. „void *“, dieser muss auf den dokumentierten Rückgabewert gecastet werden, um beim Kompilieren keine Warnungen zu erhalten.

Da ein „void *“ Zeiger auf jeden beliebigen anderen Zeiger gecastet werden kann, reicht bei allen Zeiger-Datentypen ein einfacher Castoperator. Zu den Zeiger-Datentypen zählen zum Beispiel sämtliche Microsoft Windows Handle, wie zum Beispiel „HWND“, „HFONT“, ... :

```
HWND hwnd = (HWND) DM_GetToolkitData(idObj, AT_wsidata);
```

Bei Zahlenwerten muss meist ein Zwischencast eingefügt werden, da die Größe des Datenwerts beim Cast von einem Zeiger zu einer Zahl erhalten bleiben muss, um Warnungen zu vermeiden. Hierzu kann der Datentyp „size_t“ verwendet werden, da dieser per Definition dieselbe Größe wie ein Zeiger besitzt. Anschließend kann dann auf einen kleineren Zahlentyp gecastet werden:

```
DM_UInt2 val = (DM_UInt2) (size_t) DM_GetToolkitData(idObj, AT_wsidata);
```

Hinweis Breite und Höhe in „DWORD“ gepackt

Unter Microsoft Windows werden häufig Breite und Höhe in eine „DWORD“ gepackt. Dies wird vom IDM zum Teil auch so gehandhabt. Die einzelnen Werte können dann mit den Microsoft Windows Makros „LOWORD“ und „HIWORD“ extrahiert werden:

```
DWORD size = (DWORD) (size_t) DM_GetToolkitData(id, AT_Size);
WORD width = LOWORD(size);
WORD height = HIWORD(size);
```

Hinweis IDM Pixel

Der IDM für WINDOWS 11 unterstützt hohe Auflösungen. Um Auswirkungen auf bestehende Dialogskripte so gering wie möglich zu halten, werden vom ISA DIALOG MANAGER virtuelle Pixelkoordinaten verwendet. Diese basieren auf der Größe einer Anwendung, die hohe Auflösungen nicht unterstützt, wie zum Beispiel IDM für WINDOWS 10.

Hinweis zu DPI-Werten

Achtung, alle DPI Werte sind dynamisch und können durch den Anwender geändert werden. Zum Beispiel können IDM Objekten auf einen anderen Monitor verschoben werden oder der Anwender stellt über die Systemsteuerung andere Vergrößerungsfaktoren ein.

Ist die Anwendung nicht DPI Aware (zum Beispiel IDM für WINDOWS 10) dann wird immer der Standard DPI Wert von 96 verwendet.

Hinweiszu NULL-Werten bei Resourcen

Die Funktion **DM_GetToolkitData** liefert unter MICROSOFT WINDOWS bei den font- und color Resourcen einen **NULL** Wert zurück, wenn die Resource auf **UI_NULL_FONT** bzw. **UI_NULL_COLOR** definiert wurde. Dies betrifft folgende Attribute:

- *AT_wsidata*, *AT_Font*, *AT_XFont*:
  Der Rückgabewert ist bei *UI_NULL_FONT* dann *(HFONT) 0*.
- *AT_Color*, *AT_XColor*:
  Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(COLORREF) -1L*.
- *AT_Tile*, *AT_XTile*:
  Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(HBRUSH) 0*.

Hinweis IDM Pixel

Beispiel

Abfragen einer Canvas spezifischen Struktur. Falls die Struktur noch nicht an der Canvas definiert ist, allokieren des notwendigen Speichers und übergeben der Daten an die Canvas.

```
void DML_default DM_ENTRY HoleCanvasDaten __3(
(DM_ID, thisID),
(DM_ID, canvasID),
(DM_ID, stvarID))
{
  MyData *d;
  /* Holen der Daten aus der Canvas und Abprüfen, ob man sie
  * wirklich bekommen hat, sonst Allokieren der Struktur
  /* Fetching data from the canvas and checking whether it has
  * actually arrived. Otherwise allocating the structure.
  */
  if ((d = (MyData *) DM_GetToolkitData (canvasID,
     AT_CanvasData)) == (MyData *) 0)
      if ((d = DM_Calloc (sizeof(MyData)))
        == (MyData *) 0)
      return;
    else
      if (!DM_SetToolkitData (canvasID, AT_CanvasData, d, 0))
        DM_TraceMessage ("Cannot set canvas toolkit data",
```

```
            0);
}
```

Beispiel zu AT_toolhelp am setup Objekt

Hinweis

Die im Beispiel verwendeten Funktionen „CloseToolhelp“ und „SetToolhelpDisplayTime“ sind innerhalb des Dialogs entsprechend zu definieren.

Temporäres Schließen eines geöffneten Tooltip-Controls in einer Canvas-Funktion um die Canvas neu zu zeichnen

```
#include <windows.h>
#include <commctrl.h>
#include IDMuser.h

void DML_default DM_ENTRY CloseToolhelp __0() {
  DM_ID idSetup =
  DM_ParsePath((DM_ID) 0, (DM_ID) 0, "setup", 0, 0);

  if (idSetuo != (DM_ID) 0) {
    HWND hwndToolhelp =
     (HWND) DM_GetToolkitData(idSetup, AT_toolhelp);

    if (hwndToolhelp != (HWND) 0) {
      SendMessage(
         hwndToolhelp, TTM_POP,
         (WPARAM) 0, (LPARAM) 0);
    }
  }
}
```

Anzeigezeit des Tooltip ändern

```
#include <windows.h>
#include <commctrl.h>
#include IDMuser.h

void DML_default DM_ENTRY SetToolhelpDisplayTime __1(
    (DM_Integer, iDisplayTime)) {
  DM_ID idSetup =
      DM_ParsePath((DM_ID) 0, (DM_ID) 0, "setup", 0, 0);

    if (idSetuo != (DM_ID) 0) {
      HWND hwndToolhelp =
          (HWND) DM_GetToolkitData(idSetup, AT_toolhelp);
```

```
            if (hwndToolhelp != (HWND) 0) {
              SendMessage(
                  hwndToolhelp, TTM_SETDELAYTIME,
                  (WPARAM) TTDT_AUTOPOP,
                  (LPARAM) iDisplayTime);
            }
        }
    }
```

3.32.3 Qt

Folgende Werte sind für die Attribute zugelassen:

Attribut	Objekt	Rückgabewert	Bedeutung
AT_Application	0	FPTR auf QApplication	Über dieses Attribut kann die der Anwendung zu Grunde liegende QApplication erfragt werden.
AT_CanvasData	cancas	FPTR	Über dieses Attribut können die benutzerspezifischen Daten eines Canvas-Objekts erfragt werden. Diese Daten wurden von DM_SetToolkitData oder einer Canvas-Callback-Funktion gesetzt und beinhalten jegliche benutzerspezifischen Daten (Siehe auch Kapitel „Strukturen für Canvas-Funktionen“).

Attribut	Objekt	Rückgabewert	Bedeutung
AT_Color	color	QColor / QBrush	Über dieses Attribut kann die von der Color-Ressource verwendete Farbe oder Farbverlauf (als QBrush) erfragt werden. Achtung: Es sollte als erstes immer auf eine zulässige Farbe (QColor) geprüft werden, da sich ein QBrush automatisch in eine QColor casten lässt, welche dann jedoch eine uninitialisierte aber zulässige Farbe darstellt.
AT_Font	font	QFont	Über dieses Attribut kann die von der Font-Ressource verwendete QFont erfragt werden.
AT_FontName	font	char*	Über dieses Attribut kann der Name der von der Font-Ressource verwendeten QFont erfragt werden.
AT_Tile	tile	QPixmap	Über dieses Attribut kann das QPixmap von einem Muster (tile) erfragt werden.
AT_XTile	tile	QPixmap	Siehe AT_Tile
AT_XWidget	IDM Objekte	QWidget	Dieses Attribut ermittelt das zu einer DM_ID gehörende QWidget.

Siehe auch

Funktion DM_GetToolkitDataEx

3.33 DM_GetToolkitDataEx

Diese Funktion ist eine erweiterte Form von DM_GetToolkitData und erlaubt über die Parameter *data* und *options* die Übergabe zusätzlicher Werte bzw. Strukturen.

Verfügbarkeit

ab IDM-Version A.05.02.e

```
FPTR DML_default DM_EXPORT DM_GetToolkitDataEx
(
  DM_ID objectID,
  DM_Attribute attr,
  FPTR data,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter ist der Identifikator des Objekts, dessen fenstersystemspezifischen Daten erfragt werden sollen.

-> DM_Attribute attr

Dieser Parameter gibt das Attribut an, das abgefragt werden soll.

<-> FPTR data

Pointer auf die DM_ToolkitDataArgs-Struktur. Diese dient dabei zur Kommunikation mit der **DM_GetToolkitDataEx**-Funktion. Sie erlaubt die Übergabe und Rückgabe verschiedener Wertetypen.

Für die kompatible Verwendung zu DM_GetToolkitData sollte data = NULL sein.

-> DM_Options options

Zurzeit unbenutzt, sollte mit 0 verwendet werden.

Rückgabewert

Je nach Art des abgefragten Wertes liefert diese Funktion die entsprechenden auf *FPTR* konvertierten Werte.

Anmerkungen

Die Funktion **DM_GetToolkitDataEx** existiert nicht für die Serverseite und sollte dort nicht aufgerufen werden!

3.33.1 Motif

Mit Hilfe dieser Funktion können die für X-Windows notwendigen Daten wie „window-id“, „widget“ und „color“ abgefragt werden. Die Bedeutungen dieser Datentypen werden im entsprechenden X-Windows-Handbuch erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	data	Objekt	Bedeutung
AT_CanvasData	(FPTR) 0	canvas	Dieser Wert liefert die benutzerspezifischen Daten einer Canvas. Diese Daten wurden von einer Canvas-Callback-Funktion gesetzt und beinhalten benutzerspezifische Daten (siehe auch Kapitel „Strukturen für Canvas-Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“).
AT_CellRect	DM_Tool-kitDataArgs *data	tablefield	Ermittelt für ein Tablefield die Koordinaten und Größe einer Zelle in IDM-Pixel (hierzu muss das „data->index“ Feld auf den gewünschten Index gesetzt werden (*argmask = DM_TKAM_index*)). Diese werden im *rectangle*-Feld der übergebenen DM_Tool-kitDataArgs-Strukur abgelegt. Mit Zelle ist der rechteckige Bereich im Tablefield gemeint, in dem Schatten, Fokus- und Akti-vierungsrahmen und Text gezeichnet wer-den. Nicht zur Zelle gehören die zwischen den Zellen gezeichneten Linien. Die gelieferten Koordinaten sind relativ zur lin-ken oberen Ecke des Tablefield zu sehen. Wenn Position und Größe einer Zelle ermit-telt werden konnten, liefert **DM_GetTool-kitDataEx** den Zeiger auf die angegebene DM_ToolkitDataArgs-Struktur zurück, in allen anderen Fällen NULL. Position und Größe können nur ermittelt werden, wenn sowohl Tablefield als auch Zeile und Spalte der Zelle sichtbar geschaltet sind und die Zelle ganz oder teilweise im Tablefield sichtbar ist. Eine konkrete, absolute Unsichtbarkeit (z.B. weil das Fenster außerhalb des sichtbaren Bild-schirms liegt oder anderweitig verdeckt ist) kann trotz gelieferter Position und Größe nicht ausgeschlossen werden und ist fens-tersystemabhängig.

Attribut	data	Objekt	Bedeutung
AT_DPI	DM_Tool-kitDataArgs *data	0	Liefert - mit gesetztem argmask=0 - die DPI-Informationen des Default-Bildschirms (null-ObjektID) oder die DPI-Informationen des Bildschirms auf dem das angegebene Oberflächenobjekt sichtbar gemacht dargestellt wird. Die Funktion setzt die argmask um auf *DM_TKAM_dpi* | *DM_TKAM_scaledpi*. Im **dpi**-Feld der DM_ToolkitDataArgs befindet sich dann der Default-DPI-Wert. In der Sub-Struktur **scale.dpi** der DPI-Wert auf vom vom Default-DPI-Wert die Skalierung stattfindet. Das Element **scale.factor** liefert ausserdem den Skalierungsfaktor in %. Bei einer HiDPI-awaren IDM-Anwendung auf einem Bild-schirm mit 200% Skalierung findet man dann typischerweise die Werte **dpi**=96 ,**scale.dpi**=192, **scale.factor**=200 vor. Eine Umrechnung von Pixel-Koordinaten des IDM's in „reale Bildschirmpixel“ kann dann über ***scale.factor**/100 oder ***scale.dpi**/**dpi** erfolgen.
AT_IsNull	(FPTR) 0	font	Liefert einen Wert<>*0* wenn die Schriftart ein *UI_NULL_FONT* ist.
AT_ObjectID	DM_Tool-kitDataArgs *data	0	Ermittelt für ein Widget die entsprechende Objekt-ID. Wenn erfolgreich, wird der data-Pointer zurückgegeben und in der DM_Tool-kitDataArgs-Struktur ist die ID in der data-Substruktur gespeichert.
AT_Tile	(FPTR) 0	tile	Siehe auch AT_XTile
AT_XColor	(FPTR) 0	color	Dieser Wert liefert die X-Windows-spe-zifische Struktur für die angegebene Farbe zurück. Der Rückgabewert ist dann vom Typ „Pixel“. Das angegebene Objekt muss eine Farbe sein.
AT_XColormap	(FPTR) 0	0	Dieser Wert liefert die Colormap des Default-Screens zurück.

Attribut	data	Objekt	Bedeutung
AT_XCursor	(FPTR) 0	cursor	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Cursor. Der Rückgabewert ist dann vom Typ „Cursor“. Das angegebene Objekt muss ein Cursor sein.
AT_XDepth	(FPTR) 0	0	Dieser Wert liefert die Farbtiefe des Default-Screens als *int*.
AT_XDisplay	(FPTR) 0	0	Dieser Wert liefert die Anzeige des angegebenen Dialogs. Der Rückgabewert der Funktion ist dann vom Typ „Display *“.
AT_XFont	(FPTR) 0	font	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XFontStruct *“, sofern zur Font-Definition passend. Andernfalls NULL.
AT_XFontSet	(FPTR) 0	font	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XFontSet“, sofern zur Font-Definition passend. Andernfalls NULL.
AT_XmFontList	(FPTR) 0	font	Dieser Wert liefert die X-Windows-spezifische Struktur für den angegebenen Zeichensatz zurück. Der Rückgabewert der Funktion ist vom Typ „XmFontList“, sofern zur Font-Definition passend. Andernfalls NULL.
		0 oder sichtbare IDM Objekte	Liefert die Default-Schriftart die typischerweise vom IDM bei einem nicht gesetzten font-Attribut verwendet wird.
AT_XScreen	(FPTR) 0	0	Dieser Wert liefert den Bildschirm zu dem angegebenen Dialog. Der Rückgabewert der Funktion ist dann vom Typ „Screen“.

Attribut	data	Objekt	Bedeutung
AT_XShell	(FPTR) 0	0	Dieser Wert liefert das Shell-Widget eines Objekts. Der Rückgabewert der Funktion ist ein Widget.
AT_XtAppCon-text	(FPTR) 0	0	Liefert den Application Context. Returns the Application Context.
AT_XTile	(FPTR) 0	tile	Dieser Wert liefert die X-Windows-spezifische Struktur eines Musters „tile“ zurück. Der Rückgabewert der Funktion ist dabei davon abhängig, wie das Muster definiert worden ist. Wurde es in einer externen Datei im „gif“-Format hinterlegt, wird ein „XImage*“ zurückgeliefert; wurde es direkt in der Dialog Manager Datei definiert, wird ein „Pixmap“ zurückgegeben.
	DM_Tool-kitDataArgs *data	tile	Dieser Wert liefert die X-Windows-spezifische Struktur eines Musters „tile“ zurück (mit gesetztem argmask=0 oder armask=*DM_TKAM_scaledpi*). Wenn Bildinformationen vorliegen, werden folgende Informationen in der DM_ToolkitDataArgs-Struktur zurück geliefert: » DM_TKAM_tile: In der tile-Substruktur den Bildtype (DM_GFX_PIXMAP oder DM_GFX_XIMAGE)in gfxtype. Fuer eine PIXMAP die Bildinformation im pixmap-Member und die Transparenz-Clipmaske im trans_mask-Member, Bei eine XIMAGE wird die vollstaendige Bildinformation im ximage-Member geliefert. » DM_TKAM_rectangle: Breite und Hoehe des Bildes in der rectangle Substruktur » DM_TKAM_dpi: Die DPI-Informationen analog zum AT_DPI-Aufruf. » DM_TKAM_scaledpi: Die DPI-Informationen analog zum AT_DPI-Aufruf.

Attribut	data	Objekt	Bedeutung
AT_XVisual	(FPTR) 0	0	Dieser Wert liefert eine Visual Struktur für den Default-Screen.
AT_XWidget	(FPTR) 0	IDM Objekte	Dieser Wert liefert das Widget zu dem angegebenen Objekt. Der Rückgabewert der Funktion ist dann vom Typ „Widget“.
AT_XWindow	(FPTR) 0	IDM Objekte	Dieser Wert ist das zu gehörende Fenster. Der Rückgabewert der Funktion ist dann vom Typ „Window“.

Zu Beachten bei Multiscreendialogen

Der Aufruf mit AT_ XTile bzw. AT_XColor liefert immer nur das tile bzw. die color des default Screen zurück (siehe auch Kapitel „Multiscreen Ssupport untder Motif“ im Handbuch „Programmiertechniken“).

3.33.2 Microsoft Windows

Mit Hilfe dieser Funktion können die für MICROSOFT WINDOWS notwendigen Daten erfragt werden, wie „window handle“, „instance handle“ und „color“.

Die Bedeutung dieser Datentypen wird im entsprechenden Microsoft Windows Handbuch erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_CanvasData	(FPTR) 0	canvas	FPTR	Über dieses Attribut können die benutzerspezifischen Daten eines Canvas-Objekts erfragt werden. Diese Daten wurden von DM_ SetToolkitData oder einer Canvas-Callback-Funktion gesetzt und beinhalten jegliche benutzerspezifischen Daten (Siehe auch Kapitel „Strukturen für Canvas-Funktionen“).

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_CellRect	DM_Tool-kitDataArgs *data	tablefield	data	Dieses Attribut bestimmt die Koordinaten einer Tablefield Zelle in IDM Pixel. Hierzu muss das „data->index“ Feld auf den gewünschten Index gesetzt werden (Bit „DM_TKAM_index“ in „data->argmask“ nicht vergessen). In „data->argmask“ wird das Bit „DM_TKAM_rectangle“ gesetzt und die entsprechenden Felder ausgefüllt (siehe Beschreibung DM_ToolkitDataArgs). Mit Zelle ist der rechteckige Bereich im Tablefield gemeint, in dem Schatten, Fokus- und Aktivierungsrahmen und Text gezeichnet werden. Nicht zur Zelle gehören die zwischen den Zellen gezeichneten Linien. Die gelieferten Koordinaten sind relativ zur linken oberen Ecke des Tablefield zu sehen. Wenn Position und Größe einer Zelle ermittelt werden konnten, liefert DM_GetToolkitDataEx den Zeiger auf die angegebene DM_ToolkitDataArgs-Struktur zurück, in allen anderen Fällen (FPTR) 0. Position und Größe können nur ermittelt werden, wenn sowohl Tablefield

Attribut	data	Objekt	Rück-gabewert	Bedeutung
				als auch Zeile und Spalte der Zelle sichtbar geschaltet sind und die Zelle ganz oder teilweise im Tablefield sichtbar ist. Eine konkrete, absolute Unsichtbarkeit (z.B. weil das Fenster außerhalb des sichtbaren Bildschirms liegt oder anderweitig verdeckt ist) kann trotz gelieferter Position und Größe nicht ausgeschlossen werden und ist Fenstersystem-abhängig.
AT_ClipboardText	(FPTR) 0	setup oder 0	DM_String	Dieses Attribut gibt den Textinhalt des Microsoft Windows Clipboards zurück. Der Rückgabewert wird gepuffert und ist bis zur erneuten Abfrage des Attributes gültig. Auch ein Setzen des Attributes macht den Puffer ungültig. Um den String ohne Änderung des Clipboards freizugeben, dient der Aufruf: `DM_SetToolkitData (0, AT_ClipboardText, (FPTR) 0, 0);`
AT_Color	(FPTR) 0	color	COLORREF	Siehe auch AT_XColor
		tile	HPALETTE	Siehe auch AT_XColor

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_DataType	(FPTR) 0	tile	int	Die Abfrage ist nur noch aus Kom-patibilitätsgründen vor-handen. Es sollte mit DM_ GetToolkitDataEx das Attribute AT_Tile mit gesetztem "data" Para-meter verwendet werden. Dieses Attribut gibt den Typ des Musters zurück. Die Zuordnung ist bei AT_ XTile beschrieben.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_DPI	(FPTR) 0	IDM Objekte	int	Siehe auch AT_GetDPI
	DM_Tool-kitDataArgs *data	setup oder 0	int	Dieses Attribut gibt den System DPI Wert wie bei „AT_GetDPI“ zurück. Wenn „DM_TKAM_hand-le“ gesetzt ist, wird statt-dessen der DPI Wert des Microsoft Windows Con-trol, dessen Window Handle (HWND) in „data->handle“ angegeben ist, ermittelt. Außerdem wer-den in „data->argmask“ die Bits „DM_TKAM_dpi“ und „DM_TKAM_sca-leddpi“ gesetzt und die entsprechenden Felder ausgefüllt (siehe Beschrei-bung DM_Tool-kitDataArgs).
		Übrige IDM Objekte	int	Dieses Attribut gibt den DPI Wert des Objektes wie bei „AT_GetDPI“ zurück. Außerdem wer-den in „data->argmask“ die Bits „DM_TKAM_dpi“ und „DM_TKAM_sca-leddpi“ gesetzt und die entsprechenden Felder ausgefüllt (siehe Beschrei-bung DM_Tool-kitDataArgs).
AT_Font	(FPTR) 0	font	HFONT	Siehe auch AT_XFont
		IDM Objekte	HFONT	Siehe auch AT_XFont
		setup	HFONT	Siehe auch AT_XFont

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_GetDPI	(FPTR) 0	setup oder 0	int	Dieses Attribut gibt den System DPI Wert zurück. Siehe Hinweis unten.
		IDM Objekte	int	Dieses Attribut gibt den DPI Wert des Objektes zurück. Dieser ist abhängig zu welchem Monitor das Objekt zugeordnet ist. Siehe Hinweis unten.
	HWND data	setup oder 0	int	Dieses Attribut gibt den DPI Wert des Microsoft Windows Objektes, dessen Handle (HWND) in „data“ übergeben wurde, zurück. Dieser ist abhängig zu welchem Monitor das Objekt zugeordnet ist. Siehe Hinweis unten.
AT_IsNull		font oder color	int	Hiermit kann abgefragt werden, ob die Resource auf *NULL* definiert wurde (*UI_NULL_FONT* bzw. *UI_NULL_COLOR*). Die Resource wurde auf *NULL* definiert, wenn der Rückgabewert nicht 0 ist. Hereby it can be queried whether the resource was defined to *NULL* (*UI_NULL_FONT* or *UI_NULL_COLOR*). The resource has been defined to *NULL* if the return value is not *0*.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_maxsize	(FPTR) 0	setup oder 0	int	Dieses Attribut gibt die Anzahl der noch freier WSI-ID's zurück. **Achtung:** Die Anzahl noch verfügbarer WSI-IDs sagt nichts darüber aus, wie viele Objekte tatsächlich noch sichtbar gemacht werden können! Es ist die maximale Obergrenze.
AT_ObjectID	DM_ToolkitDataArgs *data	setup oder 0	data	Dieses Attribut gibt die DM_ID eines Microsoft Windows Objektes zurück. Hierzu muss das „data->handle" Feld auf den Microsoft Windows Fenster Handle (HWND) gesetzt werden (Bit „DM_TKAM_handle" in „data->argmask" nicht vergessen). Wenn eine Dialog Manager ID ermittelt werden kann, wird der Rückgabewert auf „data" gesetzt, das data->argmask Bit „DM_TKAM_data" gesetzt und das „data->data" Feld mit der DM_ID gefüllt.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_Raster	(FPTR) 0	dialog editbox groupbox layoutbox module notebook notepage spinbox splitbox statusbar tablefield toolbar Window	DWORD	Dieses Attribut gibt die Größe des am Objekt definierten Rasters in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
		font	DWORD	Dieses Attribut gibt die Größe der Schriftart in IDM Pixel, wie sie für die Rasterberechnung verwendet wird, zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Sroll-barDimension	(FPTR) 0	groupbox notepage window	DWORD	Dieses Attribut gibt die Breite der senkrechten und die Höhe der waagrechten Scrollbar in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_Size	(FPTR) 0	dialog module	DWORD	Dieses Attribut gibt die Größe des Arbeits-bereichs des primären Monitors in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
		font	DWORD	Dieses Attribut gibt die Größe der Schriftart in IDM Pixel zurück. Die Breite wird anhand des Referenzstrings berech-net, falls einer angegeben ist. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
		Übrige IDM Objekteaußer menubox, menuitem und menu-separator	DWORD	Dieses Attribut gibt die Größe des Objektes in IDM Pixel zurück. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Tile	(FPTR) 0	color	HBRUSH	Siehe auch AT_XTile
		tile	HANDLE	Die Abfrage ist nur noch aus Kom-patibilitätsgründen vor-handen. Es sollte DM_ GetToolkitDataEx mit gesetztem "data" Para-meter verwendet werden. Siehe auch AT_XTile

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_toolhelp	(FPTR) 0	setup oder 0	HWND	Dieses Attribut gibt den Microsoft Windows Handle des tooltip Controls zurück, welches vom Dialog Manager zur Darstellung des *.toolhelp* Attributs verwendet wird. Siehe auch „DM_GetToolkitDataEx“ weiter unten.
AT_value	(FPTR) 0	RTF edittext	DM_String	Dieses Attribut gibt den vollständigen Inhalt, also mit allen Formatierungsanweisungen usw., eines RTF-Eingabefeldes zurück.
AT_VSize	(FPTR) 0	IDM Objekte außer menubox, menuitem und menuseparator	DWORD	Dieses Attribut gibt die virtuelle Größe des Objektes in IDM Pixel zurück. Gibt es keine virtuelle Größe, dann wird die reale Größe in IDM Pixel zurückgegeben. Die Breite und Höhe ist in ein „DWORD“ gepackt, siehe Hinweis unten.
AT_Widget	(FPTR) 0	USW	HWND	Siehe auch AT_XWidget

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_WinHandle	(FPTR) 0	dialog module setup oder 0	HINSTANCE	Dieses Attribut gibt den Microsoft Windows Handle der Anwen-dungsinstanz zurück.
		menubox	HMENU	Dieses Attribut gibt den Microsoft Windows Menu Handle zurück.
		menuitem menu-separator	HMENU	Dieses Attribut gibt den Microsoft Windows Menu Handle des umgebenden ***menubox*** Objekts zurück.
		Übrige IDM Objekte	HWND	Dieses Attribut gibt den Microsoft Windows Handle des Objekts zurück. Die Grup-pierungsobjekte (***group-box***, ***notepage***, ***window***, …) sind meist aus meh-reren Microsoft Windows Objekten aufgebaut, bei diesen wird das Handle des „Client“-Fensters (das Fenster in dem die Kin-dobjekte angelegt wer-den) zurückgegeben.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_wsidata	(FPTR) 0	cursor	HCURSOR	Siehe auch AT_XCursor
		font	HFONT	Siehe auch AT_XFont
		tile	HANDLE	Die Abfrage ist nur noch aus Kom-patibilitätsgründen vor-handen. Es sollte DM_ GetToolkitDataEx mit gesetztem "data" Para-meter verwendet werden. Siehe auch AT_XTile
		Übrige IDM Objekte	HWND	Dieses Attribut gibt den Microsoft Windows Handle des äußeren Microsoft Windows Objek-tes zurück, das innere kann mit „AT_WinHandle“ erfragt werden.

Attribut	data	Objekt	Rück-gabewert	Bedeutung

AT_XColor	(FPTR) 0	color	COLORREF	Dieses Attribut gibt die Microsoft Windows-spezifische Struktur für die angegebene Farbe zurück. Der Zugriff auf die Farbwerte erfolgt mit den entsprechenden Makros von Microsoft Windows:
				`COLORREF u1RGB = (COLORREF) (size_t) DM_GetToolkitDataEx (colorID, AT_ XColor, (FPTR) 0, 0);`
				`BYTE ucRed = GetRValue(u1RGB);`
				`BYTE ucGreen = GetGValue(u1RGB);`
				`BYTE ucBlue = GetBValue(u1RGB);`

		tile	HPALETTE	Dieses Attribut gibt die Microsoft Windows Farbpalette, die von dem Muster verwendet wird, zurück.
AT_XCursor	(FPTR) 0	cursor	HCURSOR	Dieses Attribut gibt den Microsoft Windows Cursor Handle zurück.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_XFont	(FPTR) 0	font	HFONT	Dieses Attribut gibt den Microsoft Windows Font Handle zurück.
		IDM Objekte	HFONT	Dieses Attribut gibt den Microsoft Windows Font Handle der an diesem Objekt verwendeten Schriftart zurück.
		setup	HFONT	Es wird der Font-Handle der verwendeten Standardschriftart zurück geliefert.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_XTile	(FPTR) 0	color	HBRUSH	Dieses Attribut gibt eine Microsoft Windows Brush der Farbe zurück. Diese Brush kann zum Füllen des Hintergrunds verwendet werden.

Attribut	data	Objekt	Rück-gabewert	Bedeutung
	(FPTR) 0	tile	HANDLE	Die Abfrage von AT_wsi-data, AT_Tile und AT_ XTile ohne Angabe des "data" Parameters ist nur noch aus Kom-patibilitätsgründen vor-handen. Es sollte der "data" Parameter ver-wendet werden. Dieses Attribut gibt die Microsoft Windows spezifische Struktur für das Muster (tile) wie in Version A.06.03.a und früher zurück. Hierzu müssen GDI Objekte zusätzlich angelegt werden. Um die neuen Microsoft Direct2D Daten, die der IDM intern verwendet, zu erhalten, muss der "data" Para-meter gesetzt werden. Der Typ des Microsoft Windows Handle ist abhängig von AT_ DataType, wobei AT_ DataType erst nach der Abfrage von AT_wsidata, AT_Tile oder AT_XTile abgefragt werden darf: » *DMF_TlkDataIsIcon*: HICON » *DMF_TlkDataIsWMF*: HMETAFILE » *DMF_TlkDataIsEMF*: HENHMETAFILE » Sonst: HBITMAP

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			» *DMF_TlkDataIsIcon*: HICON » *DMF_TlkDataIsWMF*: HMETAFILE » *DMF_TlkDataIsEMF*: HENHMETAFILE » Sonst: HBITMAP **WICHTIG**: Die zurückgelieferten Daten sollten nicht gespeichert werden, da diese automatisch freigegeben werden, wenn die tile Resource von keinem sichtbaren IDM Objekt mehr verwendet wird. **Hinweis**: Wenn mittels DM_SetToolkitData Daten gesetzt wur-	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			den, dann werden die gesetzten und nur die gesetzten Daten zurückgeliefert.	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
DM_Tool-kitDataArgs *data	tile	HANDLE / LPUNKNOWN	Dieses Attribut gibt die Micro-soft Windows spezifische Struktur für das Muster (tile) wie in Version A.06.03.a und früher zurück. Hierzu müssen GDI Objekte zusätzlich angelegt wer-den. Um die neuen Micro-soft Direct 2D Daten, die der IDM intern ver-wendet, zu erhalten, muss in "data->arg-mask" das "DM_TKAM_ tile_req" Bit gesetzt wer-den. Die gewünschten Datentypen werden in "data->tile_ req" ange-geben, fol-gende Werte sind möglich: - DM_GFX_ BMP: GDI Bit-map Handle (HBITMAP)	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			- DM_GFX_WMF: GDI Metafile Handle (HMETAFILE) - DM_GFX_EMF: GDI Enhanced Metafile Handle (HENHMETAF-ILE) - DM_GFX_ICO: GDI Icon Handle (HICON) - DM_GFX_D2D1BMP: Direct2D Bit-map (ID2D1Bit-map *) - DM_GFX_D2D1SVG: Direct2D SVG Documnet (ID2D1Sv-gDocument *) - DM_GFX_D2D1EMF: Direct2D Meta-file (ID2D1G-diMetafile *) Diese Werte können mit "bit-wise or" ver-knüpft angegeben werden. Es	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			wird dann einer davon aus-gewählt, der Datentyp DM_ GFX_BMP wird immer als Fallback ver-wendet (auch wenn dieser nicht explizit angegeben wurde). Falls notwendig und möglich wird auf einen der gewünschten Datentypen gewandelt, hier-durch werden gege-benenfalls zusätzliche Ressourcen verbraucht. Das Muster wird für einen angeforderten DPI Wert bestimmt. Die-ser wird in der angegebenen Reihenfolge aus den fol-genden Anga-ben ermittelt: » Ist in „data->argmask“ das	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			„DM_TKAM_handle“ Bit gesetzt, dann wird der DPI Wert des Microsoft Windows Control ermittelt dessen Windows Handle („HWND“) im „data->handle“ Feld steht. » Andern-falls, wenn in „data->argmask“ das „DM_TKAM_sca-ledpi“ Bit gesetzt ist, dann wird der System DPI Wert ermittelt (entspricht dem Ver-grö-ßerungs-faktor des primären Monitors). » Wenn kei-	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			ner der vor-herigen Bedin-gungen erfüllt war, dann wird die ori-ginale Bild-größe, also der DPI Wert für den die Bil-der ent-worfen wurden ver-wendet. Außerdem wer-den in „data->argmask“ die Bits „DM_TKAM_tile“, „DM_TKAM_rectangle“, „DM_TKAM_dpi“ und „DM_TKAM_sca-leddpi“ gesetzt und die ent-sprechenden Felder aus-gefüllt (siehe Beschreibung DM_Tool-kitDataArgs), wobei die DPI Werte auf den angeforderten DPI Wert	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			gesetzt werden und das Rec-tangle wird auf die Größe gesetzt, die zu diesem ange-forderten DPI Wert passt. Achtung: Dies bedeutet nicht, dass die Bild-daten die ent-sprechende Größe haben, diese müssen eventuell ska-liert werden. Der Rück-gabewert ent-spricht entweder „data->tile.data“ oder „data->tile.iunk“, siehe DM_Tool-kitDataArgs. **Hinweis**: Den Datentyp des Rück-gabewertes steht entweder im Eintrag „data->ti-le.gfxtype“ oder „data->ti-le.datatype“. Die Abfrage von AT_	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
			DataType ist obsolet und darf bei Verwendung des "data" Parameters nicht mehr verwendet werden. **WICHTIG**: Die zurückgelieferten Daten sollten nicht gespeichert werden, da diese automatisch freigegeben werden, wenn die tile-Ressource von keinem sichtbaren IDM-Objekt mehr verwendet wird. **Hinweis**: Wenn mittels DM_SetToolkitData Daten gesetzt wurden, dann werden die gesetzten und nur die gesetzten Daten zurückgeliefert.	

Attribut	data	Objekt	Rück-gabewert	Bedeutung
AT_XWidget	(FPTR) 0	USW	HWND	Dieses Attribut gibt den Microsoft Windows Handle des USW Objektes zurück.

Hinweis zum Objekt und Attribut

Das beim Aufruf angegebene Objekt muss im Allgemeinen sichtbar und damit im WSI angelegt sein, damit die zurückgegebenen Daten Sinn machen. Ressourcen werden in der Regel beim Aufruf angelegt. Wird ein Objekttyp angegeben, der beim betreffenden Attribut nicht erwähnt ist, wird in der Regel eine Fehlermeldung in die Logdatei geschrieben und „(FPTR) 0“ zurückgegeben.

Hinweis zum Zugriff auf den Rückgabewert

Der Rückgabewert der Funktion ist ein „FPTR“ bzw. „void *“, dieser muss auf den dokumentierten Rückgabewert gecastet werden, um beim Kompilieren keine Warnungen zu erhalten.

Da ein „void *“ Zeiger auf jeden beliebigen anderen Zeiger gecastet werden kann, reicht bei allen Zeiger-Datentypen ein einfacher Castoperator. Zu den Zeiger-Datentypen zählen zum Beispiel sämtliche Microsoft Windows Handle, wie zum Beispiel „HWND“, „HFONT“, ... :

```
HWND hwnd = (HWND) DM_GetToolkitDataEx(idObj, AT_wsidata, (FPTR) 0, 0);
```

Bei Zahlenwerten muss meist ein Zwischencast eingefügt werden, da die Größe des Datenwerts beim Cast von einem Zeiger zu einer Zahl erhalten bleiben muss, um Warnungen zu vermeiden. Hierzu kann der Datentyp „size_t“ verwendet werden, da dieser per Definition dieselbe Größe wie ein Zeiger besitzt. Anschließend kann dann auf einen kleineren Zahlentyp gecastet werden:

```
DM_UInt2 val = (DM_UInt2) (size_t) DM_GetToolkitDataEx(idObj, AT_wsidata,
(FPTR) 0, 0);
```

Hinweis Breite und Höhe in „DWORD“ gepackt

Unter Microsoft Windows werden häufig Breite und Höhe in eine „DWORD“ gepackt. Dies wird vom IDM zum Teil auch so gehandhabt. Die einzelnen Werte können dann mit den Microsoft Windows Makros „LOWORD“ und „HIWORD“ extrahiert werden:

```
DWORD size = (DWORD) (size_t) DM_GetToolkitDataEx(id, AT_Size, (FPTR) 0, 0);
WORD width = LOWORD(size);
WORD height = HIWORD(size);
```

Hinweis IDM Pixel

Der IDM für WINDOWS 11 unterstützt hohe Auflösungen. Um Auswirkungen auf bestehende Dialogskripte so gering wie möglich zu halten, werden vom ISA DIALOG MANAGER virtuelle Pixelkoordinaten verwendet. Diese basieren auf der Größe einer Anwendung, die hohe Auflösungen nicht unterstützt, wie zum Beispiel IDM für WINDOWS 10.

Hinweis zu DPI-Werten

Achtung, alle DPI Werte sind dynamisch und können durch den Anwender geändert werden. Zum Beispiel können IDM Objekten auf einen anderen Monitor verschoben werden oder der Anwender stellt über die Systemsteuerung andere Vergrößerungsfaktoren ein.

Ist die Anwendung nicht DPI Aware (zum Beispiel IDM für WINDOWS 10) dann wird immer der Standard DPI Wert von 96 verwendet.

Hinweiszu NULL-Werten bei Resourcen

Die Funktion **DM_GetToolkitDataEx** liefert unter MICROSOFT WINDOWS bei den font- und color Resourcen einen **NULL** Wert zurück, wenn die Resource auf **UI_NULL_FONT** bzw. **UI_NULL_COLOR** definiert wurde. Dies betrifft folgende Attribute:

- *AT_wsidata*, *AT_Font*, *AT_XFont*:
  Der Rückgabewert ist bei *UI_NULL_FONT* dann *(HFONT) 0*.
- *AT_Color*, *AT_XColor*:
  Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(COLORREF) -1L*.
- *AT_Tile*, *AT_XTile*:
  Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(HBRUSH) 0*.

Hinweis IDM Pixel

3.33.3 Qt

Folgende Werte sind für die Attribute zugelassen:

Attribut	data	Objekt	Rückgabewert	Bedeutung
AT_Application	(FPTR) 0	0	FPTR auf QApplication	Über dieses Attribut kann die der Anwendung zu Grunde liegende QApplication erfragt werden.

Attribut	data	Objekt	Rückgabewert	Bedeutung
AT_CanvasData	(FPTR) 0	cancas	FPTR	Über dieses Attribut können die benutzerspezifischen Daten eines Canvas-Objekts erfragt werden. Diese Daten wurden von DM_SetToolkitData oder einer Canvas-Callback-Funktion gesetzt und beinhalten jegliche benutzerspezifischen Daten (Siehe auch Kapitel „Strukturen für Canvas-Funktionen“).
AT_Color	(FPTR) 0	color	QColor / QBrush	Über dieses Attribut kann die von der Color-Ressource verwendete Farbe oder Farbverlauf (als QBrush) erfragt werden. Achtung: Es sollte als erstes immer auf eine zulässige Farbe (QColor) geprüft werden, da sich ein QBrush automatisch in eine QColor casten lässt, welche dann jedoch eine uninitialisierte aber zulässige Farbe darstellt.

Attribut	data	Objekt	Rückgabewert	Bedeutung
AT_DPI	(DM_Tool-kitDataArgs *data	0	int	Dieses Attribut gibt den System DPI Wert zurück. Außerdem werden in „data->argmask“ die Bits „DM_TKAM_dpi“ und „DM_TKAM_sca-leddpi“ gesetzt und die entsprechenden Felder ausgefüllt (siehe Beschreibung DM_ ToolkitDataArgs).
AT_Font	(FPTR) 0	font	QFont	Über dieses Attribut kann die von der Font-Ressource verwendete QFont erfragt werden.
AT_FontName	(FPTR) 0	font	char*	Über dieses Attribut kann der Name der von der Font-Ressource verwendeten QFont erfragt werden.

Attribut	data	Objekt	Rückgabewert	Bedeutung
AT_ObjectID	DM_Tool-kitDataArgs *data	0	data	Dieses Attribut gibt die DM_ID eines Qt Widgets zurück. Hierzu muss beim Zeiger auf DM_ToolkitDataArgs-Struktur das „data->widget“ Feld auf das QWidget und argmask = DM_TKAM_widget gesetzt sein. Wenn eine Dialog Manager ID ermittelt werden kann, wird der Rückgabewert auf „data“ gesetzt, das data->argmask Bit „DM_TKAM_data“ gesetzt und das „data->data“ Feld mit der DM_ID gefüllt (siehe Beschreibung DM_ToolkitDataArgs).
AT_Tile	(FPTR) 0	tile	QPixmap	Über dieses Attribut kann das QPixmap von einem Muster (tile) erfragt werden.

Attribut	data	Objekt	Rückgabewert	Bedeutung
AT_XTile	(FPTR) 0	tile	QPixmap	Siehe AT_Tile
	DM_Tool-kitDataArgs *data	tile	QPixmap	Dieses Attribut gibt die Qt spezifische Struktur für das Muster (tile) zurück. Außerdem werden in „data->argmask“ die Bits „DM_TKAM_tile“, „DM_TKAM_ rectangle“, „DM_ TKAM_dpi“ und „DM_ TKAM_scaleddpi“ gesetzt und die ent-sprechenden Felder ausgefüllt (siehe Beschreibung DM_ ToolkitDataArgs). Der Rückgabewert ent-spricht „data->ti-le.pixmap“.
AT_XWidget	(FPTR) 0	IDM Objekte	QWidget	Dieses Attribut ermittelt das zu einer DM_ID gehörende QWidget.

Siehe auch

Funktion DM_GetToolkitDataEx

3.34 DM_GetValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten zu erfragen

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_GetValue
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_UInt index,
  DM_Value *data,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie erfragen möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Attribut, das Sie von dem Objekt erfragen möchten. Alle zugelassenen Attribute sind in der Datei **IDMuser.h** definiert.

-> DM_UInt index

Dieser Parameter wird nur bei Vektorattributen von Objekten ausgewertet und beschreibt den Index des gesuchten Unterobjekts (z.B. Text in Listbox).

-> DM_Value*data

In diesem Parameter erhalten Sie den Wert des gesuchten Attributes. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element aus dieser Union auslesen. Den Datentyp eines jeden Attributes entnehmen Sie bitte der „Attributreferenz“.

-> DM_Options options

Mit Hilfe dieses Parameters wird u.a. gesteuert, in welcher Form Texte vom Dialog Manager aus zurückgeliefert werden, falls das entsprechende Attribut vom Typ Text ist. Folgende Belegungen sind für diesen Parameter möglich:

Option	Bedeutung
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Attributen der String in der Entwicklungssprache zurückgegeben werden soll, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Attributen der String in der aktuell eingestellten Sprache zurückgegeben werden soll.
DMF_GetTextID	Diese Option bedeutet, dass bei textuellen Attributen der String als TextID zurückgegeben werden soll. Diese ist dann sinnvoll, wenn der Text einem anderen Objekt zugewiesen werden soll.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den Dialog Manager erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine DM-Funktion ohne diese Option erfolgt und dabei ein String vom DM an die Anwendung zurückgegeben wird.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich erfragt werden.
FALSE	Das Attribut ist für das Objekt nicht zulässig.

Beispiel

Erfragen des Inhalts eines editierbaren Textes in einer Objekt-Callback-Funktion.

```
DM_Boolean DML_default DM_CALLBACK CheckFilename __1(
  (DM_CallBackArgs *, data))
{
  DM_Value value;    /*structure for DM_GetValue*/

  /* get current contents */
  if (DM_GetValue(data->object, AT_content, 0, &value,
                  DMF_GetLocalString))
    /* check the datatype */
    if(value.type == DT_string)
```

Siehe auch

Eingebaute Funktion getvalue im Handbuch „Regelsprache“

3.35 DM_GetValueIndex

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute mit zwei Indizes zu erfragen.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_GetValueIndex
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_Value *index,
  DM_Value *data,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie erfragen möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Attribut, das Sie von dem Objekt erfragen möchten. Alle zugelassenen Attribute sind in der Datei IDMuser.h definiert.

-> DM_Value *index

Hier kann der Datentyp des Index (enum, index) und dessen Wert angegeben werden.

-> DM_Value *data

In diesem Parameter erhalten Sie den Wert des gesuchten Attributes. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element aus dieser Union auslesen. Den Datentyp eines jeden Attributes entnehmen Sie bitte der „Attributreferenz“.

-> DM_Options options

Mit Hilfe dieses Parameters wird u.a. gesteuert, in welcher Form Texte vom Dialog Manager aus zurückgeliefert werden, falls das entsprechende Attribut vom Typ Text ist. Folgende Belegungen sind bei diesem Parameter möglich:

Option	Bedeutung
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Attributen der String in der Entwicklungssprache zurückgegeben werden soll, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Attributen der String in der aktuell eingestellten Sprache zurückgegeben werden soll.
DMF_GetTextID	Diese Option bedeutet, dass bei textuellen Attributen der String als TextID zurückgegeben werden soll. Diese ist dann sinnvoll, wenn der Text einem anderen Objekt zugewiesen werden soll.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den Dialog Manager erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine DM-Funktion ohne diese Option erfolgt und dabei ein String vom DM an die Anwendung zurückgegeben wird.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich erfragt werden.
FALSE	Das Attribut ist für das Objekt nicht zulässig.

Beispiel

Abfragen eines vektoriellen benutzerdefinierten Attributes einer Groupbox oder eines Fensters.

```
void DML_default DM_ENTRY GetInfo  __1((DM_ID, obj))
{
  DM_Value  attr;
  DM_Value data, index;
  DM_ID groupbox;

  DM_GetValue(obj, AT_parent, 0, &data, 0);
  groupbox = data.value.id;
  DM_TraceMessage("\nin GetInfo\n",DMF_Printf);

  /*
  ** Holen der Anzahl des benutzerdefinierten Attributes
  */
```

```
  if (DM_GetValue(obj, AT_membercount, 0, &data, 0))
    DM_TraceMessage("groupbox.membercount = %ld\n",
        DMF_Printf, data.value.integer);

  /*
  ** Setzen des Namens des benutzerdefinierten Attributes
  */
   index.type = DT_string;
   index.value.string = ".StringVec";
   if (DM_GetValueIndex(groupbox, AT_label, &index, &attr, 0))
   {
    DM_Integer n, i;
    DM_GetValueIndex(groupbox, AT_count, &attr, &data, 0);
    n = data.value.integer;
    for (i=1 ; i<=n; i++)
    {
      DM_GetValue(groupbox, attr.value.attribute, i, &data,
        0);
      DM_TraceMessage ("erst attribut%ld.string %s\n",
        DMF_Printf, i, data.value.string);
    }
  }
}
```

Siehe auch

Eingebaute Funktion getvalue im Handbuch „Regelsprache“

Methode get

3.36 DM_GetVectorValue

Mit Hilfe dieser Funktion können Attribute erfragt werden, die bei einem Objekt mehrfach vorkommen, sog. "vektorielle Attribute". Zusätzlich können über diese Funktion benutzerdefinierte Attribute erfragt werden.

Im Gegensatz zum normalen DM_GetValue werden bei diesem Aufruf Strukturen vom Dialog Manager aufbereitet und Speicherplatz allokiert. Dieser muss dann anschließend unbedingt mit Hilfe der Funktion DM_FreeVectorValue wieder freigegeben werden.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DML_default DM_EXPORT DM_GetVectorValue
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_Value *firstindex,
  DM_Value *lastindex,
  DM_VectorValue **values,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie erfragen möchten.

-> DM_Attribute attr

Dieser Parameter bezeichnet das Attribut, das erfragt werden soll.

-> DM_Value *firstindex

Mit Hilfe dieses Parameters wird gesteuert, welcher Bereich des Inhalts durch diese Funktion erfragt werden soll. Dabei wird in diesem Parameter der Startpunkt des Bereichs definiert. Das bedeutet bei einem eindimensionalen Attribut, dass der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Startwert belegt wird. Bei einem zweidimensionalen Attribut bedeutet das, dass der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Startwert belegt wird. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Startpunkt folgende Defaultwerte:

listbox	integer = 1

poptext	integer = 1
tablefield	index.first = 1, index.second = 1
treeview	Integer = 1

-> DM_Value *lastindex

Mit Hilfe dieses Parameters wird gesteuert, welcher Bereich des Inhalts durch diese Funktion erfragt werden soll. Dabei wird in diesem Parameter der Endpunkt des Bereichs definiert. Das bedeutet für ein eindimensionales Attribut, dass der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Endwert belegt wird. Für ein zweidimensionales Attribut bedeutet das, dass der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Endwert belegt wird. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Endpunkt folgende Defaultwerte:

listbox	integer = Object.itemcount
poptext	integer = Object.itemcount
tablefield	index.first = object.rowcount, index.second = object.colcount
treeview	Integer = Object.itemcount

<- DM_VectorValue **values

Dieser Parameter ist ein Zeiger auf die Werte, die erfragt werden sollen. Über das Feld *type* in der DM_VectorValue-Struktur wird gesteuert, welchen Datentyp die einzelnen Werte haben. Über das Feld *count* in der DM_VectorValue-Struktur wird gesteuert, wie viele Werte in dem Vektor enthalten sind. Die Felder *type* und *count* werden durch den Funktionsaufruf ausgeführt.

-> DM_Options options

Mit Hilfe dieses Parameters wird gesteuert, in welcher Form Texte vom Dialog Manager aus zurückgeliefert werden, falls das entsprechende Attribut vom Typ Text ist:

Option	Bedeutung
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Attributen der String in der Entwicklungssprache zurückgegeben werden soll, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Attributen der String in der aktuell eingestellten Sprache zurückgegeben werden soll.

Option	Bedeutung
DMF_GetTextID	Diese Option bedeutet, dass bei textuellen Attributen der String als TextID zurückgegeben werden soll. Diese ist dann sinnvoll, wenn der Text einem anderen Objekt zugewiesen werden soll.
DMF_ DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den Dialog Manager erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine DM-Funktion ohne diese Option erfolgt und dabei ein String vom DM an die Anwendung zurückgegeben wird.
DMF_UseStringBuffer	Der temporäre Puffer für Strings wird genutzt. Die Freigabe von Strings erfolgt frühestens bei Aufruf einer DM-Funktion ohne gesetzte *DMF_DontFreeLastStrings*-Option.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich erfragt werden.
FALSE	Das Attribut ist für das Objekt nicht zulässig.

Hinweis zur Abfrage des Attributs *.format* mit DM_GetVectorValue()

Es ist zu beachten, dass **DM_GetVectorValue()** bei der Abfrage von *.format* die Formatdefinition nur bei Verwendung von Formatressourcen zurückliefern kann. Bei Formatdefinitionen per Formatstring oder Formatfunktion wird Null zurückgeliefert. Daher wird die Nutzung von Formatressourcen empfohlen.

Mit den Optionen *DMF_GetMasterString* und *DMF_GetLocalString* erhält man einen Formatstring wie bei einer Formatdefinition per Formatstring.

Damit entsprechen die zurück gelieferten Werte der Funktion **DM_GetVectorValue()** denen der Funktion **DM_GetValueIndex()** bzw. eines **getvalue()** in der Regelsprache.

Beispiel

Abfrage des Inhalts eines Tablefields mit 5 Spalten zeilenweise:

```
/*
*write the content of a tablefield to a file
*the file format is described above
*/
DM_Boolean DML_default DM_ENTRY SaveTable_ _2(
(DM_ID, Table),
(char *, filename))
```

```
{
    DM_boolean retval = FALSE;
    DM_VectorValue *vector;

    if (DM_GetVectorValue (table, AT_field, (DM_Value *) 0,
        (DM_Value *) 0, &vector, 0))
    {
        FILE *f;

        if ((f = fopen(filename, "w")))
        {
          int vpos = 0;
          int i;
          retval = TRUE;
          while ((vpos + 5) < vector->count)
          {
              DM_boolean ok = TRUE;
              for (i=0; i<5, i++)
                  if (!vector->vector.stringPtr[vpos+i]
                && !*vector->vector.stringPtr[vposi])
                      ok = FALSE;
            if (ok)
                for (1=0, i<5; i++)
                {
                fputs(vector->vector.stringPtr[vpos+i], f);
                    putc((i<4) ? ´ ´ : ´\n´, f);
                vpos += 5;
                }
          }
          fclose(f);

          DM_FreeVectorValue(vector,0);
    }
    return retval;
}
```

Siehe auch

Objekte listbox, poptext, tablefield, treeview

Kapitel „Benutzerdefinierte Attribute“ im Handbuch „Benutzerdefinierte Attribute und Methoden“

3.37 DM_IndexReturn

Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen Index-Werten (**DM_Index**) an den Aufrufer. Bei Verwendung von lokalen Variablen und Strukturen in einer C-Funktion sind diese nach der Rückgabe ungültig. Mit dieser Funktion kann problemlos und sicher ein lokaler Index zurückzugeben werden.

Zu diesem Zweck wird eine temporäre Kopie angelegt.

```
DM_Index * DML_default DM_EXPORT DM_IndexReturn
(
  DM_Index   *pindex,
  DM_Options  options
)
```

Parameter

-> DM_Index * pindex

Dieser Parameter verweist auf den Index welcher zurückgegeben wird.

-> DM_Options options

Hier sind keine Optionen erforderlich, ist mit *0* zu belegen.

Rückgabewert

Zurückgegeben wird ein für die Funktionsrückgabe gültiger Zeiger auf eine **DM_Index**-Struktur oder *NULL* im Fehlerfall. Ein Fehler kann z.B. vorliegen, wenn die Funktion im falschen „runstate“ aufgerufen wird oder das Kopieren nicht ausgeführt werden konnte.

Beispiel

Dialogdatei

```
dialog YourDialog
function index SwapIndex(index Idx);

on dialog start
{
  print SwapIndex([1,3]);
  exit();
}
```

C-Teil

```
...

DM_Index* DML_default DM_ENTRY StringOf(DM_Index* Idx)
{
  DM_Index newIdx;
```

```
  newIdx.first = Idx->second;
  newIdx.second = Idx->first;

  /* wrong: return &newIdx; => newIdx is local! */
  return DM_IndexReturn(&newIdx, 0);
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_StringReturn, DM_ValueReturn

3.38 DM_Initialize

Mit Hilfe dieser Funktion werden interne Datenstrukturen im DM initialisiert. Diese Funktion muss genau einmal von der Anwendung aufgerufen werden.

```
DM_Boolean DML_default DM_EXPORT DM_Initialize
(
  int far *argc,
  char far * far *argv,
  DM_Options options
)
```

Parameter

<-> int far *argc

In diesem Parameter wird ein Zeiger auf die Anzahl der Kommandozeilen-Argumente übergeben. Dieser Parameter wird dabei vom DM verändert. Er entfernt alle Argumente, die er verarbeiten kann, aus dem Argumentvektor und vermindert die Zahl der Argumente entsprechend.

<-> char far * far *argv

Vektor auf die Argumentliste. Aus dieser Liste entfernt der DM alle Argumente, die er direkt verarbeiten kann.

-> DM_Options options

Option	Bedeutung
0	Um keine Option zu definieren.
DMF_FatalNetErrors	Stellt ein kompatibles Verhalten des DISTRIBUTED DIALOG MANAGERS (DDM) zu den IDM-Versionen vor A.05.01.d ein, mit einem sofortigen Abbruch auf Client- und Server-Seite bei Netzwerk-, Protokoll- und Versionsfehlern. Das heißt, außer für lokale Applikationen werden keine *start*- und *finish*-Ereignisse mehr ausgelöst und es erfolgt kein Aufruf von **AppFinish** mehr. Die Option *DMF_FatalNetErrors* ist also innerhalb der Funktion **AppMain** der Client-Anwendung verwendbar sowie auf der Server-Seite durch Rekompilieren von **startup.c** (mit dem zusätzlichen Compiler-Define XXX_FATALNETERRORS) und erneutem Linken der Server-Anwendung. Diese Option ist primär für Anwendungsfälle vorgesehen, in denen die Benutzung der Startoption **-IDMfataneterrors** nicht möglich ist. *DMF_FatalNetErrors* hat Vorrang vor **-IDMfatalneterrors**. **Siehe auch** Startoption **-IDMfatalneterrors** im Kapitel „Startoptionen“ des Handbuchs „Entwicklungsumgebung“

Rückgabewert

TRUE	Der DM hat die Initialisierung erfolgreich durchgeführt.
FALSE	Der DM konnte seine internen Strukturen nicht richtig initialisieren. In diesem Fall sollte die Anwendung beendet werden, da ein sinnvolles Weiterarbeiten nicht möglich ist.

Beim Aufruf dieser Funktion ist zu beachten, dass ihr alle Optionen, die die Anwendung nicht verarbeiten kann, übergeben werden. Der DM entfernt dann aus diesem Argumentvektor alle Argumente, die für ihn bestimmt sind und übergibt den restlichen Vektor an das Fenstersystem, das durch diese Funktion ebenfalls initialisiert wird. Durch diese Verhaltensweise ist der Anwender in der Lage, die Anwendung mit für das Fenstersystem bestimmten Optionen zu starten.

Beispiel

Anfang eines Standard-Hauptprogramms:

```
int DML_c AppMain (argc, argv)
int argc;
char far * far *argv;
{
  DM_ID dialogID;

  /* Initialisierung */
  if (!DM_Initialize (&argc, argv, 0))
    return (1);
```

3.39 DM_InitMSW

Diese Funktion muss in der Windows-Variante des Dialogs Manager aufgerufen werden, wenn das ausgelieferte "startup.obj" durch ein eigenes Startprogramm ersetzt worden ist. Diese Funktion übernimmt das Parsen der Kommandozeile und speichert alle wichtigen Informationen, die einem Windows-Programm beim Start als Parameter mitgegeben werden.

```
char far * far*  DML_default DM_EXPORT DM_InitMSW
(
  HANDLE hInstance,
  HANDLE hPrevInstance,
  LPSTR lpCmdLine,
  int *argc
)
```

Parameter

-> HANDLE hInstance

In diesem Parameter wir ein Zeiger auf die aktuelle Anwendungsinstanz übergeben. Den Wert hierfür erhält jedes Windows-Programm als Parameter.

-> HANDLE hPrevInstance

In diesem Parameter wir eine Zeiger auf die vorangegangene Anwendungsinstanz übergeben. Den Wert hierfür erhält jedes Windows-Programm als Parameter.

<-> LPSTR lpCmdLine

Das ist die eigentliche Kommandozeile, die der Benutzer beim Starten des Programms angegeben hat. Diese Zeile wird von dieser Funktion in einzelne Parameter zerlegt und von dieser Funktion als Rückgabewert zurückgegeben.

-> int *argc

In diesem Parameter wird die Anzahl der Parameter zurückgegeben, die von dieser Funktion aus der Kommandozeile gebildet wurden.

Rückgabewert

Diese Funktion liefert ein Array mit Strings als Ergebnis zurück. Dabei entsprechen die einzelnen Strings der zerlegten Kommandozeile.

Beispiel

Standard-Startup-Datei des Dialog Managers für Windows

```
int PASCAL WinMain __4(
(HANDLE, hInstance),
(HANDLE, hPrevInstance),
(LPSTR, lpCmdLine),
(int, nCmdShow))
{
```

```
  int argc;
  char far * far *argv;

  argv = DM_InitMSW(hInstance, hPrevInstance, lpCmdLine,
    &argc);

  if (argv)
    return (main (argc, argv));

  return (-1);
}
```

3.40 DM_InputHandler

Mit Hilfe dieser Funktion können zusätzliche Nachrichten in der Anwendung empfangen und verarbeitet werden. Die Art der Nachrichten sowie die Art des Nachrichtenempfanges ist fenstersystemabhängig. Die Funktion selber ist ebenfalls fenstersystemabhängig und wird in den nachfolgenden Kapitel dargestellt.

3.40.1 Microsoft Windows

In diesen Fenstersystemen dient der Input-Handler dafür, dass auf beliebige Nachrichten, die an Objekte verschickt wird, reagiert werden kann. Diese Nachrichten sind vom jeweiligen Fenstersystem definiert.

```
HWND DML_default DM_EXPORT DM_InputHandler
(
  DM_InputHandlerProc funcp,
  FPTR funcarg,
  DM_UInt msg,
  DM_UInt iomode,
  DM_UInt operation,
  DM_Options options
)
```

Parameter

-> DM_InputHandlerProc funcp

In diesem Parameter wird ein Zeiger auf die Funktion übergeben, die beim Eintreffen der angegebenen Nachricht aufgerufen werden soll.

-> FPTR funcarg

In diesem Parameter wird eine Zeiger auf eine von der Anwendung definierten Struktur übergeben. Diese Struktur wird dann beim Aufruf der installierten Input-Handler-Funktion an die Anwendung übergeben. Der Dialog Manager merkt sich diesen Parameter nur, ohne ihn selber zu interpretieren.

-> DM_UInt msg

In diesem Parameter wird die Message angegeben, bei deren Eintreffen die angegebene Funktion aufgerufen werden soll. Hier können alle im Fenstersystem definierten Messages angegeben werden.

-> DM_UInt iomode

In diesem Parameter wird dem Dialog Manager mitgeteilt, wie der installierte Input-Handler zu interpretieren ist. Dazu sind folgende Konstanten definiert:

iomode	Bedeutung
DMF_ModeAny	Diese Option ist nicht zulässig, wenn mit Hilfe der DM_InputHandler-Funktion ein Input-Handler installiert werden soll.
DMF_ModeMsgNotify	Diese Option besagt, dass der installierte Input-Handler nur informiert werden möchte, wenn die angegebene Nachricht eingetroffen ist. Die eigentliche Handhabung der Nachricht wird dabei vom Dialog Manager übernommen.
DMF_ModeMsgManage	Diese Option bedeutet, dass der installierte Input-Handler die Abarbeitung der angegebenen Nachricht vollständig übernimmt. Der Dialog Manager reicht diese Nachricht also nur an die angegebene Funktion weiter und verarbeitet sie nicht.

-> DM_UInt operation

In diesem Parameter wird der Funktion die eigentlich auszuführende Operation mitgeteilt. Dazu sind folgende Konstanten definiert:

operation	Bedeutung
DMF_RegisterHandler	Mit Hilfe dieses Wertes wird ein Input-Handler im Dialog Manager installiert.
DMF_WithdrawHandler	Mit Hilfe dieses Wertes wird ein vorher installierter Input-Handler deinstalliert.
DMF_DisableHandler	Mit Hilfe dieses Wertes wird ein Input-Handler vorübergehend deaktiviert.
DMF_EnableHandler	Mit Hilfe dieses Wertes wird ein deaktivierter Input-Handler wieder aktiviert.

-> DM_Options options

Normalerweise wird für diesen Parameter 0 angegeben. Wenn exakt ein Handler deinstalliert werden soll, kann über eine zusätzliche Option gesteuert werden, dass alle Funktionsargumente verglichen werden und der dieser Handler deinstalliert wird.

Option	Bedeutung
DMF_CheckFuncarg	Diese Option bedeutet, dass alle Funktionsargumente zum Suchen des Handler benützt werden sollen und nur genau der gleiche deinstalliert werden soll. Das kann dann nützlich sein, wenn mehrere Handler installiert worden sind und einer von diesen deinstalliert werden soll.

Rückgabewert

Ist der Rückgabewert ungleich HWND 0, so wird in diesem Parameter die HWND des Objektes zurückgegeben, an das der Input-Handler gebunden worden ist. Eine HWND 0 bedeutet, dass das Installieren des Input-Handler nicht möglich war.

Beispiel

Das nachfolgende Beispiel für die PC-Plattformen zeigt, wie man mit Hilfe dieser Funktion eine asynchrone Funktion

```
gethostbyname
```

implementieren kann.

```
/* Definition von statischen Variablen */
static HWND TcpWinHwnd = (HWND) 0;
    /* Definition der gewünschten Message-Nummer */
static UINT TcpWinMsgGetXByY = 0x6FE1;

/*
** Die nachfolgende Funktion lässt eine freie Message-Nummer
** berechnen. Diese übernimmt DM_ProposeInputHandlerArgs
** Das Ergebnis wird zurückgeliefert
*/
static boolean TcpWin_CheckAvail __1(
(TranspDescr *, tpdesc))
{
  DM_InputHandlerArgs InpArgs;

  /* liefert WinHandle des unsichtbaren Fensters, an dem
  ** Input-Handler haengt und gibt freie Message zurueck.
  */
  InpArgs.hwnd = (HWND) 0;
  InpArgs.message = TcpWinMsgGetXByY;
  InpArgs.wParam = (WPARAM) 0;
  InpArgs.lParam = (LPARAM) 0;
  InpArgs.mresult = (LRESULT) 0;
  InpArgs.userdata = (FPTR) 0;

  DM_ProposeInputHandlerArgs (&InpArgs, DMF_DontTrace);
  TcpWinHwnd = InpArgs.hwnd;
  TcpWinMsgGetXByY = InpArgs.message;

}

/*
** Die nachfolgende Funktion ist die eigentliche Handler-
** Funktion. Sie entnimmt die gewünschten Daten den entspre-
```

```
** chenden Strukturen
*/
static DM_Boolean DML_default DM_CALLBACK TcpWinGetXByYHandler __3(DM_
InputHandlerArgs far *, pInpArgs),
  (DM_UInt, msg),
  (DM_UInt, iomode))
  {
    if (msg == TcpWinMsgGetXByY)
    {
      if (   (WSAGETASYNCERROR (pInpArgs->lParam) == 0)
      || (WSAGETASYNCERROR (pInpArgs->lParam) == WSABASEERR))
      {
        TcpWinHostent = (struct hostent FAR *) (FPTR)
          TcpWinBuffer;
      }
      return (FALSE);
     }
     return (TRUE);
  }

/*
** Diese Funktion hat die Steuerung. Sie lässt die freie
** Message berechnen, installiert den Input-Handler und ruft
** dann die asynchrone Funktion gethostbyname auf.
*/
static struct hostent FAR * TcpWin_gethostbyname __1(
(const char FAR *, name))
{
  HANDLE h = WSAAsyncGetHostByName (TcpWinHwnd,
    TcpWinMsgGetXByY,name,TcpWinBuffer,MAXGETHOSTSTRUCT);
  TcpWinHostent = (struct hostent FAR *) 0;

  if ((h != (HANDLE) 0)
  && (DM_InputHandler (TcpWinGetXByYHandler, (FPTR) 0,
        TcpWinMsgGetXByY, DMF_ModeMsgNotify,
        DMF_RegisterHandler, DMF_DontTrace)
    != (HWND) 0)
    && DM_WaitForInput (TcpWinMsgGetXByY, 0,
    DMF_IgnoreExtEvent | DMF_DontTrace))
    {
      DM_InputHandler (TcpWinGetXByYHandler, (FPTR) 0,
        TcpWinMsgGetXByY, DMF_ModeMsgNotify,
        DMF_WithdrawHandler,
        DMF_DontTrace | DMF_CheckFuncarg);
    }
```

```
    return (TcpWinHostent);
}
```

3.40.2 Motif

In diesem Fenstersystem dient der Input-Handler dafür, dass auf beliebigen File-Deskriptoren auf Ereignisse gewartet werden kann.

So können z.B. Nachrichten, die aus anderen Prozessen kommen, verarbeitet werden.

```
DM_Boolean DML_default DM_EXPORT DM_InputHandler
(
  DM_InputHandlerProc funcp,
  FPTR funcarg,
  int FileDescriptor,
  DM_UInt iomode,
  DM_UInt operation,
  DM_Options options
)
```

Parameter

-> DM_InputHandlerProc funcp

In diesem Parameter wird ein Zeiger auf die Funktion übergeben, die beim Eintreffen von Nachrichten aufgerufen werden soll.

-> FPTR funcarg

In diesem Parameter wird ein Zeiger auf eine von der Anwendung definierten Struktur übergeben. Diese Struktur wird dann beim Aufruf der installierten Input-handler-Funktion an die Anwendung übergeben. Der Dialog Manager merkt sich diesen Parameter nur, ohne ihn selber zu interpretieren.

-> int FileDescriptor

In diesem Parameter wird ein FileDescriptor angegeben, auf dem auf eintreffende Nachrichten gewartet werden soll. Diese Nachrichten werden dann dem installierten Input-Handler übergeben.

-> DM_UInt iomode

In diesem Parameter wird dem Dialog Manager mitgeteilt, wie der installierte Input-Handler zu interpretieren ist. Dazu sind folgende Konstanten definiert:

iomode	Bedeutung
DMF_ModeAny	Diese Option ist nicht zulässig, wenn mit Hilfe der DM_InputHandler-Funktion ein Input-Handler installiert werden soll.

iomode	Bedeutung
DMF_ModeRead	Diese Option teilt dem Dialog Manager mit, dass der angegebene Input-Handler aufgerufen werden soll, wenn von dem angegebenen File-Deskriptor gelesen worden ist.
DMF_ModeWrite	Diese Option teilt dem Dialog Manager mit, dass der angegebene Input-Handler aufgerufen werden soll, wenn auf dem angegebenen File-Deskriptor geschrieben worden ist.

-> DM_UInt operation

In diesem Parameter wird der Funktion die eigentlich auszuführende Operation mitgeteilt. Dazu sind folgende Konstanten definiert:

operation	Bedeutung
DMF_RegisterHandler	Mit Hilfe dieses Wertes wird ein Input-Handler im Dialog Manager installiert.
DMF_WithdrawHandler	Mit Hilfe dieses Wertes wird ein vorher installierter Input-Handler deinstalliert.
DMF_DisableHandler	Mit Hilfe dieses Wertes wird ein Input-Handler vorübergehend deaktiviert.
DMF_EnableHandler	Mit Hilfe dieses Wertes wird ein deaktivierter Input-Handler wieder aktiviert.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Bitte geben Sie hier deshalb eine 0 an.

Rückgabe

TRUE	Der Input-Handler wurde erfolgreich installiert
FALSE	Der Input-Handler konnte nicht installiert werden.

Beispiel

```
static int FileDescr[2];

    DM_ID dialogID;
static void DM_ENTRY GenExtEvent __0()
/* ------------------------------------------------------- */
/* This function generates an external event by writing to the output
channel.*/
/* ------------------------------------------------------- */
{
```

```
    static int len, i = 0;
    static char *string[]= { "Hello, Europe",
                     "Hello, America",
                     "Hello, Africa",
                     "Hello, Asia",
                     "Hello, Australia" };

    len = strlen(string[i]);
    if ( write(FileDescr[1],string[i],len+1) < len+1 )
    {
          printf("Cannot write to pipe !\n");
          exit(1);
    }
    if ( (++i) > 4 )
      i = 0;

    if ( i>3) close(FileDescr[0]);
}

static DM_Boolean InputHandler
__3(
    (FPTR, udata),
    (int, fdescr),
    (DM_UInt, iomode)
)
/* ------------------------------------------------------ */
/* This handler function is called when the input channel */
/* is ready to be read from. */
/* ------------------------------------------------------ */
{
   char buffer[256];

   if (iomode == DMF_ModeBroken)
   {
       DM_ID obj_id;
        DM_InputHandler (InputHandler, (FPTR) 0,
               FileDescr[0], DMF_ModeRead,
               DMF_WithdrawHandler, 0) ;
       obj_id = dialogID;
       {
        DM_Value argv;

        argv.type = DT_string;
        argv.value.string = "Test";

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

```
    /* It is not allowed to set attribute values here */
    /* . So send an external event to the target object.*/
    /*                                                  */
    /* NOTE: It is recommended to set option
    /* `DMF_Synchronous' here because this handler is
    /* called synchronously */
    /* ------------------------------------------- */

    DM_QueueExtEvent (obj_id, 4712, 1,
        &argv,DMF_Synchronous);
 }
     return(TRUE);

}
if ( read(fdescr,buffer,256) > 0 )
{
    DM_ID obj_id;

    if ( obj_id = dialogID)
    {
       DM_Value argv;

       argv.type = DT_string;
       argv.value.string = buffer;

        /* ---------------------------------- */
       /* It is not allowed to set attribute values here */
      /* . So send an external event to the target object.*/
      /*                                                  */
     /* NOTE: It is recommended to set option */
    /* `DMF_Synchronous' here because this handler is */
    /* called synchronously */
    /* ------------------------------------------- */

    if ( DM_QueueExtEvent (obj_id, 4711, 1,
       &argv,DMF_Synchronous) )
    {
        return TRUE;
    }
    else
    {
        printf ("Cannot send event to Dummy_Button !\n");
        exit(1);
    }
    }
    else
```

```
        {
        printf("Cannot get ID of DummyText !\n");
        exit(1);
        }
    }
    else
    {
        printf("Cannot read from pipe !\n");
        exit(1);
    }
    return FALSE;
}

static DM_FuncMap FuncMap[] =
{
    { "GenExtEvent",   (DM_EntryFunc) GenExtEvent }
};

#define ApplFuncCount (sizeof (FuncMap)/sizeof(FuncMap[0]))

int AppInit __4
(
(DM_ID, appl),
(DM_ID, dialog),
(int, argc),
(char, **argv)
)
{

    DM_BindCallBacks (FuncMap, ApplFuncCount, appl, 0);

    dialogID = dialog;
    /* ------------------------------------------------- */
    /* Create pipe and install handler on the input channel. */
    /* ------------------------------------------------- */

    if (-1 == pipe (FileDescr))
    {
       printf("Could not create pipe !\n");
       return(1);
    }
    if ( !DM_InputHandler(InputHandler, (FPTR) 0,
            FileDescr[0], DMF_ModeRead,
            DMF_RegisterHandler, 0) )
    {
       printf("Cannot register input handler !\n");
```

```
            return(1);
        }

        return(0);
    }
```

3.41 DM_InstallNlsHandler

Mit Hilfe dieser Funktion kann eine Funktion installiert werden, die die Texte aus den Message-Katalogen bereitstellt.

```
void DML_default DM_EXPORT DM_InstallNlsHandler
(
  DM_NlsFunc func
)
```

Parameter

-> DM_NlsFunc func

Adresse der Funktion, die Texte bereitstellen kann.

Sie hat die folgende Form:

```
DM_String (DML_default DM_CALLBACK *DM_NlsFunc)
          (DM_Int4 msgno, int *codepage);
```

Diese Funktion erhält als ersten Parameter die Nummer des gewünschten Textes und als zweiten Parameter einen Pointer auf die verlangte Codepage (z.B. CP_ascii, CP_iso8859, ...).

Die Funktion gibt den Text zurück, der der Nummer zugewiesen ist. Wenn kein geeigneter Text verfügbar ist, ist es der Funktion erlaubt, einen *NULL*-Pointer zurückzugeben.

Wenn diese Funktion den Text in einer anderen Codepage zurückliefert, muss die Funktion die verwendete Codepage in **codepage* abspeichern.

Siehe auch

Ressource text

3.42 DM_InstallWSINetHandler

Mit dieser Funktion der DM-Schnittstelle werden die benutzerdefinierten Funktionen, in denen die Verschlüsselungssoftware aufgerufen wird, registriert.

Diese Funktion muss in **AppMain** vor **DM_Initialize** aufgerufen werden.

Resultatwert und Parameter der benutzerdefinierten Funktionen sind vorgegeben.

```
DM_Boolean DML_default DM_EXPORT DM_InstallWSINetHandler
(
  DM_WSINetFunctions *wsinetfunctions,
  DM_Uint Operation,
  DM_Options Options
)
```

Parameter

-> DM_WSINetFunctions *wsinetfunctions

Struktur, die die Funktionszeiger auf die benutzerdefinierten Funktionen enthält. Sie hat folgende Form:

```
DM_WSINetFunctions
{
  DMAcceptProc,
  DM_SessionProc,
  DM_ShutDownProc,
  DM_OpenProc,
  DM_CloseProc,
  DM_SendProc,
  DM_ExistsMessageProc,
  DM_RecvProc,
  DM_FreeWarningProc
}
```

Die einzelnen Funktionstypen haben folgende Form:

```
int <name of DM_AcceptProc function>
    (int serverfd, void *cliaddr, void *addrlen, char *message)
void * <name of DM_SessionProc function>
    (int clientfd, void *support, char *message)
void <name of DM_ShutDownProc function>()
int <name of DM_OpenProc function
    (int *port, void **supportptr, char *message, char **warning)
int <name of DM_CloseProc function>
    (void *connptr, int *clientfd, char *message)
int <name of DM_SendProc function>
    (void *connptr, char *buffer, int length, char message)
int <name of DM_ExistsMessageProc function>
    (void *connptr, char *message)
```

```
int <name of DM_Recv function>
    (void *connptr, char *buffer, int count, char *message)
void <name of DM_FreeWarningProc function>
    (char *warning)
```

-> DM_Uint Operation

Eine von den zwei vordefinierten Konstanten:

1. *DMF_RegisterHandler* zum Registrieren der benutzerdefinierten Funktionen.
2. *DMF_WithdrawHandler* zum Deregistrieren der benutzerdefinierten Funktionen.

-> DM_Options Options

Wird nicht benutzt und ist mit 0 vorzubelegen.

Rückgabewert

TRUE	Für *DMF_RegisterHandler*: Funktionen konnten registriert werden. Für *DMF_WithdrawHandler*: Rückgabewert immer = TRUE.
FALSE	Für *DMF_RegisterHandler*: Funktionen konnten nicht registriert werden.

3.42.1 Benutzerdefinierte Funktionen

Beschreibung der einzelnen Funktionen:

DM_AcceptProc

Akzeptiert eine Verbindung zu einem Client.

DM_SessionProc

Gibt den Descriptor des Clients als Rückgabewert *void* * aus.

Setzt Parameter für die Client-Session.

DM_ShutDownProc

Nimmt Abschlussaktionen vor beim Schließen der Verbindung.

DM_OpenProc

Konfiguriert den Socket und öffnet ihn.

DM_CloseProc

Schließt die Verbindung.

DM_SendProc

Sendet eine Message an den Client.

DM_ExistsMessage

Überprüft, ob der Client eine Message geschickt hat.

DM_RecvProc

Liest eine Message, die vom Client geschickt wurde.

DM_FreeWarningProc

Gibt Speicherplatz, der in **DM_OpenProc** für den Parameter *warning* allokiert wurde, frei.

Reihenfolge des Aufrufs ist normalerweise

1. DM_OpenProc function
2. DM_FreeWarningProc function
3. DM_AcceptProc function
4. DM_SessionProc function
5. DM_SendProc function, DM_Recv function, DM_ExistsMessageProc function im Wechsel
6. DM_CloseProc function
7. DM_ShutDownProc function

Falls eine eigene Verschlüsselung mit Hilfe der benutzerdefinierten Funktionen verwirklicht werden soll, muss beim Benutzen der Parameter als Ein- und Ausgabe die vom IDM vorgegebene Reihenfolge der Aufrufe berücksichtigt werden.

3.43 DM_LoadDialog

Mit Hilfe dieser Funktion können die Dialoge in die Anwendung geladen werden.

```
DM_ID DML_default DM_EXPORT DM_LoadDialog
(
  DM_String path,
  DM_Options options
)
```

Parameter

-> DM_String path

In diesem Parameter wird mit Hilfe eines Pfades die zu ladende Datei angegeben. Hierbei gilt wie bei allen Dateizugriffen, dass der angegebene Name wie folgt aufgebaut sein darf:

Umgebungsvariable:Name der Dialogdatei.

Die vorangestellte Umgebungsvariable dient als Pfad, unter dem die Dialogdatei gesucht werden soll.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Bitte geben Sie hier deshalb eine 0 an.

Rückgabewert

0	Die angegebene Datei ist keine fehlerfreie DM-Datei oder die Datei wurde vom DM nicht gefunden.
!=0	Identifikator des vom DM geladenen Dialogs.

Der Dialog Manager ist in der Lage, die Dialogdaten sowohl aus einem ASCII-File als auch einer Binärdatei zu laden. Der Vorteil der Binärdatei gegenüber des ASCII-File liegt vor allem in der wesentlich kürzeren Ladezeit, da hierbei keinerlei interne Überprüfungen durchgeführt werden müssen. Das Erzeugen der Binärdatei erfolgt dabei mit folgendem Befehl:

```
idm +writebin <Name der Binärdatei> <Name der ASCII-Datei>
```

Beispiel

Auszug aus einer AppMain-Funktion:

```
/*
** Laden des Dialogs. Standardname im Programm vorgegeben,
** kann über Kommandozeile überschrieben werden
*/
switch(argc)
{
  case 1:
      dialogID = DM_LoadDialog (dialogfile, 0);
```

```
  break;
  case 2:
      dialogfile = argv[1];
      dialogID = DM_LoadDialog (dialogfile, 0);
  break;
  default:
      DM_TraceMessage("Zuviele Argumente ",
          DMF_LogFile | DMF_InhibitTag);
  return(0);
  break;
}

if (!dialogID)
{
  DM_TraceMessage("%s: Could not load dialog \"%s\"",
          DMF_LogFile | DMF_InhibitTag | DMF_Printf,
          argv[0], dialogfile);
  return(1);
}
```

Siehe auch

Eingebaute Funktion load im Handbuch „Regelsprache“

3.44 DM_LoadProfile

Mit dieser Funktion können die vom Benutzer veränderbaren Variablen aus einer Datei eingelesen und vom Dialog Manager verarbeitet werden. Damit ist es möglich, Dialoge durch den Endanwender beeinflussbar zu machen, ohne dass dieser Endanwender den Dialog in Source oder den Dialog Manager hat.

```
DM_Boolean DML_default DM_EXPORT DM_LoadProfile
(
  DM_ID dialog,
  DM_String filename,
  DM_Options options
)
```

Parameter

-> DM_ID dialog

Dieses ist der Identifikator des Dialoges, zu dem das angegebene Profile eingelesen werden soll.

-> DM_String filename

Dieser Name bezeichnet den Namen der Profile-Datei.

Hierbei gilt wie bei allen Dateizugriffen, dass der angegebene Name wie folgt aufgebaut sein darf:

Umgebungsvariable:Name der Dialogdatei.

Die vorangestellte Umgebungsvariable dient als Pfad, unter dem die Dialogdatei gesucht werden soll.

In der Dialogbeschreibungsdatei steht z.B.

```
config variable integer HUGO;
```

Diese Variable soll nun mittels der Profile-Datei gesetzt werden. Die Datei sieht dann wie folgt aus:

```
HUGO := 5;
```

-> DM_Options options

Unbenutzt. Muss 0 sein.

Rückgabewert

TRUE	Datei konnte eingelesen werden.
FALSE	Datei konnte nicht eingelesen werden.

Beispiel

Dialogdatei

```
dialog YourDialog
{
```

```
}
config variable string WindowText := "Sorry no profile";
config variable integer WindowXPos := 5;
config variable integer WindowYPos := 5;

window W1
{
    .width 25;
    .title "Testwindow";
    .visible false;
    .xraster 10;
    .yraster 16;
    .posraster true;
    .sizeraster true;
    child pushbutton End

    {
    .xleft 7;
    .width 9;
    .height 2;
    .text "End";
    }
}
on End select
{
    exit ();
}

on dialog start
{
    W1.xleft :=WindowXPos;
    W1.ytop :=WindowYPos;
    W1.title :=WindowText;
    W1.visible :=true;
}
```

Profile

```
WindowXPos:=10;
WindowYPos:=5;
WindowText:="Out Of Profile";
```

Dieses Profile kann nur über **DM_LoadProfile** in die Applikation bzw. über **-profile <Dateiname>** in den Simulator geladen werden.

Siehe auch

C-Funktion DM_SaveProfile

3.45 DM_Malloc

Mit Hilfe dieser Funktion können Sie Speicherplatz allokieren. Diese Allokierung erfolgt dabei abhängig vom zugrundeliegenden Betriebssystem mit den dort verfügbaren Funktionen. Über diese Funktionen allokierter Speicher darf nur mit der Funktion DM_Free freigegeben bzw. mit DM_Realloc verändert werden.

```
DM_Pointer DML_default DM_EXPORT DM_Malloc
(
  DM_UInt4 size
)
```

Parameter

-> DM_UInt4 size

Dieser Parameter gibt die Größe des neu zu allokierenden Speicherbereichs an.

Warnung

Dieser Speicherbereich darf nicht **> 64 KByte** sein, falls die Anwendung mit Microsoft Windows laufen soll!

Rückgabewert

Pointer auf den allokierten Speicherbereich. Konnte der Speicher nicht allokiert werden, wird der *NULL*-Pointer zurückgegeben.

Beispiel

Für einen String soll Speicherplatz angelegt werden.

```
char * string;

if ((string = DM_Malloc(20)))
    strcpy (string, "1234567");
```

3.46 DM_NetHandler

Mit Hilfe dieser Funktion kann ein NetHandler angemeldet werden. Ein NetHandler ist eine benutzerdefinierte Funktion, die der IDM unmittelbar vor dem Versenden und unmittelbar nach dem Empfangen von Netzpaketen mit dem Inhalt dieser Pakete aufruft.

Der Distributed Dialog Manager schickt alle Daten im Klartext übers Netz, das heißt, die Daten sind für jedermann lesbar. Um vertrauliche Daten vor Unbefugten zu schützen, benötigt der IDM-Anwender eine Möglichkeit, Daten die über das Netz geschickt werden, zu verschlüsseln. Diese Möglichkeit wird ihm mit der Anmeldung von NetHandlern gegeben, in denen der IDM-Anwender eine Verschlüsselung vornehmen kann.

Außer einer Verschlüsselung sind natürlich auch andere Anwendungen von NetHandlern denkbar.

```
void DML_default DM_EXPORT DM_NetHandler
(
  DM_NetHandlerProc NetHandler,
  DM_UInt           Operation,
  DM_Options        Options
)
```

Parameter

-> DM_NetHanderProc NetHandler

Funktionszeiger auf den benutzerdefinierten Handler. Der Handler muss das folgende Format aufweisen:

```
void DML_default DM_Callback <ProcName> (reason)
DM_NetInfo far *reason;
{
  /* Code des Anwenders */
}
```

-> DM_UInt Operation

Dieser Parameter kann mit folgenden Werten belegt sein:

- *DMF_RegisterHandler*
  Anmelden des Handlers
- *DMF_WithdrawHandler*
  Abmelden des Handlers
- *DMF_EnableHandler*
  Einschalten des Handlers (noch nicht implementiert)
- *DMF_DisableHandler*
  Abschalten des Handlers (noch nicht implementiert)

Nach seiner Registrierung ist ein NetHandler automatisch eingeschaltet. Es können mehrere NetHandler installiert werden, aber jeder Handler höchstens einmal.

-> DM_Options Options

Wird nicht benutzt und ist mit *0* zu initialisieren.

Bitte beachten Sie, dass solche NetHandler, die Datenpakete modifizieren, beim sendenden und empfangenden Prozess unbedingt synchron angemeldet werden müssen. Wenn der eine Prozess zum Beispiel bereits verschlüsselt sendet, der andere Prozess aber noch unverschlüsselte Daten erwartet, kann dies zum Absturz der Prozesse oder sogar des Systems führen.

Der Aufruf der angemeldeten Handler erfolgt in der umgekehrten Reihenfolge, in der sie angemeldet wurden. Es gibt dazu eine Ausnahme, die später erläutert wird.

Den NetHandlern wird beim Aufruf eine Struktur **DM_NetInfo** übergeben, die alle erforderlichen Informationen und die Daten des Netzpaketes enthält:

```
struct {
  char        *data;   /* Datenpaket */
  DM_Integer  length;  /* Datenumfang in Bytes */
  DM_Integer  action;  /* Kommunikationsvorgang */
  DM_Integer  error;   /* Fehlernummer */
  int         socket;  /* Kommunikationsschnittstelle */
  DM_ID       applID;  /* Objekt-ID der Applikation */
} DM_NetInfo;
```

Es gibt fünf vordefinierte Konstanten, von denen eine im Strukturelement *action* dem Handler mitgegeben wird:

- *DM_NET_SEND*
- *DM_NET_RECEIVE*
- *DM_NET_CONNECT*
- *DM_NET_MESSAGE*
- *DM_NET_ERROR*
- *DM_NET_MESSAGE*
- *DM_NET_CONNECTMESSAGE*

Bei *DM_NET_SEND* soll das übergebene Datenpaket verschickt werden, bei *DM_NET_RECEIVE* wurde es empfangen. Bei diesen Vorgängen kann der Inhalt des Datenpaketes manipuliert, zum Beispiel ver- bzw. entschlüsselt, werden.

Die Konstante *DM_NET_CONNECT* besagt, dass gerade eine Verbindung aufgebaut wird. Dabei darf der Inhalt des Datenpakets nicht verändert werden, sonst schlägt der Verbindungsaufbau fehl.

DM_NET_CONNECTMESSAGE kennzeichnet die STDOUT-Ausgabe zum Verbindungsaufbau der mittels *.exec* gestarteten Anwendungsseite. Diese muss unverändert an IDM weitergeleitet werden.

DM_NET_MESSAGE kennzeichnet die STDOUT-Ausgabe der mittels *.exec* gestarteten Anwendungsseite. Die Ausgabe wird Zeilenweise an den NetHandler weitergegeben (Ausnahme wenn der Socket geschlossen wird). Achtung: Als erstes wird ein 0-Byte gesendet (rsh-Protokol ?) und der IDM

schickt einen Zeilenumbruch vor dem Verbindungsaufbau.
Achtung: 0-Bytes bleiben als 0-Bytes erhalten. Die Länge des Strings sollte unbedingt beachtet werden.

Ist die Konstante *DM_NET_RECEIVE* gesetzt, werden die angemeldeten NetHandler in derselben Reihenfolge aufgerufen, in der sie angemeldet wurden. Dieser Fall bildet die oben erwähnte Ausnahme. Sie ist erforderlich, weil beim Senden vorgenommene Datenmanipulationen durch mehrere Handler beim Empfang in umgekehrter Reihenfolge wieder rückgängig gemacht werden müssen. Wird vor dem Senden beispielsweise zuerst ein protokollierender und dann ein verschlüsselnder Handler aufgerufen, müssen die Daten nach dem Empfang erst vom verschlüsselnden Handler entschlüsselt werden, bevor sie vom protokollierenden Handler aufgezeichnet werden können.

Datenmanipulationen, die bei *DM_NET_SEND* und *DM_NET_RECEIVE* vorgenommen werden, müssen invers zueinander sein, d.h. Daten, die beim Empfänger mit *DM_NET_RECEIVE* durch den Handler geschleust wurden, müssen identisch sein mit den Daten, die zuvor beim Sender mit *DM_NET_SEND* durch den Handler geschleust wurden.

Dabei ist die Menge der Datenpakete als Stream zu betrachten, denn was vom Sender als ein Paket seinem Handler übergeben wird, kommt beim Handler des Empfängers nicht unbedingt als ein Paket an. Dazwischen können die TCP/IP-Implementation oder andere Netzwerkkomponenten noch Puffer geschaltet haben. Folglich müssen eventuelle Manipulationen an den Paketinhalten unabhängig von der Position innerhalb eines Pakets sein.

Der Zeiger *data* auf den Inhalt des Netzpaketes darf vom Handler nicht verändert werden. Auch ein Reallokieren von *data* an derselben Adresse ist nicht erlaubt. Die Datenpakete können also nicht vergrößert werden, weshalb keine Dekomprimierung komprimierter Daten möglich ist.

Beispiel

```
/* NetHandler zum Verschluesseln von Daten
   durch Bit-Invertierung */
void DML_default DM_CALLBACK MyNetHandler
  __1((DM_NetInfo *, info)) {
  switch (info->action) {
    case DM_NET_SEND:
    case DM_NET_RECEIVE: {
      int ZaVa;
      for(ZaVa=0;ZaVa<info->length;ZaVa++)
        (info->data)[ZaVa]=~(info->data)[ZaVa];
        break;
    }
  }
}
```

Die Anmeldung des Handlers bei Client und Server kann zum Beispiel jeweils vor der Anbindung der C-Funktionen an den IDM geschehen:

```
if (!DM_NetHandler(MyNetHandler,DMF_RegisterHandler,0))
  DM_TraceMessage("NetHandler registering error", 0);
```

3.47 DM_OpenBox

Mit Hilfe dieser Funktion kann eine entsprechend angegebene ***Messagebox*** oder **Dialogbox** (***Fenster*** mit gesetztem Attribut *.dialogbox* = *true*) geöffnet werden. Das Programm wartet bis der Benutzer diese Messagebox bzw. Dialogbox wieder geschlossen hat.

Hinweis

Bei Verwendung in Funktionen, welche ***Records*** als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“ und das Kapitel „Hinweis bei Verwendung von DM Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“.

```
DM_Boolean DM_OpenBox
(
  DM_ID       objectID,
  DM_ID       parentID,
  DM_Value   *retval,
  DM_Options  options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter spezifiziert die ***Messagebox*** bzw. **Dialogbox**, die geöffnet werden soll.

-> DM_ID parentID

Dieser Parameter spezifiziert das ***Fenster***, in dem die ***Messagebox*** erscheinen soll. Der Parameter kann ignoriert werden. Falls dieser Parameter angegeben ist, muss ein ***Fenster*** oder *NULL* angegeben werden.

Falls das Fenstersystem dies unterstützt, wird die ***Messagebox*** über dem Vater-Fenster zentriert dargestellt. Andernfalls wird die Position vom Fenstersystem selbst festgelegt (z.B. Bildschirmmitte).

-> DM_Value *retval

Enthält nach Schließen der ***Messagebox*** bzw. **Dialogbox** den Rückgabewert des jeweiligen Objekts:

- Bei ***Messageboxen*** die Nummer des gedrückten Buttons. Dafür gibt es folgende Definitionen:
  - *MB_abort*
  - *MB_cancel*
  - *MB_ignore*
  - *MB_no*
  - *MB_ok*

- *MB_retry*
- *MB_yes*

» Bei **Dialogboxen** den in der Funktion closequery definierten Wert.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit *0* belegt sein.

Rückgabewert

Der Rückgabewert sagt aus, ob die ***Messagebox*** bzw. **Dialogbox** geöffnet werden konnte.

Siehe auch

C-Funktion DM_QueryBox

Objekt Messagebox

Eingebaute Funktion querybox im Handbuch „Regelsprache“

3.48 DM_ParsePath

Mit dieser Funktion kann der Identifikator eines Objekts erfragt werden, wenn mehr als ein Dialog geladen ist und sich das gesuchte Objekt nicht im zuerst geladenen Dialog befindet.

```
DM_ID DML_default DM_EXPORT DM_ParsePath
(
  DM_ID      dialogid,
  DM_ID      rootid,
  DM_String  path,
  DM_UInt    idx,
  DM_Options options
)
```

Parameter

-> DM_ID dialogid

Dies ist der Identifikator des Dialogs, in dem nach dem Objekt gesucht werden soll.

-> DM_ID rootid

Dieser Parameters steuert, ab welchem Objekt der IDM die Suche nach dem Objekt beginnt. Dabei gibt es folgende Möglichkeiten:

- *rootid = 0*
  Der IDM sucht in der gesamten Dialogdefinition nach dem angegebenen Objekt.
  Dies ist der Normalfall. Auf diese Art können auch die Identifikatoren von Regeln, Funktionen, Variablen und Ressourcen erfragt werden.
- *rootid != 0*
  Der IDM sucht ab dem angegebenen Objekt nur auf der nächsttieferen Hierarchiestufe; tiefere Hierarchiestufen werden nicht durchsucht.
  Diese Vorgehensweise ist nur dann angebracht, wenn in einem Dialog ein Objektname mehr als einmal auftritt.

-> DM_String path

Dieser Pfad definiert das gesuchte Objekt. Er muss eindeutig ein Objekt beschreiben.

Existiert der Objektname nur einmal innerhalb des Dialogs, so reicht die Angabe des Namens, um die gewünschte Referenz zu erhalten. Ist der Objektname nicht eindeutig, muss das Objekt über einen Pfad von Objektnamen getrennt durch einen Punkt beschrieben werden.

-> DM_UInt idx

In diesem Parameter wird angegeben, das wievielte Vorkommen eines Objekts mit dem angegebenen Namen gesucht werden soll.

Der Zähler beginnt bei *0*. Um das erste Vorkommen eines Objekts zu suchen, ist hier *0* anzugeben.

-> DM_Options options

Dieser Parameter wird zur Zeit nicht benutzt.

Rückgabewert

0 Das gesuchte Objekt wurde nicht gefunden oder der Name ist nicht eindeutig.
Das heißt, es gibt keins oder mehrere Objekte mit dem angegebenen Namen.

!= 0 Identifikator des gesuchten Objekts.

Anmerkung

Wenn *"setup"* als *path*angegeben wird und sowohl *dialogid* als auch *rootid* gleich *0* sind, dann wird das ***Setup***-Objekt zurückgegeben.

Beispiel

```
void DML_default DM_ENTRY OkButtonCallback __1((DM_ID, dialogID))
{
  DM_ID ID1;
  DM_ID ID2;

  /* Abfrage eines Objektes im Dialog global */
  ID1 = DM_ParsePath(dialogID, 0, "FirstObject", 0, 0);

  /* Abfrage über einen Pfad */
  ID2 = DM_ParsePath(dialogID, 0, "FirstObject.Child1", 0, 0);
}
```

Siehe auch

Eingebaute Funktion parsepath im Handbuch „Regelsprache“

3.49 DM_PathToID

Achtung

Die Funktion **DM_PathToID** ist veraltet und wird nur noch aus Kompatibilitätsgründen unterstützt. An ihrer Stelle sollte DM_ParsePath verwendet werden.

Mit Hilfe dieser Funktion wird der Ihnen bekannte externe Name eines Objekts in die interne Bezeichnung umgewandelt. Diese interne Bezeichnung eines Objekts ist über einen Programmlauf hinweg konstant, so dass Sie die ID eines oft benötigten Objekts nicht bei jedem Zugriff erfragen müssen.

```
DM_ID DML_default DM_EXPORT DM_PathToID
(
  DM_ID rootid,
  DM_String path
)
```

Parameter

-> DM_ID rootid

Mit Hilfe dieses Parameters können Sie steuern, ab welchem Objekt der Dialog Manager die Suche nach dem von Ihnen gewünschten Objekt beginnt. Dabei gibt es folgende Möglichkeiten:

» *rootid = 0*
Der Dialog Manager sucht in der gesamten Dialogdefinition außer in Modulen nach dem angegebenen Objekt.
Dies ist der Normalfall. Auf diese Art können auch die Identifikation von Regeln, Funktionen, Variablen und Ressourcen erfragt werden.

» *rootid != 0*
Der Dialog Manager soll ab dem angegebenen Objekt auf der nächsttieferen Hierarchiestufe suchen. Wenn das Objekt ein Modul ist, wird in diesem Modul nach dem angegebenen Objekt gesucht.
Diese Vorgehensweise ist nur dann angebracht, wenn in einem Dialog ein Objektname mehr als einmal auftritt.
Regeln, Funktionen, Variablen und Ressourcen können auf diese Weise nicht erfragt werden.

-> DM_String path

Mit Hilfe dieses Pfades wird das gesuchte Objekt beschrieben. Dieser Pfad muss eindeutig ein Objekt beschreiben. Existiert der Objektname nur einmal innerhalb des Dialoges, so reicht die Angabe des Namens, um die gewünschte Referenz zu erhalten. Ist der Objektname nicht eindeutig, muss das Objekt über einen Pfad von Objektnamen, getrennt durch einen Punkt, beschrieben werden.

Rückgabewert

0 Das gesuchte Objekt wurde nicht gefunden oder der Name ist nicht eindeutig.

!= 0 Identifikator des gesuchten Objekts.

Mit Hilfe des so erhaltenen Identifikators eines Objekts können Sie nun auf dessen Attribute zugreifen.

Beispiel

```
DM_Value value;
DM_ID id;

/* The file could be opened, enable the other objects */
value.type = DT_boolean;
value.value.boolean = TRUE;

/* Get the id of the edittext "Actives" */
if ((id=DM_PathToID(0, "Actives")))
  /* Change the object to sensitive */
  DM_SetValue(id, AT_sensitive, 0, &value, DMF_ShipEvent);
```

Siehe auch

Funktion DM_ParsePath

Eingebaute Funktion parsepath im Handbuch „Regelsprache“

3.50 DM_PictureHandler

Mit einem benutzerdefinierten Grafik-Handler (GFX-Handler) können Bilder in Grafikformaten, die der IDM nicht unterstützt, geladen und angezeigt werden. Dafür ruft der IDM jedes Mal, wenn ein Bild geladen werden muss (weil es z.B. über eine ***Tile***-Ressource in der Anwendung angebunden wurde), als erstes die angemeldeten Grafik-Handler auf. Kann einer dieser Handler das Bild laden, übergibt er die Bilddaten an den IDM und es werden keine weiteren Grafik-Handler aufgerufen. Wenn kein Grafik-Handler das Bild laden konnte, versucht der IDM selbst das Bild zu parsen. Das Verhalten ist dann so, als ob keine Grafik-Handler benutzt wurden.

Grafik-Handler werden mit der Funktion DM_PictureReaderHandler beim IDM an- und abgemeldet.

Ein Grafik-Handler muss wie folgt definiert sein:

```
DM_Boolean DML_default DM_CALLBACK <ProcName>
(
  DM_PicInfo * pic
)
{
  /* benutzerdefinierter Code */
}
```

Beim Aufruf wird dem Grafik-Handler ein Zeiger auf die Struktur **DM_PicInfo** übergeben. Diese Struktur enthält alle Informationen über ein Bild, das geladen oder dessen Speicherplatz freigegeben werden soll.

```
typedef struct {
  DM_Integer  struct_size;  // Groesse der Struktur zur Verifikation der
Version
  DM_UInt1    task;         // Task fuer den Grafik-Handler
  DM_String   fname;        // Dateiname des Bildes
  DM_String   name;         // Name des Bildes wie an tile oder image
angegeben
#if defined(WSIWIN)
  DM_UInt2    width;        // Breite des Bildes
  DM_UInt2    height;       // Hoehe des Bildes
  DM_UInt2    type;         // Bildtyp
  HANDLE      palette;      // Handle fuer die Farbpalette
  HANDLE      image;        // Handle fuer das Bild
  HANDLE      trans_mask;   // Handle fuer die Transparenzmaske
#endif
#if defined(MOTIF) || defined(QT)
  DM_UInt2    type;         // Bildtyp
  DM_Pointer  image;        // Zeiger auf die Bilddaten
  DM_Pointer  trans_mask;   // Zeiger auf die Transparenzmaske
#endif
  DM_Integer  trans_color;  // Index der transparenten Farbe
} DM_PicInfo;
```

Bedeutung der Elemente

DM_Integer struct_size

Hier wird die Größe der Struktur übergeben. Damit kann im Grafik-Handler mit **sizeof()** überprüft werden, ob die Größe der übergebenen Struktur mit der verwendeten Strukturdefinition übereinstimmt.

DM_UInt1 task

Definiert, was der Grafik-Handler tun soll. Es sind zwei Aufgaben möglich:

- *DM_GFX_TASK_LOAD*
  Laden eines Bildes.
  Der Grafik-Handler muss eine Bilddatei mit dem Dateinamen *fname* laden und folgende Elemente der **DM_PicInfo**-Struktur setzen.
  - *width* (erforderlich unter MICROSOFT WINDOWS; sonst in den Bilddaten enthalten)
  - *height* (erforderlich unter MICROSOFT WINDOWS; sonst in den Bilddaten enthalten)
  - *type* (erforderlich)
  - *palette* (optional; nur MICROSOFT WINDOWS)
  - *image* (erforderlich)
  - *trans_mask* (optional; unterstützt ab IDM-Version A.05.02.e)
  - *trans_color* (optional; zurzeit nicht ausgewertet)
- *DM_GFX_TASK_UNLOAD*
  Speicher für die Bilddaten freigeben.
  In diesem Fall sind folgende Strukturelemente gesetzt, damit der Grafik-Handler den Speicherplatz dieser Objekte freigeben kann:
  - *palette* (falls vorhanden; nur MICROSOFT WINDOWS)
  - *image*
  - *trans_mask* (falls vorhanden; unterstützt ab IDM-Version A.05.02.e)

DM_String fname

Nur bei *task* == *DM_GFX_TASK_LOAD* gültig.

In diesem Element werden Pfad und Dateiname des Bildes angegeben, das geladen werden soll.

Der IDM überprüft, ob das Bild geladen werden kann. Wenn das der Fall ist, enthält *fname* den aufgelösten Dateinamen des Bildes.

Wenn das Bild nicht geladen, d.h. der Dateiname nicht aufgelöst werden kann, dann ist *name* == *fname*.

DM_String name

Nur bei *task* == *DM_GFX_TASK_LOAD* gültig.

Das Element enthält den unaufgelösten Dateinamen des Bildes, wie er an der ***Tile***-Ressource bzw. am ***Image***-Objekt angegeben wurde.

Das Strukturelement ist ab IDM-Version A.05.02.e vorhanden.

DM_Integer trans_color

Hier kann die transparent darzustellende Farbe übergeben werden.

Das Element wird zurzeit nicht ausgewertet.

Belegung der Strukturelemente unter Microsoft Windows

DM_UInt2 width
DM_UInt2 height

Nur bei *task* == *DM_GFX_TASK_LOAD* gültig.

In diesen Elementen muss der Grafik-Handler als Rückgabewert die Breite und Höhe des geladenen Bildes übergeben.

DM_UInt2 type

In diesem Element muss der Grafik-Handler den Typ des Handles zurückgeben, der in *image* übergeben wird.

Wertebereich

- *DM_GFX_BMP* Windows Bitmap
- *DM_GFX_WMF* Windows Meta File
- *DM_GFX_EMF* Enhanced Meta File
- *DM_GFX_ICO* Windows Icon

HANDLE palette

Für *task* == *DM_GFX_TASK_LOAD*

Der Grafik-Handler kann zusammen mit dem Bild eine Farbpalette (Windows Handle Type *HPALETTE*) an den IDM übergeben. Ist dieses Element *NULL*, wird die System-Farbpalette verwendet.

Im Allgemeinen sollte darauf verzichtet werden, für jedes Bild eine eigene Farbpalette zu verwenden, da es beim Fokuswechsel auf Systemen mit geringer Farbtiefe zu Farbverfälschungen kommen kann. Stattdessen sollte der Grafik-Handler die Farben eines Bildes mit der System-Farbpalette abgleichen, und so für die Farbechtheit des Bildes sorgen.

Für *task* == *DM_GFX_TASK_UNLOAD*

Wenn eine Farbpalette verwendet wurde, wird in diesem Element deren Handle vom IDM an den Grafik-Handler übergeben, damit der Handler den Speicher für die Farbpalette freigeben kann.

HANDLE image

Für *task* == *DM_GFX_TASK_LOAD*

In diesem Element muss der Grafik-Handler den Handle des von ihm geladenen Bildes zurückgeben. Der Typ des Handles muss mit der Angabe in *type* übereinstimmen.

Für *task* == *DM_GFX_TASK_UNLOAD*

In diesem Element wird vom IDM der Handle des Bildes, dessen Speicher freizugeben ist, an den Grafik-Handler übergeben.

HANDLE trans_mask

Dieses Strukturelement wird erst ab IDM-Version A.05.02.e ausgewertet.

Für *task* == *DM_GFX_TASK_LOAD*

In diesem Element kann eine Transparenzmaske übergeben werden. Die Transparenzmaske muss eine monochrome Windows Bitmap (Datentyp *HBITMAP*) mit der gleichen Größe wie das in *image* übergebene Bild sein.

Eine Transparenzmaske wird nur für den Bildtyp *DM_GFX_BMP* unterstützt, bei anderen Bildtypen wird sie ignoriert.

Für *task* == *DM_GFX_TASK_UNLOAD*

Wenn eine Transparenzmaske verwendet wurde, wird in diesem Element deren Handle vom IDM an den Grafik-Handler übergeben, damit der Handler den Speicher für die Transparenzmaske freigeben kann.

Belegung der Strukturelemente unter Motif und Qt

DM_UInt2 type

In diesem Element muss der Grafik-Handler den Typ des Bildes zurückgeben, das in *image* übergeben wird

Wertebereich beim IDM FÜR MOTIF

- *DM_GFX_XIMAGE* XImage-Bild
- *DM_GFX_PIXMAP* PixMap-Bild (unterstützt ab IDM-Version A.05.02.e)

Wertebereich beim IDM FÜR QT

- *DM_GFX_QPIXMAP*
- *DM_GFX_QIMAGE*
- *DM_GFX_QICON*

DM_Pointer image

Für *task* == *DM_GFX_TASK_LOAD*

In diesem Element muss der Grafik-Handler entsprechend dem *type*-Element einen Zeiger auf ein PixMap-Bild oder eine XImage-Struktur mit den Bilddaten übergeben.

Die XImage-Struktur kann unter X Windows mit der Xlib-Funktion **XCreateImage()** generiert werden. Hierfür sind Informationen über aktuelle Bildschirmeinstellungen (z.B. Display, Visual, Screen, Depth) notwendig. Diese Daten können mit der Schnittstellenfunktion DM_GetToolkitData abgefragt werden.

Für *task == DM_GFX_TASK_UNLOAD*

In diesem Element wird vom IDM ein Zeiger auf die Bilddaten, deren Speicher freizugeben ist, an den Grafik-Handler übergeben.

DM_Pointer trans_mask

Dieses Strukturelement wird erst ab IDM-Version A.05.02.e ausgewertet.

Für *task == DM_GFX_TASK_LOAD*

In diesem Element kann ein Zeiger auf eine Transparenzmaske übergeben werden. Unter MOTIF muss die Transparenzmaske ein PixMap-Bild sein, unter QT muss ihr Datentyp dem in *type* angegebenen Bildtyp entsprechen. Die Transparenzmaske muss die gleiche Größe wie das in *image* übergebene Bild haben.

Für *task == DM_GFX_TASK_UNLOAD*

Wenn eine Transparenzmaske verwendet wurde, wird in diesem Element ein Zeiger darauf vom IDM an den Grafik-Handler übergeben, damit der Handler den Speicher für die Transparenzmaske freigeben kann.

Rückgabewert

DM_TRUE	Das Bild konnte geladen werden.
DM_FALSE	Im Fehlerfall.

Anmerkungen

Grafik-Handler sind systemabhängig. Wie die **DM_PicInfo**-Struktur zeigt, unterscheiden sich je nach System die Datenformate für die Rückgabe von Bilddaten.

Da ein Grafik-Handler zum Laden eines Bildes Speicher allokieren muss (z.B. für *image*, *palette* und *trans_mask*), muss er auch mit der Freigabe dieses Speichers beauftragt werden. Deshalb ruft der IDM die Grafik-Handler mit *task == DM_GFX_TASK_LOAD* auf, wenn ein Bild gebraucht wird. Wenn das Bild nicht mehr gebraucht wird, werden die Grafik-Handler mit *task == DM_GFX_TASK_UNLOAD* aufgerufen, damit sie den Speicherplatz freigeben können.

Der erste Grafik-Handler, der den Speicher freigibt, liefert *DM_TRUE* zurück, sodass danach keine weiteren Handler aufgerufen werden. Dies muss nicht unbedingt der selbe Handler sein, der das Bild geladen hat.

Vom IDM erfolgt keine Zuordnung zwischen einem Bild und dem für das Laden und somit auch für das Freigeben des Bildes zuständigen Grafik-Handler. Wenn z.B. alle Grafik-Handler den Speicher auf die gleiche Art und Weise allokieren, dann kann auch jeder Handler den Speicher eines anderen freigeben. Das heißt, der erste aufgerufene Handler gibt sofort den Speicher frei.

Beispiel

Ein typischer GFX-Handler hat in etwa folgende Struktur:

```
DM_Boolean DML_default DM_CALLBACK MyGfxHandler __1((DM_PicInfo *, pic))
{
  /* zuerst feststellen, was zu tun ist */
  if (pic->task == DM_GFX_TASK_LOAD) {
    /* das eigentliche Laden erfolgt in LoadPicture_... */

#if defined(WIN32)
    /* Bild laden (Microsoft Windows) */
    pic->image = LoadPicture_Win32 (pic->fname);
#endif
#if defined(MOTIF) || defined(QT)
    /* Bild laden (XWindows) */
    pic->image = LoadPicture_X (pic->fname);
#endif

    if (pic->image) {
      /* Rueckgabewerte in pic einragen */

#if defined(WIN32)
      pic->type    = DM_GFX_...;
      pic->palette = Handle auf Palette, falls gewuenscht;
      pic->width   = Breite des Bildes;
      pic->height  = Hoehe des Bildes;
#endif
#if defined(MOTIF) || defined(QT)
      pic->type = DM_GFX_...;
#endif

      return DM_TRUE;  /* Erfolg */
    } else {
      /* Bild konnte nicht geladen werden, vielleicht ist ein anderer
         GFX-Handler oder der Grafikparser des IDM erfolgreicher */
      return DM_FALSE;
    }
  } else {
    /* Speicher freigeben */

#if defined(WIN32)
    if (pic->image)       /* hier geladen */
      DeleteObject(pic->image);
    if (pic->palette)     /*  hier generiert */
      DeleteObject(pic->palette);
#endif
#if defined(MOTIF) || defined(QT)
    if (pic->image)       /* hier geladen */
```

```
      XDestroyImage((XImage *)pic->image);
    if (pic->trans_mask)  /*  hier generiert */
      free(pic->trans_mask);
#endif

    return DM_TRUE;
  }
}
```

Der GFX-Handler sollte vor dem Laden von Bilddateien und somit in der Regel vor dem Laden des Dialogs angemeldet werden:

```
if (!DM_PictureReaderHandler (MyGfxHandler, DMF_RegisterHandler, 0))
  DM_TraceMessage("GFX-Handler registering error", 0);
...
/* Dialog laden */
```

Siehe auch

Funktion DM_PictureReaderHandler

3.51 DM_PictureReaderHandler

Mit dieser Funktion können benutzerdefinierte Grafik-Handler (GFX-Handler) beim IDM registriert und verwaltet werden. Diese Handler werden vom IDM zum Laden von Bildern aufgerufen, die bei ***Tile***-Ressourcen oder ***Image***-Objekten angegeben sind.

Benutzerdefinierte Grafik-Handler sind in DM_PictureHandler beschrieben.

```
DM_Boolean DML_default DM_EXPORT DM_PictureReaderHandler
(
  DM_PictureReaderProc  funcp,
  DM_UInt  operation,
  DM_UInt  options
)
```

Parameter

-> DM_PictureReaderProc funcp

Zeiger auf einen Grafik-Handler, für den die in *operation* definierte Aktion ausgeführt erden soll.

Der Handler muss folgendes Format aufweisen:

```
DM_Boolean DML_default DM_CALLBACK <ProcName>
(
  DM_PicInfo * pic
)
{
  /* benutzerdefinierter Code */
}
```

Entsprechend ist der Datentyp *DM_PictureReaderProc* wie folgt definiert:

```
typedef DM_Boolean (DML_default DM_CALLBACK * DM_PictureReaderProc) __
((DM_PicInfo * pic));
```

-> DM_UInt operation

Dieser Parameter definiert, welche Aktion für den Grafik-Handler ausgeführt werden soll.

Wertebereich

- *DMF_RegisterHandler*
  Meldet den Handler beim IDM an.
- *DMF_WithdrawHandler*
  Meldet den Handler beim IDM ab.
- *DMF_EnableHandler*
  Aktiviert den Handler.

» *DMF_DisableHandler*
Deaktiviert den Handler.

Nach seiner Anmeldung ist ein Grafik-Handler automatisch aktiv. Es können mehrere Grafik-Handler angemeldet werden, aber jeder Handler höchstens einmal.

-> DM_UInt options

Zurzeit unbenutzt; muss *0* sein.

Rückgabewert

DM_TRUE	Die Aktion konnte erfolgreich ausgeführt werden.
DM_FALSE	Im Fehlerfall.

Anmerkungen

Die angemeldeten Grafik-Handler werden in der umgekehrten Reihenfolge aufgerufen, in der sie angemeldet wurden.

Da Grafik-Handler beim Laden von Bildern Speicher allokieren, darf ein Handler nicht deaktiviert oder abgemeldet werden, bevor der von ihm reservierte Speicher nicht wieder von ihm oder einem anderen Grafik-Handler freigegeben wurde. Wenn alle Grafik-Handler ihren Speicher auf die selbe Art und Weise allokieren, kann ein Grafik-Handler den von anderen Handlern allokierten Speicher freigeben. Dann gibt der erste aufgerufene Grafik-Handler den Speicher frei, unabhängig davon, welcher Handler ihn zuvor allokiert hat. In diesem Fall genügt es, wenn ein Grafik-Handler angemeldet und aktiv bleibt, um den von Grafik-Handlern allokierten Speicher wieder freizugeben.

Beim Beenden einer Anwendung ist das Abmelden von Grafik-Handlern nicht notwendig.

Siehe auch

Funktion DM_PictureHandler

3.52 DM_ProposeInputHandlerArgs

Mit Hilfe dieser Funktion kann vom Dialog Manager eine nicht belegte Nachricht erfragt werden. Diese ist dazu notwendig, dass beim Vergeben einer Nachrichtennummer keine Überschneidungen mit bereits anderen vergebenen Nachrichtennummern auftreten können.

Diese Funktion gibt es nur unter MICROSOFT WINDOWS.

```
DM_Boolean DML_default DM_EXPORT DM_ProposeInputHandlerArgs
(
  DM_InputHandlerArgs pInputArgs,
  DM_Options options
)
```

Parameter

<-> DM_InputHandlerArgs pInputArgs

Das Element msg oder message in diesem Parameter ist beim Aufruf dieser Funktion mit der Nummer der Nachricht belegt. Dieser Wert wird vom Dialog Manager als Wunsch interpretiert. Bei der Rückgabe enthält dieses Element die wirklich angelegte Nachrichtennummer. Es muss sich dabei um eine vom Benutzer definierte Nachricht handeln, liegt also im Bereich von WM_USER bis 0x7FFF.

Bei der Rückgabe dieser Funktion ist zusätzlich das Element hwnd belegt. In diesem Element wird das Objekt zurückgegeben, an das die Funktion gebunden worden ist.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

TRUE	Funktion wurde erfolgreich ausgeführt.
FALSE	Funktion wurde nicht erfolgreich ausgeführt da entweder die angegebene Nachricht im falschen Bereich war oder ein *NULL*-Pointer für den ersten Parameter angegeben worden ist.

Beispiel

Suche nach einer freien Message

```
static boolean TcpWin_CheckAvail __1(
(TranspDescr *, tpdesc))
{
  DM_InputHandlerArgs InpArgs;

  /* liefert WinHandle des unsichtbaren Fensters, an dem
  ** Input-Handler haengt und gibt freie Message zurueck.
  */
  InpArgs.hwnd = (HWND) 0;
```

```
  InpArgs.message = TcpWinMsgGetXByY;
  InpArgs.wParam = (WPARAM) 0;
  InpArgs.lParam = (LPARAM) 0;
  InpArgs.mresult = (LRESULT) 0;
  InpArgs.userdata = (FPTR) 0;

  DM_ProposeInputHandlerArgs (&InpArgs, DMF_DontTrace);
  TcpWinHwnd = InpArgs.hwnd;
  TcpWinMsgGetXByY = InpArgs.message;

}
```

3.53 DM_QueryBox

Mit Hilfe dieser Funktion kann eine entsprechend angegebene Messagebox geöffnet werden. Das Programm wartet bis der Benutzer diese Messagebox wieder geschlossen hat.

```
DM_Enum DML_default DM_EXPORT DM_QueryBox
(
  DM_ID objectID,
  DM_ID parentID,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter spezifiziert die Messagebox , die geöffnet werden soll.

-> DM_ID parentID

Dieser Parameter spezifiziert das Fenster, in dem die Messagebox erscheinen soll. Der Parameter kann ignoriert werden. Falls dieser Parameter angegeben ist, muss ein Fenster oder *NULL* angegeben werden.

Falls das Fenstersystem es erlaubt, wird die Messagebox über dem Vater-Fenster zentriert dargestellt. Andernfalls wird die Position vom Fenstersystem selbst festgelegt (z.B. Bildschirmmitte).

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

Nummer des gedrückten Buttons. Dafür gibt es folgende Definitionen:

- *MB_abort*
- *MB_cancel*
- *MB_ignore*
- *MB_no*
- *MB_ok*
- *MB_retry*
- *MB_yes*

Hinweis zum IDM für Motif

Bitte beachten Sie in Bezug auf die Anordnung eines Dialogfelds vor oder hinter anderen Fenstern und Dialogfeldern auf dem Bildschirm (Z-Ordnung) den Hinweis im Kapitel „Z-Ordnung von Fenstern und Dialogfeldern“ der „Objektreferenz“.

Beispiel

Von einer C-Funktion aus soll eine Messagebox geöffnet werden. Diese C-Funktion hat dann in etwa folgendes Aussehen:

```
DM_Boolean DML_default DM_ENTRY MESSBOX __1((DM_ID id))
{
    DM_Value data;
    /* Setzen des anzuzeigenden Textes */
    data.type = DT_string;
    data.value.string = "HALLO TEST";
    DM_SetValue(id , AT_text , 0 , &data , DMF_ShipEvent );

    /* Setzen des Titels der Messagebox */
    data.type = DT_string;
    data.value.string = "Testausgabe";
    DM_SetValue(id , AT_title , 0 , &data , DMF_ShipEvent );
    /*
    ** Öffnen der Messagebox und Return von TRUE, falls
    ** OK gedrückt wird
    */
    return ((DM_QueryBox(id,0,0)== MB_ok) ? TRUE : FALSE);
}
```

Siehe auch

C-Funktion DM_OpenBox

Objekt Messagebox

Eingebaute Funktion querybox im Handbuch „Regelsprache“

3.54 DM_QueryError

Mit Hilfe dieser Funktion kann die Anwendung den Fehlercode abfragen, den der letzte DM-Aufruf erzeugt hat. Der Dialog Manager gibt die Anzahl der Fehler und die Fehler zurück.

```
DM_UInt DML_default DM_EXPORT DM_QueryError
(
  DM_ErrorCode buffp[ ],
  DM_UInt bufflen,
  DM_Options options
)
```

Parameter

<-> DM_ErrorCode buffp[]

Dies ist ein Array von Fehlercodes. Es wird durch den Dialog Manager belegt, muss aber innerhalb der Anwendung allokiert sein. Wenn das Array nicht groß genug ist, entfallen die letzten Fehler. Um sicherzugehen, dass alle Fehlercodes, die im Dialog Manager gespeichert sind, an die Anwendung übergeben werden können, sollte das Feld die Größe von 32 haben.

-> DM_UInt bufflen

Dies ist die Länge eines Fehlercode-Arrays, der dem Dialog Manager übergeben wurde.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

Die Anzahl der gültigen Fehler im Array.

Beispiel

Zentrale Routine für Fehlerbehandlung

```
static void QueryError()
{
    DM_ErrorCode errorbuffer[32];
    /* buffer für die aufgetretenen Fehler */
    register int i;
    int errors;    /* Anzahl der Fehler */

    if ((errors = DM_QueryError(errorbuffer, 32, 0)))
    for (i = 0; i < errors; i++)
        DM_TraceMessage(DM_ErrMsgText(errorbuffer[i], 0), 0);
}
```

3.55 DM_QueueExtEvent

Die Ausführung einer dem externen Ereignis zugeordneten Regel wird durch die Anwendung über die Funktion **DM_QueueExtEvent** vorgemerkt. „Vorgemerkt“ bedeutet, dass die Regel nicht sofort ausgeführt wird, sondern das externe Ereignis in eine Queue eingetragen wird und entsprechend den Mechanismen für Dialogereignisse weiterverarbeitet wird.

Dieses Ereignis wird dann vom IDM über die normale Ereignisverarbeitungslogik abgearbeitet und führt zur Bearbeitung einer Regel im Schema `on <object> extevent <no.>`.

```
DM_Boolean DML_default DM_EXPORT DM_QueueExtEvent
(
  DM_ID       objectID,
  DM_Int4     event_no,
  DM_UInt     argc,
  DM_Value   *argv,
  DM_Options  options
)
```

Parameter

-> DM_ID objectID

Dies ist der Identifikator des Objekts, an das dieses externe Ereignis geschickt werden soll.

-> DM_Int4 event_no

Dieser Parameter ist die Nummer des externen Ereignisses, das ausgelöst werden soll.

-> DM_Int argc

In diesem Parameter wird die Anzahl der Parameter (bis zu 16) übergeben.

-> DM_Value *argv

Mit Hilfe dieses Parameters werden die Argumente (bis zu 16) angegeben, die beim Regelaufruf vom IDM mit übergeben werden sollen. Dieser Vektor muss dabei die in dem Parameter *argc* angegebene Länge haben.

-> DM_Options options

Als Option können bei dieser Funktion folgende Werte angegeben werden:

Option	Bedeutung
DMF_DontTrace	Diese Option bedeutet, dass der Funktionsaufruf nicht mitprotokolliert werden soll, falls die Anwendung mit der Trace-Option gestartet worden ist.

Option	Bedeutung
DMF_Synchronous	Diese Option kann gesetzt werden, wenn sichergestellt ist, dass die Funktion **DM_QueueExtEvent** synchron zum Prozess aufgerufen wird. Dann kann die Funktion intern effizienter arbeiten. Synchron ist z.B. dann nicht der Fall, wenn aus einem Signal-Handler diese Funktion aufgerufen wird.
DMF_NoCriticalSection	Diese Option verhindert unter MICROSOFT WINDOWS die Benutzung einer „Critical Section“.

Rückgabewert

DM_TRUE Externes Ereignis konnte in die Queue gestellt werden.

DM_FALSE Externes Ereignis konnte nicht in die Queue gestellt werden.

Anmerkung zu Microsoft Windows

Die Funktion **DM_QueueExtEvent** verwendet ab IDM-Version A.05.01.a eine „Critical Section“ um sicherzustellen, dass sie vollständig abgearbeitet ist, bevor diese Funktion oder die Funktion **DM_SendEvent** das nächste Mal aufgerufen wird. Wird eine der beiden Funktionen in einer Situation aufgerufen, in der eine „Critical Section“ nicht zulässig ist, dann kann die Verwendung der „Critical Section“ durch die Option *DMF_NoCriticalSection* unterbunden werden.

Achtung

Ein Thread, der eine der beiden Funktionen ausführt darf nicht abgebrochen werden.

Beispiel

In einem Dialog soll auf ein Signal des Betriebssystems reagiert werden. Der Dialog sieht dann etwa wie folgt aus:

```
on dialog extevent 4711 (integer ErrCode)
{
  variable string S;
  S := "Dialog hat extevent " + itoa(ErrCode) + " erhalten";
  print S;
}
```

Das zugehörige C-Programm sieht etwa wie folgt aus:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <IDMuser.h>

DM_ID dialogID;
/* Diese Funktion wurde als Signal-Handler installiert */
```

```
void handler __1((int, sig))
{
  DM_Value data;

  data.type = DT_integer;
  data.value.integer = sig;
  DM_QueueExtEvent(dialogID, 4711, 1, &data, 0);
}
```

Siehe auch

C-Funktionen **DM_SendEvent**, **DM_SendMethod**

Kapitel „Externe Ereignisse“ und eingebaute Funktion sendevent() im Handbuch „Regelsprache“

3.56 DM_Realloc

Mit Hilfe dieser Funktion kann ein bereits allokierter Speicherbereich vergrößert oder verkleinert werden. Dieser zu verändernde Speicherbereich muss mit DM_Malloc allokiert worden sein.

```
DM_Pointer DML_default DM_EXPORT DM_Realloc
(
  DM_Pointer ptr,
  DM_UInt4 size
)
```

Parameter

-> DM_Pointer ptr

Dieser Parameter ist der Zeiger auf den bereits allokierten Speicherbereich, der in seiner Größe verändert werden soll.

-> DM_UInt4 size

In diesem Parameter wird die neue Größe des Speicherbereichs angegeben.

Warnung

Auf MS-Windows darf diese Größe nicht größer als 64 KByte sein!

Rückgabewert

Pointer auf den neu allokierten Speicherbereich. Konnte der Speicher nicht allokiert werden, wird der *NULL*-Pointer zurückgegeben.

Beispiel

Ein String soll in einen bereits allokierten Speicherbereich kopiert werden.

```
char * string;
if ((string = DM_Realloc(string, strlen("12345")+1))
    strcpy(string, "12345");
```

3.57 DM_ResetMultiValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, mehrere Attribute von verschiedenen DM-Objekten in einem Funktionsaufruf auf ihren Modellwert zurückzusetzen. Diese Funktion sollte daher v.a. im Zusammenhang mit dem verteilten DM eingesetzt werden, da sie die Netzwerkbelastung deutlich reduziert.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_ResetMultiValue
(
  DM_MultiValue *values,
  DM_UInt count,
  DM_ID dialogID,
  DM_String pathname,
  DM_Options options
)
```

Parameter

<-> DM_MultiValue *values

In diesem Parameter wird eine Liste von Attributen und Objekten übergeben, die zurückgesetzt werden sollen. Ist dabei das Element in der Struktur für das Objekt auf 0 gesetzt, wird das im Parameter *pathname* beschriebene Objekt genommen. Die Liste muss dabei mindestens die in Parameter *count* angegebene Länge haben.

-> DM_UInt count

In diesem Parameter wird die Länge des im Parameter *values* angegebenen Objekt-Attribut-Vektors übergeben.

-> DM_ID dialogID

Dieser Parameter beschreibt den Dialog, zu dem die angegebenen Objekte gehören. Muss nur dann angegeben werden, wenn die ID des Objektes dessen Attribute zurückgesetzt werden sollen, nicht bekannt ist und daher der Name des Objektes im Parameter *pathname* angegeben ist.

-> DM_String pathname

Dieser Parameter bezeichnet das Objekt, dessen Attribute zurückgesetzt werden soll. Ist nur belegt, wenn interne ID von Objekt noch nicht bekannt ist.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

TRUE	Attribute konnten erfolgreich zurückgesetzt werden.
FALSE	Mindestens ein Attribut konnte nicht zurückgesetzt werden.

Beispiel

Zurücksetzen von Koordinaten bei mehreren Objekten

```
void DML_default DM_ENTRY Reset  __3((DM_ID, o1),
                     (DM_ID, o2),
                     (DM_ID, o3))
{
    DM_MultiValue val[3];

/* Setzen der jeweiligen Objekt-ID */
    val[0].object = o1;
    val[1].object = o2;
    val[2].object = o3;

/* Setzen des jeweiligen Index-Types */
    val[0].index.type = DT_void;
    val[1].index.type = DT_void;
    val[2].index.type = DT_void;

/* Setzen des jeweiligen Attributes */
    val[0].attribute = AT_xleft;
    val[1].attribute = AT_width;
    val[2].attribute = AT_xright;

    DM_ResetMultiValue(val, 3, dialogID, (char *)0, 0);
}
```

3.58 DM_ResetValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten auf den Wert des zugehörigen Modells oder Defaults zurückzusetzen.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_ResetValue
(
  DM_ID objectID;
  DM_Attribute attr;
  DM_UInt index;
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie zurücksetzen möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Attribut, das Sie von dem Objekt zurücksetzen möchten. Alle zugelassenen Attribute sind in der Datei **IDMuser.h** definiert.

-> DM_UInt index

Dieser Parameter wird nur bei Vektorattributen von Objekten ausgewertet und beschreibt den Index des gesuchten Unterobjekts (z.B. Text in Listbox).

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind, und auf das Ändern des angegebenen Attributes reagieren.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich zurückgesetzt werden.
FALSE	Das Attribut konnte nicht zurückgesetzt werden.

3.59 DM_ResetValueIndex

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute mit zwei Indices auf den Wert des zugehörigen Defaultattributs (mit Index 0) zurückzusetzen.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_ResetValueIndex
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_Value *index,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie ändern möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Objektattribut, das Sie ändern möchten. Alle zugelassenen Attribute sind in der Datei **IDMuser.h** definiert.

-> DM_Value *index

Hier kann der Datentyp des Index (enum, index) und dessen Wert angegeben werden.

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Zurücksetzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind und auf das Ändern des angegebenen Attributes reagieren.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich zurückgesetzt werden.
FALSE	Das Attribut konnte nicht zurückgesetzt werden.

3.60 DM_SaveProfile

Diese Funktion schreibt die aktuellen Werte aller konfigurierbaren ***Record***-Instanzen (*.configurable = true*) und **globalen Variablen** (deklariert mit **config**) eines Dialogs oder Moduls in eine Konfigurationsdatei (profile), aus der sie mit der Funktion **DM_LoadProfile()** wieder geladen werden können.

Bei ***Records*** werden standardmäßig nur Werte, die nicht geerbt sind, in die Datei geschrieben. Um auch die geerbten Werte in die Datei zu schreiben ist der Parameter *options* auf *DMF_SaveAll* zu setzen.

Es werden nur Werte aus dem angegebenen Dialog oder Modul gespeichert. Aus anderen Modulen importierte ***Records*** und Variablen werden nicht berücksichtigt.

```
DM_Boolean DML_default DM_EXPORT DM_SaveProfile
(
  DM_String  filename,
  DM_ID      dialog,
  DM_String  comment,
  DM_Options options
)
```

Parameter

-> DM_String filename

Dieser Parameter definiert den Dateinamen der Konfigurationsdatei. Es kann ein Dateipfad angegeben werden, der auch eine Umgebungsvariable enthalten darf.

-> DM_ID dialog

Dieser Parameter enthält den Identifikator des Dialogs oder Moduls, dessen ***Record***- und Variablenwerte in die Datei geschrieben werden sollen.

-> DM_String comment

In diesem Parameter kann ein Text angegeben werden, der als Kommentar in die Konfigurationsdatei geschrieben wird.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_SaveAll	Schreibt zusätzlich die geerbten Werte in die Konfigurationsdatei.

Rückgabewert

DM_TRUE Das Speichern der Werte in der Konfigurationsdatei war erfolgreich.

DM_FALSE	Die Werte konnten nicht gespeichert werden. Dies kann an Fehlern beim Zugriff auf die Datei oder einer ungültigen Modul-ID liegen.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

C-Funktion DM_LoadProfile

Eingebaute Funktion saveprofile

3.61 DM_SendEvent

Die Ausführung einer dem externen Ereignis zugeordneten Regel wird durch die Anwendung über die Funktion **DM_SendEvent** vorgemerkt. „Vorgemerkt“ bedeutet, dass die Regel nicht sofort ausgeführt wird, sondern das externe Ereignis in eine Queue eingetragen wird und entsprechend den Mechanismen für Dialogereignisse weiterverarbeitet wird.

Dieses Ereignis wird dann vom IDM über die normale Ereignisverarbeitungslogik abgearbeitet und führt zur Bearbeitung einer Regel im Schema `on <object> extevent <no.>`.

```
DM_Boolean DML_default DM_EXPORT DM_SendEvent
(
  DM_ID       objectID,
  DM_Value   *eventData,
  DM_UInt     argc,
  DM_Value   *argv,
  DM_Options  options
)
```

Parameter

-> DM_ID objectID

Dies ist der Identifikator des Objekts, an das dieses externe Ereignis geschickt werden soll.

-> DM_Value *eventData

Dieser Parameter ist das externe Ereignis, das ausgelöst werden soll.

-> DM_Int argc

In diesem Parameter wird die Anzahl der Parameter (bis zu 16) übergeben.

-> DM_Value *argv

Mit Hilfe dieses Parameters werden die Argumente (bis zu 16) angegeben, die beim Regelaufruf vom IDM mit übergeben werden sollen. Dieser Vektor muss dabei die in dem Parameter *argc* angegebene Länge haben.

-> DM_Options options

Als Option können bei dieser Funktion folgende Werte angegeben werden:

Option	Bedeutung
DMF_DontTrace	Diese Option bedeutet, dass der Funktionsaufruf nicht mitprotokolliert werden soll, falls die Anwendung mit der Trace-Option gestartet worden ist.

Option	Bedeutung
DMF_Synchronous	Diese Option kann gesetzt werden, wenn sichergestellt ist, dass die Funktion **DM_SendEvent** synchron zum Prozess aufgerufen wird. Dann kann die Funktion intern effizienter arbeiten. Synchron ist z.B. dann nicht der Fall, wenn aus einem Signal-Handler diese Funktion aufgerufen wird.
DMF_NoCriticalSection	Diese Option verhindert unter MICROSOFT WINDOWS die Benutzung einer „Critical Section“.

Rückgabewert

DM_TRUE Externes Ereignis konnte in die Queue gestellt werden.

DM_FALSE Externes Ereignis konnte nicht in die Queue gestellt werden.

Anmerkung zu Microsoft Windows

Die Funktion **DM_SendEvent** verwendet ab IDM-Version A.05.01.a eine „Critical Section“ um sicherzustellen, dass sie vollständig abgearbeitet ist, bevor diese Funktion oder die Funktion **DM_QueueExtEvent** das nächste Mal aufgerufen wird. Wird eine der beiden Funktionen in einer Situation aufgerufen, in der eine „Critical Section“ nicht zulässig ist, dann kann die Verwendung der „Critical Section“ durch die Option *DMF_NoCriticalSection* unterbunden werden.

Achtung

Ein Thread, der eine der beiden Funktionen ausführt darf nicht abgebrochen werden.

Siehe auch

C-Funktionen **DM_QueueExtEvent**, **DM_SendMethod**

Kapitel „Externe Ereignisse“ und eingebaute Funktion sendevent() im Handbuch „Regelsprache“

Ressource message

3.62 DM_SendMethod

Mit dieser Funktion wird ein Methodenaufruf in die Ereignis-Warteschlange gestellt und asynchron aus der Ereignisschleife (DM_EventLoop) heraus ausgeführt. Die Funktion ist damit eine einfachere Alternative zum Verschicken eines externen Ereignisses mit **DM_SendEvent()** und Aufrufen der Methode in der Ereignisregel für dieses externe Ereignis.

DM_SendMethod() unterstützt maximal *14* Argumente für den Methodenaufruf und kann nicht für Methoden mit `output`-Parametern verwendet werden.

Rückgabewerte von Methoden können nicht verarbeitet werden.

Definition

```
DM_Boolean DML_default DM_EXPORT DM_SendMethod
(
  DM_ID      object,
  DM_Method  method,
  DM_UInt    argc,
  DM_Value  *argv,
  DM_Options options
)
```

Parameter

-> DM_ID *object*

Objekt dessen Methode asynchron aufgerufen werden soll.

-> DM_Method *Method*

Identifikator der aufzurufenden Methode.

-> DM_Int argc

In diesem Parameter wird die Anzahl der Argumente für den Methodenaufruf (bis zu *14*) übergeben.

-> DM_Value *argv

Mit Hilfe dieses Parameters werden die Argumente (bis zu *14*) angegeben, die beim Methodenaufruf vom IDM mit übergeben werden sollen. Dieser Vektor muss dabei die in dem Parameter *argc* angegebene Länge haben.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Derzeit bitte nur *0* angeben.

Rückgabewert

DM_TRUE	Methodenaufruf wurde in die Ereignis-Warteschlange gestellt.
DM_FALSE	Methodenaufruf konnte nicht in die Ereignis-Warteschlange gestellt werden.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

C-Funktion DM_SendEvent

Eingebaute Funktion sendmethod

3.63 DM_SetContent

Mit Hilfe dieser Funktion kann von der Anwendung aus der Inhalt eines Objekts auf einmal gesetzt werden. Dieses geht dann wesentlich schneller als das Setzen des Objektinhalts über DM_SetValue und das Attribut AT_content. Diese Funktion kann beim Tablefield, Treeview-Objekt, Poptext und Listbox eingesetzt werden.

```
DM_Boolean DML_default DM_EXPORT DM_SetContent
(
  DM_ID objectID,
  DM_Value *firstindex,
  DM_Value *lastindex,
  DM_Content *contentvec,
  DM_UInt count,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter bezeichnet das Objekt, das mit dem neuen Inhalt gefüllt werden soll.

-> DM_Value *firstindex

Über diesen Parameter wird gesteuert, welcher Bereich des Inhalts durch diese Funktion modifiziert werden soll. Dabei wird in diesem Parameter der Startpunkt des Bereichs definiert. Bei einer Listbox oder einem Poptext muss der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Startwert belegt werden. Bei einem Tablefield muss der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Startwert belegt werden. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Startpunkt folgende Defaults:

poptext	integer = 1
listbox	integer = 1
tablefield	index.first = 1, index.second = 1
treeview	integer = 1

-> DM_Value *lastindex

Steuert, welcher Bereich des Inhalts durch diese Funktion modifiziert werden soll. Dabei wird in diesem Parameter der Endpunkt des Bereichs definiert. Bei einer Listbox oder einem Poptext muss der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Endewert belegt werden. Bei einem Tablefield muss der Typ in der DM_Value-Struktur

auf DT_index gesetzt und der Index-Wert in der Union mit dem Endewert belegt werden. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, wird der Endpunkt durch die Größe des neuen Inhalts definiert. Der Inhalt der Objekte wird hinter dem letzten modifizierten Eintrag abgeschnitten.

poptext	integer = Object.itemcount
listbox	integer = Object.itemcount
tablefield	index.first = object.rowcount, index.second = object.colcount
treeview	Integer = Object.itemcount

-> DM_Content *contentvec

In diesem Array wird der neue Inhalt des Objekts übergeben. Die hier enthaltene Information kann nach dem erfolgreichen Aufruf der Funktion DM_SetContent in der Anwendung wieder gelöscht werden, da sich der DM die Information kopiert.

Beim Objekt Tablefield wird der Inhalt als Liste angegeben und entsprechend der Indexangaben gelesen. Das Attribut .direction bestimmt hierbei, ob das durch firstindex bzw. lastindex angegebene Rechteck zeilen- oder spaltenweise gefüllt werden soll.

Bei .direction = 1 wird das Rechteck zeilenweise gefüllt.

Bei .direction = 2 wird das Rechteck spaltenweise gefüllt.

-> DM_UInt count

Dieser Parameter gibt die Anzahl der Elemente an, die mit diesem Aufruf gesetzt werden soll.

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind und auf das Ändern des angegebenen Attributes reagieren.

Option	Bedeutung
DMF_UseUserData	Diese Option besagt, dass beim Auswerten des Attributvektors das Attribut *.userdata* beachtet werden soll. Ist die Option *DMF_UseUserData* gesetzt, kopiert der DM die Userdata für jeden Objekteintrag. Wird die Option nicht gesetzt, wird die Userdata ignoriert.
DMF_OmitActive	Diese Option besagt, dass beim Auswerten des Attributvektors das Attribut *.active* nicht beachtet werden soll. Ist die Option *DMF_OmitActive* gesetzt, ignoriert der DM den Aktivierungszustand für jeden Objekteintrag. Wird die Option nicht gesetzt, wird der Aktivierungszustand der Einträge ganz normal übernommen.
DMF_OmitStrings	Diese Option besagt, dass beim Auswerten des Attributvektors das Attribut *.content* nicht beachtet werden soll. Ist die Option *DMF_OmitStrings* gesetzt, ignoriert der DM die Strings für jeden Objekteintrag. Wird die Option nicht gesetzt, werden die Inhalte der Einträge ganz normal übernommen.
DMF_OmitSensitive	Diese Option besagt, dass beim Auswerten des Attributvektors das Attribut *.sensitive* nicht beachtet werden soll. Ist die Option *DMF_OmitSensitive* gesetzt, ignoriert der DM die Selektierbarkeit für jeden Objekteintrag. Wird die Option nicht gesetzt, wird die Selektierbarkeit der Einträge ganz normal übernommen.

Rückgabewert

TRUE	Das Füllen des Objekts wurde erfolgreich ausgeführt.
FALSE	Das Objekt konnte nicht gefüllt werden.

Beispiel

Füllen eines Tablefields

```
static DM_Content *content;
static ushort ColCount;

void DML_default DM_ENTRY ContInit__2(
  (DM_ID, table)
  (long, fillRows))
{
  DM_Value data;
  ushort rowcount;
  ushort rowheader;
  ushort count;
  DM_Value first, last;
```

```
  ushort i;

  DM_GetValue(table, AT_rowcount, 0, &data, 0);
  rowcount = data.value.integer;

  DM_GetValue(table, AT_colcount, 0, &data, 0);
  colcount = data.value.integer;

  DM_GetValue(table, AT_rowheader, 0, &data, 0);
  rowheader = data.value.integer;

  count = (rowcount-rowheader) * ColCount;

  content = (DM_Content*)DM_Malloc(count*sizeof(DM_Content));

  for (i=0; i<count; i++)
  {
    char buf[10];

    sprintf(buf, "<%8d>", i);

    content[i].string = DM_Strdup(buf);
    content[i].active = FALSE;
    content[i].sensitive = TRUE;
  }

  if (fillRows > (rowcount-rowheader))
    fillRows = rowcount-rowheader;

  if (fillRows)
  {
    first.type = DT_index;
    first.value.index.first = rowheader + 1;
    first.value.index.second =  1;

    last.type = DT_index;
    last.value.index.first = fillRows + rowheader;
    first.value.index.second =  ColCount;

    DM_SetContent(table, &first, &last, content,
                  fillRows*ColCount,0);
  }
}
```

Siehe auch

Objekte listbox, poptext, tablefield, treeview

3.64 DM_SetMultiValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, mehrere Attribute von verschiedenen DM-Objekten in einem Funktionsaufruf zu setzen. Diese Funktion sollte daher v.a. im Zusammenhang mit dem verteilten DM eingesetzt werden, da sie Netzwerkbelastung deutlich reduziert.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_SetMultiValue
(
  DM_MultiValue *values,
  DM_UInt count,
  DM_ID dialogID,
  DM_String pathname,
  DM_Options options
)
```

Parameter

<-> DM_MultiValue *values

In diesem Parameter wird eine Liste von Attributen und Objekten übergeben, die gesetzt werden sollen. Ist dabei das Element in der Struktur für das Objekt auf 0 gesetzt, wird das im Parameter *pathname* beschriebene Objekt genommen. Die Liste muss dabei mindestens die im Parameter *count* angegebene Länge haben.

-> DM_UInt count

Dieser Parameter bezeichnet die Länge des im Parameter *values* angegebenen Objekt-Attribut-Vektors.

-> DM_ID dialogID

Dieser Parameter beschreibt den Dialog, zu dem die angegebenen Objekte gehören. Muss nur dann angegeben werden, wenn die ID des Objektes dessen Attribute gesetzt werden sollen, nicht bekannt ist und daher der Name des Objektes im Parameter *pathname* angegeben ist.

-> DM_String pathname

Dieser Parameter bezeichnet das Objekt, dessen Attribute gesetzt werden soll. Er ist nur belegt, wenn die interne ID von Objekt noch nicht bekannt ist.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

TRUE	Attribute konnten erfolgreich gesetzt werden.
FALSE	Mindestens ein Attribut konnte nicht gesetzt werden.

Beispiel

Setzen von Koordinaten bei mehreren Objekten.

```
void DML_default DM_ENTRY Set  __3((DM_ID, o1),
                   (DM_ID, o2),
                   (DM_ID, o3))
{
  DM_MultiValue val[3];
/* Setzen des jeweiligen Objekts */
  val[0].object = o1;
  val[1].object = o2;
  val[2].object = o3;

/* Setzen des jeweiligen Index-Typs */
  val[0].index.type = DT_void;
  val[1].index.type = DT_void;
  val[2].index.type = DT_void;

/* Setzen des jeweiligen Attributes */
  val[0].attribute = AT_xleft;
  val[1].attribute = AT_width;
  val[2].attribute = AT_xright;

/* Setzen des Datentyps des Attributes */
  val[0].data.type = DT_integer;
  val[1].data.type = DT_integer;
  val[2].data.type = DT_integer;

  val[0].data.value.integer = 10;
  val[1].data.value.integer = 50;
  val[2].data.value.integer = 10;

  DM_SetMultiValue(val, 3, dialogID, (char *)0, 0);
}
```

3.65 DM_SetToolkitData

Mit dieser Funktion ist ein direkter Zugang zum Fenstersystem möglich. Dies bedeutet, dass es der Anwendung mit dieser Funktion möglich ist, Attribute zu ändern, die nicht vom ISA Dialog Manager unterstützt werden, aber im Toolkit existieren.

```
DM_Boolean DML_default DM_EXPORT DM_SetToolkitData
(
  DM_ID objectID,
  DM_Attribute attr
  FPTR value,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter ist der Identifikator des Objekts, dessen fenstersystemspezifischen Daten geändert werden sollen.

-> DM_Attribute attr

Mit Hilfe dieses Parameters können Sie definieren, welches Fenstersystem-Attribut geändert werden soll.

-> FPTR value

Mit Hilfe dieses Parameters können neue, fenstersystemspezifische Daten des Objekts gesetzt werden.

-> DM_Options options

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich gesetzt werden.
FALSE	Das Attribut konnte nicht erfolgreich gesetzt werden.

Die Attribute und die damit verbundenen Rückgabewerte sind fenstersystemabhängig und werden in den nachfolgenden Kapiteln erklärt.

3.65.1 Motif

Mit Hilfe dieser Funktionen können die für X-Windows notwendigen Daten geändert werden, wie „window-id“, „widget“ und „color“. Die Bedeutung dieser Datentypen wird in den entsprechenden X-Windows-Handbüchern erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	Bedeutung
AT_CanvasData	Dieser Wert speichert die benutzerspezifischen Daten einer Canvas. Diese Daten werden bei der angegebenen Canvas gemerkt und beinhalten benutzerspezifische Daten. **Siehe auch** Kapitel „Strukturen für Canvas-Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“
AT_XAppClass	Mit Hilfe dieses Attributes kann die Xt-Application-Class gesetzt werden.
AT_XColor	Dieser Wert setzt die X-Windows-spezifische Struktur für die angegebene Farbe. Der Wert des Parameters value der Funktion sollte vom Typ „Pixel“ sein.
AT_XCursor	Dieser Wert setzt die X-Windows-spezifische Struktur für den angegebenen Cursor. Der Wert des Parameters value der Funktion sollte vom Typ „Cursor“ sein.
AT_XFont	Dieser Wert setzt die X-Windows-spezifische Struktur für die angegebenen Schrift (Font). Der Wert des Parameters value der Funktion sollte vom Typ „XFontStruct*“ sein. Die Verfügbarkeit dieses Attributes ist abhängig von der verwendeten Motif-Version.
AT_XFontSet	Dieser Wert setzt die X-Windows-spezifische Struktur für die angegebenen Schrift (Font). Der Wert des Parameters value der Funktion sollte vom Typ „XFontSet“ sein. Die Verfügbarkeit dieses Attributes ist abhängig von der verwendeten Motif-Version.
AT_XmFontList	Dieser Wert setzt die X-Windows-spezifische Struktur für die angegebenen Schrift (Font). Der Wert des Parameters value der Funktion sollte vom Typ „XmFontList“ sein. Die Verfügbarkeit dieses Attributes ist abhängig von der verwendeten Motif-Version.
AT_XtAddEvents	Mit Hilfe dieses Attributs können zusätzliche X-Events für eine Canvas angefordert werden, z.B. Mausbewegungen. Bei Verwendung dieses Attributes muss im Parameter *value* die Ereignismaske ("event_mask") und in *options* die nicht-selektierten Ereignisse übergeben werden („non-maskable“). Sollen keine zusätzlichen Ereignisse mehr an die Canvas geschickt werden, müssen *value* und *options* auf 0 gesetzt werden.

Zu beachten bei Multiscreendialogen

Beim Aufruf mit AT_XTile bzw. AT_XColor kann immer nur das tile bzw. die color des default Screen gesetzt werden.

Siehe auch

Kapitel „Multiscreen Ssupport untder Motif“ im Handbuch „Programmiertechniken“

3.65.2 Microsoft Windows

Mit Hilfe dieser Funktionen können die für Microsoft Windows notwendigen Daten geändert werden. Die Bedeutung dieser Datentypen wird in den entsprechenden Microsoft Windows-Handbüchern erklärt.

Folgende Werte sind für die Attribute zugelassen:

Attribut	Bedeutung
AT_CanvasData	Dieser Wert speichert die benutzerspezifischen Daten einer Canvas. Diese Daten werden bei der angegebenen Canvas gemerkt und beinhalten benutzerspezifische Daten. **Siehe auch** Kapitel „Strukturen für Canvas-Funktionen“ im Handbuch „C-Schnittstelle - Grundlagen“

AT_ClipboardText	Nur am ***Setup***-Objekt zulässig.
	Mit Hilfe des Attributs AT_ClipboardText kann der Inhalt des MS Windows Clipboards gesetzt werden:
	`DM_SetToolkitData(<setup>, AT_ClipboardText, str, 0);`
	Der erhaltene String bleibt bis zum erneuten Aufruf von DM_GetToolkitData bzw. DM_SetToolkitData gültig.
	Um den String ohne Änderung des Clipboards freizugeben dient der Aufruf
	`DM_SetToolkitData(<setup>, AT_ClipboardText, (FPTR) 0, 0);`

AT_WinDisableAll	Dieser Wert macht alle Toplevel-Fenster derselben Anwendung insensitiv - mit Ausnahme des Fensters, dessen DM-ID als *value*-Parameter spezifiziert ist.
AT_WinEnableAll	Dieser Wert macht alle Toplevel-Fenster derselben Anwendung sensitiv - mit Ausnahme des Fensters, dessen DM-ID als *value*-Parameter spezifiziert ist.

Attribut	Bedeutung
AT_XColor	Hiermit lässt sich bei einer Farbressource ein Microsoft-Windows RGB-Wert setzen. Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück.
AT_XTile	Hiermit lässt sich bei einer Farbressource ein Microsoft-Windows Brush setzen. Bei dem Wert muss es sich um einen gültigen Microsoft Windows Brush Handle handeln. Der Wert sollte unbedingt dem Wert von *AT_XColor* entsprechen, da nicht vorherzusagen ist, wann der Dialog Manager den *AT_XColor*-Wert oder den *AT_XTile*-Wert verwendet. Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück. **Achtung** Bei einem **ungültigen Wert** kann der ISA Dialog Manager **abstürzen**.
AT_wsidata	Hiermit lässt sich bei einer Cursor-Ressource ein Microsoft-Windows Cursor setzen. Bei dem Wert muss es sich um einen gültigen Microsoft Windows Cursor Handle handeln. Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück-.The value *0* resets to the original resource value. Er gibt die originale Ressource frei, falls sie vom Dialog Manager nicht mehr benötigt wird. **Achtung** Bei einem **ungültigen Wert** kann der ISA Dialog Manager **abstürzen**.
AT_wsidata	Hiermit lässt sich bei einer Font-Ressource ein Microsoft-Windows Font setzen. Bei dem Wert muss es sich um einen gültigen Microsoft Windows Font Handle handeln. Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück. **Achtung** Bei einem **ungültigen Wert** kann der ISA Dialog Manager **abstürzen**.

Attribut	Bedeutung
AT_Tile / AT_XTile / AT_wsi-data	Hiermit lässt sich bei einer Tile-Ressource ein eigenes Bild setzen. Abhängig von der gesetzten Option müssen folgende Datentypen angegeben werden: - DMF_TlkDataIsIcon: Microsoft Windows Icon Handle - DMF_TlkDataIsWMF: Microsoft Windows Metafile Handle - DMF_TlkDataIsEMF: Microsoft Windows Enhanced Metafile Handle - DMF_TlkDataIsD2D1Bmp: Microsoft Direct 2D Bitmap (ID2D1Bit-map *) - DMF_TlkDataIsD2D1SVG: Microsoft Direct 2D SVG Documnet (ID2D1SvgDocument *) - DMF_TlkDataIsD2D1EMF: Microsoft Direct 2D Metafile (ID2D1G-diMetafile *) - sonst : Microsoft-Windows Bitmap Handle Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück. **Achtung** Bei einem **ungültigen Wert** kann der ISA Dialog Manager **abstür-zen**. **Hinweis**: Wird ein HANDLE gesetzt, dann wird dieser in der Regel intern zu einem Microsoft Direct2D-Objekt gewandelt. Eine Abfrage liefert den gesetzten HANDLE zurück und nicht den gewandelten Wert.
AT_XColor	Hiermit lässt sich bei einer Tile-Ressource ein Microsoft-Windows Farbpalette setzen, mit der eine Bitmap (nicht Icon) gezeichnet wird. Bei dem Wert muss es sich um einen gültigen Microsoft Windows Palette Handle handeln. Der Wert *0* setzt wieder auf den originalen Ressourcenwert zurück. **Achtung** Bei einem **ungültigen Wert** kann der ISA Dialog Manager **abstür-zen**.

3.66 DM_SetValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten zu verändern.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_SetValue
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_UInt index,
  DM_Value *data,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie ändern möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Objektattribut, das Sie ändern möchten. Alle zugelassenen Attribute sind in der Datei **IDMuser.h** definiert.

-> DM_UInt index

Dieser Parameter wird nur bei Vektorattributen von Objekten ausgewertet und beschreibt den Index des gesuchten Unterobjekts (z.B. Text in Listbox).

-> DM_Value*data

In diesem Parameter wird der Wert übergeben, den das Attribut annehmen soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Union belegen. Den Datentyp eines jeden Attributes entnehmen Sie bitte der „Attributreferenz“.

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.

Option	Bedeutung
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind, und auf das Ändern des angegebenen Attributes reagieren.
DMF_AcceptChild	Diese Option ist gültig bei Attribut *AT_options* (Objekt Canvas) unter Motif, denn damit wird eine Canvas bezeichnet, die Kinder haben kann.
DMF_NoFocusFrame	Diese Option ist gültig bei Attribut *AT_options* (Objekt Canvas) unter Motif, denn damit wird eine Canvas bezeichnet, die keinen Fokusrahmen haben soll.
DMF_XlateString	Diese Option besagt, dass der angegebene String, bevor er dem Objekt zugewiesen wird, in die aktuell eingestellte Sprache übersetzt werden soll. Dies geht nur, wenn der Text bereits intern vorhanden ist und eine Übersetzung dafür vorliegt.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich gesetzt werden.
FALSE	Das Attribut konnte nicht gesetzt werden.

Beispiel

Anwendungs-Callback-Funktion, die überprüft, ob ein eingegebener String einer existierenden Datei entspricht.

```
DM_Boolean DML_default DM_CALLBACK CheckFilename __1(
(DM_CallBackArgs *, data))
{
  DM_Value value;    /* structure for DM_SetValue */
  FILE *fptr;    /* file pointer */
  DM_ID id;    /* Identifier of object */


  /* get the current content */
  if (DM_GetValue(data->object, AT_content, 0, &value,
        DMF_GetLocalString))
    /* check the datatype */
    if(value.type == DT_string)
    {
      /* try to open the file */
      if(!((fptr = fopen(value.value.string, "r"))))
      {
```

```
        /*
         * the file cannot be opened for reading.
         * activate the edittext again
         */
        value.type = DT_boolean;
        value.value.boolean = TRUE;
        DM_SetValue(data->object, AT_active, 0, &value,
            DMF_Inhibit);

        /*
         * The file cannot be read. So don't continue
         * processing with the rules
         */
        return (FALSE);
      }
      else
        fclose(fptr);

      /*
      * the file could be opened. enable the other objects
      */
      value.type = DT_boolean;
      value.value.boolean = TRUE;

      /* Get the id of the edittext "Actives" */
      if ((id = DM_PathToID(0, "Actives")))
        /* Change the object to sensitive */
        DM_SetValue(id, AT_sensitive, 0, &value,
            DMF_ShipEvent);
      /*
       * the last parameter must be ShipEvent,
       * because a rule should be triggered
       */

      /*
       * Everything is ok. So let the rule process normally
       */
      return(TRUE);
    }

  /* Too many errors don't continue the rule processing */
  return (FALSE);
}
```

Siehe auch

Eingebaute Funktion setvalue im Handbuch „Regelsprache“

3.67 DM_SetValueIndex

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute vom DM-Objekt Tablefield zu verändern. Diese Funktion ist in der Lage, mit zwei Indices zu arbeiten.

Die dabei zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte der „Objektreferenz“.

```
DM_Boolean DML_default DM_EXPORT DM_SetValueIndex
(
  DM_ID objectID,
  DM_Attribute attr,
  DM_Value *index,
  DM_Value *data,
  DM_Options options;
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie ändern möchten.

-> DM_Attribute attr

Dieser Parameter beschreibt das Objektattribut, das Sie ändern möchten. Alle zugelassenen Attribute sind in der Datei **IDMuser.h** definiert.

-> DM_Value *index

Hier kann der Datentyp des Index (enum, index) und dessen Wert angegeben werden.

-> DM_Value *data

In diesem Parameter wird der Wert übergeben, den das Attribut annehmen soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Union belegen. Den Datentyp eines jeden Attributes entnehmen Sie bitte der „Attributreferenz“.

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.

Option	Bedeutung
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind und auf das Ändern des angegebenen Attributes reagieren.
DMF_XlateString	Diese Option besagt, dass der angegebene String, bevor er dem Objekt zugewiesen wird, in die aktuell eingestellte Sprache übersetzt werden soll. Dies geht nur, wenn der Text bereits intern vorhanden ist und eine Übersetzung dafür vorliegt.

Rückgabewert

TRUE	Das Attribut konnte erfolgreich gesetzt werden.
FALSE	Das Attribut konnte nicht gesetzt werden.

Beispiel

Ein Tablefield soll zeilenweise über die Funktion DM_SetValueIndex gefüllt werden.

```
void DML_default DM_ENTRY SetTable __3 ((DM_ID,      tableID),
                                        (DM_Integer, Rows),
                                        (DM_Integer, Cols))
{
  DM_Value index, data;
  int      row, column;

  index.type = DT_index;
  data.type = DT_string;
  data.value.string = "Neuer String";
  for (row = 1; row <= (int) Rows; row++)
  {
    index.value.index.first = row;
    for (column = 1; column <= (int) Cols; column++)
    {
      index.value.index.second = column;
      DM_SetValueIndex (tableID, AT_content, &index, &data,
          DMF_Inhibit);
    }
  }
}
```

Siehe auch

Eingebaute Funktion setvalue im Handbuch „Regelsprache“

Methode set

3.68 DM_SetVectorValue

Mit Hilfe dieser Funktion können Attribute gesetzt werden, die bei einem Objekt mehrfach vorkommen, sog. "vektorielle Attribute".

```
DM_Boolean DML_default DM_EXPORT DM_SetVectorValue
(
  DM_ID objectID
  DM_Attribute attr,
  DM_Value *firstindex,
  DM_Value *lastindex,
  DM_VectorValue *values,
  DM_Options options
)
```

Parameter

-> DM_ID objectID

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie ändern möchten.

-> DM_Attribute attr

Dieser Parameter bezeichnet das Attribut, das gesetzt werden soll.

-> DM_Value *firstindex

Über diesen Parameter wird gesteuert, welcher Bereich des Inhalts durch diese Funktion modifiziert werden soll. Dabei wird in diesem Parameter der Startpunkt des Bereichs definiert. Dabei muss für ein eindimensionales Attribut der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Startwert belegt werden. Für ein zweidimensionales Attribut muss der Typ in der DM_Value-Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Startwert belegt werden. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, hat der Startpunkt folgende Defaults:

listbox	integer = 1
poptext	integer = 1
tablefield	index.first = 1, index.second = 1
treeview	integer = 1

-> DM_Value *lastindex

Steuert, welcher Bereich des Inhalts durch diese Funktion modifiziert werden soll. In diesem Parameter wird der Endpunkt des Bereichs definiert. Dabei muss für ein eindimensionales Attribut der Typ in der DM_Value-Struktur auf DT_integer gesetzt und der Integer-Wert in der Union mit dem Endewert belegt werden. Für ein zweidimensionales Attribut muss der Typ in der DM_Value-

Struktur auf DT_index gesetzt und der Index-Wert in der Union mit dem Endewert belegt werden. Hierbei wird in index.first die Zeile, in index.second die Spalte eingetragen.

Anmerkung

Wenn dieser Parameter ein *NULL*-Pointer ist, wird der Endpunkt durch die Größe des neuen Vektors definiert. Das Attribut der Objekte wird hinter dem letzten modifizierten Eintrag angeschnitten, z.B.

listbox	integer = Object.itemcount
poptext	integer = Object.itemcount
tablefield	index.first = object.rowcount, index.second = object.colcount
treeview	Integer = Object.itemcount

-> DM_VectorValue *values

Zeiger auf die Werte, die gesetzt werden sollen. Über das Feld *type* in der DM_VectorValue Struktur wird gesteuert, welchen Datentyp die einzelnen Werte haben.

Über das Feld *count* in der DM_VectorValue Struktur wird gesteuert, wie viele Werte in dem Vektor enthalten sind.

-> DM_Options options

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

Option	Bedeutung
DMF_Inhibit	Diese Option bedeutet, dass der Funktionsaufruf keine internen Ereignisse auslösen soll. Wenn dieses Flag gesetzt ist, kann vor allem bei vielen Attribut-Änderungen durch die Anwendung viel Performance gewonnen werden.
DMF_ShipEvent	Diese Option bedeutet, dass der Funktionsaufruf die internen Ereignisse auslösen soll. Dadurch werden Regeln ausgelöst, die für dieses Objekt definiert sind und auf das Ändern des angegebenen Attributes reagieren.
DMF_XlateString	Diese Option besagt, dass der angegebene String, bevor er dem Objekt zugewiesen wird, in die aktuell eingestellte Sprache übersetzt werden soll. Dies geht nur, wenn der Text bereits intern vorhanden ist und eine Übersetzung dafür vorliegt.

Rückgabewert

TRUE	Die Attribute konnten erfolgreich gesetzt werden.

FALSE	Mindestens ein Attribut konnte nicht gesetzt werden.

Beispiel

Einer Listbox soll ein neuer Inhalt gesetzt werden.

```
void DML_default DM_ENTRY SetVector  __1((DM_ID, lb)){  DM_Value first, last;
  DM_VectorValue vec;
  char *list[9]

  list[0] = "V_1";
  list[1] = "V_2";
  list[2] = "V_3";
  list[3] = "V_4";
  list[4] = "V_5";
  list[5] = "V_6";
  list[6] = "V_7";
  list[7] = "V_8";
  list[8] = "V_9";

/*
** Setzen der Start- und Endezeile
** Inhalt der Zeile 1 bis 9 wird durch den neuen Inhalt
** ersetzt, Rest bleibt unverändert
*/
  first.type = DT_integer;
  first.value.integer = 1;
  last.type = DT_integer;
  last.value.integer  = 9;

/* Setzen der Anzahl von Zeilen, die gesetzt werden soll */
  vec.type = DT_string;
  vec.count = 9;
  vec.vector.stringPtr = list;
  DM_SetVectorValue(lb, AT_content, &first, &last, &vec,
    DMF_ShipEvent);

}
```

Siehe auch

Objekte listbox, poptext, tablefield, treeview

3.69 DM_ShutDown

Mit Hilfe dieser Funktion wird der Dialog Manager vollständig beendet. Nach einem Aufruf dieser Funktion ist ein Weiterarbeiten mit dem Dialog Manager unmöglich, da alle globalen Initialisierungsschritte rückgängig gemacht werden.

Diese Funktion wird normalerweise von der Funktion aufgerufen, die das AppMain in der Anwendung aufruft.

Daher darf diese Funktion nur dann aufgerufen werden, wenn das eigentliche Main-Programm im Dialog Manager durch ein eigenes ersetzt wird.

```
void DML_default DM_EXPORT DM_ShutDown
(
  void
)
```

Parameter

Keine.

Rückgabewert

Keiner.

Beispiel

Startprogramm des Dialog Managers, das im Normalfall über die Datei **startup.o** bzw. **startup.obj** dazugelinkt wird.

```
int cdecl main __2(
(int, argc),
(char far * far *, argv))
{
  register int status;
  static char running = 0;

  if ((status = running++) == 0)
  {
    if ((status = DM_BootStrap(&argc, &argv)) == 0)
    {
      DM_InitOptions(&argc, argv, 0);
      DM_TraceMessage ("[AC] Transfer to AppMain(...)",
        DMF_Printf | DMF_InhibitTag);
      status = AppMain (argc, argv);
      DM_TraceMessage ("[AR] AppMain() = %d", DMF_Printf |
        DMF_InhibitTag, status);

      DM_ShutDown();
    }
    else DM_TraceMessage ("Bootstrap failed", DMF_LogFile);
```

```
  }
  else
    DM_FatalAppError ("Unexpected restart", -1, 0);
  return (status);
}
```

3.70 DM_StartDialog

Mit Hilfe dieser Funktion wird die eigentliche Dialoganwendung gestartet. Der DM meldet dabei alle benötigten Ressourcen (Farben, Cursor, Zeichensätze, usw.) beim Fenstersystem an, bringt alle beim Dialog auf sichtbar definierten Toplevel-Objekte auf den Bildschirm und führt die Startregel aus.

```
DM_Boolean DML_default DM_EXPORT DM_StartDialog
(
  DM_ID dialogID,
  DM_Options options
)
```

Parameter

-> DM_ID dialogID

Dieses ist der Identifikator des zu startenden Dialoges. Diesen Identifikator haben Sie von der Funktion DM_LoadDialog als Rückgabewert erhalten.

-> DM_Options options

Dieser Parameter ist für zukünftige Versionen reserviert. Bitte geben Sie deshalb hier eine 0 an.

Rückgabewert

TRUE	Der Dialog konnte erfolgreich gestartet werden.
FALSE	Der Dialog konnte nicht gestartet werden, weil schon ein anderer Dialog läuft oder weil der angegebene Parameter kein Dialog ist.

Beispiel

Typisches Hauptprogramm für Dialog Manager Anwendungen

```
int DML_c DM_CALLBACK AppMain __2(
(int, argc),
(char far * far *, argv))
{
  DM_ID dialogID;

  /*
   * Initialize the Dialog Manager
  */
  if (!DM_Initialize (&argc, argv, 0))
  {
    DM_TraceMessage("could not initialize", DMF_LogFile);
    return (1);
  }

  /*
   * Load the dialog file
```

```
  */

  dialogID = DM_LoadDialog ("tabdemo.dlg",0);
  if (!dialogID)
  {
    DM_TraceMessage("could not load dialog", DMF_LogFile);
    return(1);
  }

  /*
   * Start the dialog and enter event loop
  */

  if (DM_StartDialog (dialogID, 0))
    DM_EventLoop (0);
  else
    return (1);

  return (0);
}
```

Siehe auch

Eingebaute Funktion run im Handbuch „Regelsprache“

3.71 DM_StopDialog

Mit Hilfe der Funktion DM_StopDialog wird ein Dialog beendet. War dies der letzte laufende Dialog, wird die Ereignis-Schleife verlassen.

```
DM_Boolean DML_default DM_EXPORT DM_StopDialog
(
  DM_ID dialogID,
  DM_Options options
)
```

Parameter

-> DM_ID dialogID

Dieses ist der Identifikator des zu stoppenden Dialogs. Diesen Identifikator haben Sie von der Funktion DM_LoadDialog als Rückgabewert erhalten.

-> DM_Options options

In diesem Parameter sind folgende Werte möglich:

Option	Bedeutung
DMF_ForceDestroy	Diese Option bedeutet, dass der angehaltene Dialog gelöscht werden soll. Ist diese Option nicht gesetzt, so bleibt der angehaltene Dialog im Speicher erhalten und kann erneut gestartet werden.

Rückgabewert

TRUE	Der Dialog konnte erfolgreich angehalten werden.
FALSE	Das Dialog konnte nicht angehalten werden.

Siehe auch

Eingebaute Funktion stop im Handbuch „Regelsprache“

3.72 DM_StrCreate

Mit der Funktion **DM_StrCreate** kann ein Text in einer vorgegebenen Kodierung erzeugt werden.

```
DM_String DML_default DM_EXPORT DM_StrCreate
(
  DM_String str,
  DM_UInt1 strCP,
  DM_UInt1 toCP,
  DM_Options options
)
```

Parameter

-> DM_String str

Quelltext mit dem der neue angelegte Text initialisiert wird.

-> DM_UInt1 strCP

Kodierung (Codepage) des Quelltextes.

-> DM_UInt1 toCP

Kodierung (Codepage) des neu angelegten Textes.

-> DM_Options options

Zurzeit unbenutzt, sollte mit 0 angegeben werden.

Parameterwerte

Für die Parameter *strCP* und *toCP* sind neben den bekannten Codepage-Konstanten (CP_ascii, CP_iso8859...) auch folgende Codepage-Konstanten möglich:

Konstante	Bedeutung
CP_appl	aktuell eingestellte Anwendungscodepage
CP_format	aktuell eingestellte Formatcodepage
CP_input	aktuell eingestellte Eingabecodepage
CP_output	aktuell eingestellte Ausgabecodepage
CP_display	aktuell eingestellte Darstellungscodepage
CP_system	aktuell eingestellte Systemcodepage

Diese Konstanten können **nur** mit DM_StrCreate verwendet werden.

Rückgabewert

Der neu angelegte Text.

Der Text muss mit DM_Free freigegeben werden, wenn er nicht mehr gebraucht wird.

Verfügbarkeit

Ab IDM-Version A.05.02.k

3.73 DM_Strdup

Mit Hilfe dieser Funktion können beliebige Strings dupliziert werden. Für die Kopien der Strings werden die Dialog Manager Funktionen für die Speicherverwaltung DM_Malloc benutzt. Daher können diese Strings nur über die Funktion DM_Free freigegeben werden.

```
DM_String DML_default DM_EXPORT DM_Strdup
(
  DM_string string
)
```

Parameter

-> DM_string string

Dieser Parameter ist der String, der dupliziert werden soll.

Rückgabewert

Diese Funktion liefert einen Zeiger auf das Duplikat des Strings zurück oder einen *NULL*-Pointer, wenn der String nicht dupliziert werden konnte.

Beispiel

Ein String, der vom Dialog Manager übergeben worden ist, soll für die Anwendung gesichert werden.

```
  DM_String new_string;

void DML_default DM_ENTRY StoreString __1((DM_String string))
{

  if (new_string = DM_Strdup(string) == (DM_String) 0)
    DM_TraceMessage("String konnte nicht angelegt werden",
        0);

}
```

Wenn dieser String nicht mehr benötigt wird, muss dieser String über die Funktion DM_Free freigegeben werden.

```
void DML_default DM_ENTRY FreeString __0((void))
{

  DM_Free (new_string);

}
```

3.74 DM_StringChange

Diese Funktion kann zur Änderung bzw. Manipulation eines gemanagten Strings verwendet werden.

Neben der reinen Ersetzung des Strings ist über die *DMF_AppendValue*-Option das Konkatenieren von Strings möglich.

Handelt es sich beim *pstring*-Parameter um einen noch ungemanagten String, so wird dieser kopiert und von da an vom IDM gemanagt. Wurde dabei noch die Option *DMF_StaticValue* gesetzt, wird dieser als statischer bzw. globaler String behandelt und nicht wie sonst bei der Rückkehr der Funktion wieder freigegeben.

Grundsätzlich dürfen nur String-Argumente oder lokale bzw. globale Strings gemanagt werden.

```
DM_Boolean DML_default DM_EXPORT DM_StringChange
(
  DM_String * pstring,
  DM_String   newstring,
  DM_Options  options
)
```

Parameter

<-> DM_String * pstring

Dieser Parameter verweist auf die String-Referenz die manipuliert werden soll. Es kann sich dabei um einen schon gemanagten String handeln, muss es aber nicht.

-> DM_String newstring

Dieser Parameter verweist auf den String, welcher der String-Referenz (*pstring*-Parameter) zugewiesen oder an sie angehängt werden soll.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *pstring*-Parameters in eine gemanagte String-Referenz wird diese als statische bzw. globale String-Referenz behandelt.
DMF_AppendValue	Der String im *newstring*-Parameter wird an den String im *pstring*-Parameter angehängt.

Rückgabewert

DM_TRUE	String wurde erfolgreich manipuliert.

DM_FALSE	Ein Fehler ist aufgetreten. Dabei kann es sich um einen falschen „runstate“ bei Aufruf handeln, oder aber eine ungültigen String-Referenz.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue StringOf(integer I);

on dialog start
{
  print StringOf(123);
  print StringOf(-42);
  exit();
}
```

C-Teil

```
...

DM_String DML_default DM_ENTRY StringOf(DM_Integer I)
{
  char      buf[10];
  DM_String data;
  DM_String negative;

  if (I>=0)
  {
    /* return an unmanaged local string */
    sprintf(buf, "%d", (int)I);
    data = buf;
    /* wrong: return data; => buf is a local char array! */
    return DM_StringReturn(&data, 0);
  }
  else
  {
    /* return a managed string */
    DM_StringInit(&negative, 0);  /* can be omitted */
    DM_StringChange(&negative, "!!negative", &data, 0);
    return DM_StringReturn(&negative);
    /* return &negative; => also possible for managed strings! */
  }
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_StringInit, DM_StringReturn

3.75 DM_StringInit

Mit dieser Funktion kann ein String in einen vom IDM gemanagten lokalen oder globalen bzw. statischen String umgewandelt werden. Dadurch sind weitere Manipulationen des Strings durch die Funktion DM_StringChange möglich, wie auch die erleichterte Rückgabe als Parameter oder Rückgabewert. Der IDM übernimmt dann die Verwaltung des Speichers für den String.

Für gemanagte Strings darf nur ein lesender Zugriff auf die String-Zeichen oder den String-Pointer erfolgen. Eine Freigabe über **DM_Free()** darf nicht erfolgen!

Der String wird durch diese Funktion mit *NULL* initialisiert.

Durch Angabe der Option *DMF_StaticValue* wird der String als statisch angesehen und nicht, wie sonst für lokale Strings üblich, nach der Rückkehr der Funktion freigegeben.

Grundsätzlich dürfen nur String-Argumente oder lokale bzw. globale Strings gemanagt werden.

```
DM_Boolean DML_default DM_EXPORT DM_StringInit
(
  DM_String * pstring,
  DM_Options  options
)
```

Parameter

-> DM_String * pstring

Dieser Parameter verweist auf den String-Pointer der zu initialisieren ist und gemanagt werden soll.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Der *pstring*-Parameter wird als statische bzw. globale String-Referenz initialisiert.

Rückgabewert

DM_TRUE	String wurde erfolgreich initialisiert und ist nun gemanagt.
DM_FALSE	Der String konnte nicht gemanagt oder initialisiert werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function string FormatString(integer Op, string String);
```

```
on dialog start
{
  print FormatString (-1, "**");          // set decoration
  print FormatString (0, "hello world");  // print with decoration
  exit();
}
```

C-Teil

```
...

DM_String DML_default DM_ENTRY FormatString (DM_Integer Op, DM_String String)
{
  static DM_String decoration = NULL;
  DM_String newstring;

  if (!decoration)
  {
    /* static initialization */
    DM_StringInit(&decoration, DMF_StaticValue);
    DM_StringChange(&decoration, "-", 0);
    /* above can be done simpler via:
     * DM_StringChange(&decoraton, "-", DMF_StaticValue);
     */
  }

  switch(Op)
  {
  case -1: /* change decoration */
    DM_StringChange(&decoration, String);
    return &decoration;
  case 1:  /* decor only at the beginning */
    DM_StringChange(&newString, decoration, 0);
    DM_StringChange(&newString, String, DMF_AppendValue);
    break;
  case 2:  /* decor only at the end */
    DM_StringChange(&newString, String, 0);
    DM_StringChange(&newString, decoration, DMF_AppendValue);
    break;
  default:
    DM_StringChange(&newString, decoration, 0);
    DM_StringChange(&newString, String, DMF_AppendValue);
    DM_StringChange(&newString, decoration, DMF_AppendValue);
    break;
  }
  return newstring;
  /* also possible: return DM_StringReturn(&newstring, 0); */
```

```
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_StringChange, DM_StringReturn, DM_ValueInit

3.76 DM_StringReturn

Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen Strings (*DM_String*-Werten) an den Aufrufer. Bei Verwendung von lokalen Variablen und Strukturen in einer C-Funktion sind diese nach der Rückgabe ungültig. Mit dieser Funktion kann problemlos und sicher ein lokaler String zurückgegeben werden.

Zu diesem Zweck wird, wenn nötig, eine temporäre Kopie angelegt (z.B. der darin liegende String kopiert). Ein Kopieren findet für gemanagte Wertereferenzen nicht statt. Der zurückgegebene *DM_String* sollte dann mit `return` an den Aufrufer weitergegeben werden.

```
DM_String DML_default DM_EXPORT DM_StringReturn
(
  DM_String * pstring,
  DM_Options  options
)
```

Parameter

-> DM_String * pstring

Dieser Parameter verweist auf den String welcher zurückgegeben wird. Es kann sich dabei um einen gemanagten String handeln, muss aber nicht.

-> DM_Options options

Hier sind keine Optionen erforderlich, ist mit *0* zu belegen.

Rückgabewert

Zurückgegeben wird ein für die Funktionsrückgabe gültiger String oder *NULL* im Fehlerfall. Ein Fehler kann z.B. vorliegen wenn die Funktion im falschen „runstate“ aufgerufen wird, der gemanagte String ungültig ist oder das Kopieren nicht ausgeführt werden konnte.

Zu beachten

Für die Rückgabe von *output*-String-Parametern können die Funktionen DM_StringInit und DM_StringChange verwendet werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue StringOf(integer I);

on dialog start
{
  print StringOf(123);
  print StringOf(-42);
  exit();
```

```
}
```

C-Teil

```
...

DM_String DML_default DM_ENTRY StringOf(DM_Integer I)
{
  char      buf[10];
  DM_String data;
  DM_String negative;

  if (I>=0)
  {
    /* return an unmanaged local string */
    sprintf(buf, "%d", (int)I);
    data = buf;
    /* wrong: return data; => buf is a local char array! */
    return DM_StringReturn(&data, 0);
  }
  else
  {
    /* return a managed string */
    DM_StringInit(&negative, 0);  /* can be omitted */
    DM_StringChange(&negative, "!!negative", &data, 0);
    return DM_StringReturn(&negative);
    /* return &negative; => also possible for managed strings! */
  }
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_IndexReturn, DM_StringChange, DM_StringInit, DM_ValueReturn

3.77 DM_TraceMessage

Mit Hilfe dieser Funktion können Trace-Meldungen von der Anwendung in das Tracefile des Dialog Managers geschrieben werden.

```
void DML_c DM_EXPORT DM_TraceMessage
(
  DM_string string,
  DM_Options options
)
```

Parameter

-> DM_string string

Dieser Parameter ist der String, der in das Tracefile geschrieben werden soll.

-> DM_Options options

Für diese Funktion sind folgende Optionen zulässig:

Option	Bedeutung
DMF_InhibitTag	Mit Hilfe dieser Option kann die Anwendung beeinflussen, ob der Dialog Manager den Zeilenanfang ("header") „[UM]“ am Anfang einer Zeile schreibt oder nicht. Wird der Parameter auf *DMF_InhibitTag* gesetzt, wird der Zeilenanfang nicht gedruckt. Wenn diese Option nicht gesetzt ist, sieht die Meldung im Tracefile wie folgt aus: `*[UM]:Angegebener String`
DMF_Printf	Wird diese Option gesetzt, wird der Parameter *string* wie bei der C-Funktion **printf** interpretiert. Die entsprechenden zusätzlichen Parameter müssen hinter dem Parameter *options* angegeben werden. Floating-Point Formate werden hier nicht unterstützt.
DMF_LogFile	Wird diese Option gesetzt, erfolgt die Ausgabe nicht in das Tracefile, sondern in das Logfile.

Anmerkung

Die angegebenen Trace-Meldungen werden nur gedruckt, wenn der Dialog Manager mit der Trace-Option gestartet wird.

Beispiel

Ausgabe einer Meldung im Hauptprogramm, wenn die Funktion DM_Initialize FALSE als Ergebnis zurückliefert.

```
int DML_c DM_CALLBACK AppMain __2(
(int, argc),
(char **, argv))
```

```
{
    DM_ID dialogID;

    /*     * Initialisierung des Dialog Managers     */
    if (!DM_Initialize (&argc, argv, 0))
    {
        DM_TraceMessage("could not initialize",
            DMF_LogFile);
        return (1);
    }
```

3.78 DM_ValueChange

Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung.

Es findet eine automatische Umwandlung von einer ungemanagten Wertereferenz in eine gemanagte Wertereferenz statt. Ist in diesem Fall die Option *DMF_StaticValue* gesetzt, so wird eine statische bzw. globale gemanagte Wertereferenz gesetzt. Für eine bessere Kontrolle empfiehlt sich auch die explizite Nutzung von DM_ValueInit.

Handelt es sich beim *value*-Parameter um eine Sammlung, z.B. vom Typ *DT_vector*, *DT_list*, *DT_hash*, *DT_matrix* oder *DT_refvec*, so kann durch Angabe des *index*-Parameters ein Elementwert ausgetauscht werden. Analog zu den vordefinierten Attributen ist eine Erweiterung einer Sammlung möglich indem schrittweise der um *+1* erhöhte Index verwendet wird. Für assoziative Felder ist einfach die Nutzung eines noch nicht verwendeten Indexschlüssels möglich.

Wird eine Sammlung im *data*-Parameter dem Ziel komplett (also mit *NULL* als *index*-Parameter) zugewiesen, so wird der gesamte Wert mit allen Werteelementen kopiert. Die Umwandlung eines Argumentes in einen lokal gemanagten Wert erfordert ebenso ein komplettes Kopieren um die weitere Manipulation zu ermöglichen.

```
DM_Boolean DML_default DM_EXPORT DM_ValueChange
(
  DM_Value   *value,
  DM_Value   *index,
  DM_Value   *data,
  DM_Options  options
)
```

Parameter

-> DM_Value * value

Dieser Parameter verweist auf die Wertereferenz, welche verändert werden soll. Falls sie noch nicht vom IDM gemanagt wird, wird sie in eine gemanagte Wertereferenz überführt.

-> DM_Value * index

Dieser Parameter kann für die Änderung von Elementwerten in Sammlungen verwendet werden und dient zur Angabe des Index. Ansonsten sollte er mit *NULL* belegt werden. Dieser Parameter muss nicht eine gemanagte Wertereferenz sein.

-> DM_Value * data

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln. Ist dieser Wert *NULL*, so wird der Wert bzw. das Element auf *DT_undefined* gesetzt.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *value*-Parameters in eine gemanagte Wertereferenz wird diese als statische bzw. globale Wertereferenz behandelt.
DMF_AppendValue	Bei Sammlungen wird der Datenwert aus *data* hinten angehängt. Der Index muss hierfür *NULL* sein.
DMF_SortBinary	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung des neu gesetzten Wertes. Kombinierbar mit *DMF_SortReverse*.
DMF_SortLinguistic	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung, welche für Strings nach linguistischen Regeln (siehe hierzu die Funktion **sort()**) erfolgt. Kombinierbar mit *DMF_SortReverse*.
DMF_SortReverse	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung in umgekehrter Reihenfolge.

Rückgabewert

DM_TRUE	Funktion erfolgreich ausgeführt, die Wertesetzung war erfolgreich oder der Wert war schon gesetzt.
DM_FALSE	Wertesetzung konnte nicht durchgeführt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue FindData(hash DataHash, string Pattern,
                           anyvalue FirstIndex output);

on dialog start
{
  variable hash Stations := ["1"=>"ARD","2"=>"ZDF","9"=>"SWR3"];
  variable anyvalue Idx;

  print "Found(D)=" + FindData(Stations,"D",Idx);
  print " at " + Idx;
  exit();
}
```

```
...

static DM_Value InvalidIndex;
static DM_Value InvalidValue;

DM_Value * DML_default DM_ENTRY FindData(DM_Value *DataHash, DM_String
Pattern,
                                         DM_Value *FirstIndex)
{
  DM_Value index;
  DM_Value value;

  /* initialize the managed values */
  DM_ValueInit(&index, DT_void, NULL, 0);
  DM_ValueInit(&value, DT_void, NULL, 0);
  DM_StringInit(&retString, 0);

  count = DM_ValueCount(DataHash, NULL, 0);
  while(count>0)
  {
    /* loop through the hash */
    if (DM_ValueIndex(DataHash, count--, &index, 0)
        && DM_ValueGet(DataHash, &index, &value, DMF_GetLocalString)
        && value.type == DT_string)
    {
      if (strstr(value.value.string, Pattern))
      {
        /* return the first found index & value */
        DM_ValueChange(FirstIndex, NULL, &index, 0);
        return DM_ValueReturn(&value, 0);
      }
    }
  }

  /* return the invalid index & value */
  DM_ValueChange(FirstIndex, NULL, &InvalidIndex, 0);
  return &InvalidValue;
}

...

int DML_c AppMain __2((int, argc), (char **,argv))
{
  DM_Value data;

  ...
```

```
data.type = DT_string;
data.value.string = "NO-VALUE";
DM_ValueChange(&InvalidValue, NULL, &data, DMF_StaticValue);
data.type = DT_string;
data.value.string = "INVALID-INDEX";
DM_ValueChange(&InvalidIndex, NULL, &data, DMF_StaticValue);

...

DM_StartDialog(...)
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

3.79 DM_ValueCount

Liefert die Anzahl der Werte in einer Sammlung (ohne die Standardwerte) zurück. Wenn gewünscht kann auch der Indextyp bzw. der höchste Indexwert zurückgeliefert werden.

Der zurückgelieferte Wert gibt die Anzahl der Werte (ohne die Standardwerte) an. Zusammen mit der Funktion DM_ValueIndex lassen sich so einfach Schleifen über alle indizierten Werte bzw. Elemente realisieren.

Folgende Rückgaben sind je nach *value*-Parameter zu erwarten:

***value*-Typ**	***retvalue*-Rückgabetyp**	**Bemerkung**
DT_refvec, *DT_list*, *DT_vector*	*DT_integer*	Höchster Indexwert
DT_matrix	*DT_index*	Höchster Indexwert
DT_hash	*DT_datatype*	Beliebiger Indextyp (*anyvalue*)
ansonsten	*DT_void*	Unindizierter WertN

```
DM_UInt DML_default DM_EXPORT DM_ValueCount
(
  DM_Value   *value,
  DM_Value   *retvalue,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Diese Parameter verweist auf die Wertereferenz, von welcher die Werteanzahl geholt wird. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_Value * retvalue

Wenn dieser Parameter nicht *NULL* ist, wird hierin der Count-Wert zurückgeliefert. Es kann sich hierbei um eine gemanagte Wertereferenz handeln, muss aber nicht.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	**Bedeutung**
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.

Option	Bedeutung
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

0 … INT_MAX	Anzahl von Werten (ohne die Standardwerte bei Index *[0]*, *[0,*]* oder *[*,0]*).
retvalue	Höchster Indexwert, kann entweder *void* (skalarer Wert), ein *integer*-Wert (eindimensionales Feld), ein *index*-Wert (zweidimensionales Feld) oder ein Datentyp (assoziatives Feld) sein.

Beispiel

Dialogdatei

```
dialog YourDialog
function integer CountIntegers(anyvalue List);

on dialog start
{
  variable matrix M := [
    [0,0]=>-1,[1,1]=>"PLZ",[1,2]=>"City",[2,1]=>53123,[2,2]=>"Bonn" ];
  print "#Integers in Hash: " + CountInteger(M);
  exit();
}
```

C-Teil

```
...

static DM_Value InvalidIndex;
static DM_Value InvalidValue;

DM_Integer DML_default DM_ENTRY CountInteger(DM_Value *List)
```

```
{
  DM_Value   index;
  DM_Value   count;
  DM_Value   value;
  DM_Integer icount = 0;

  /* initialize the managed values */
  if (DM_ValueCount(List, &count, 0)>0 && List->type == DT_matrix
      && count->type == DT_index)
  {
    index.type = DT_index;
    index.value.index.first = 0;
    index.value.index.second = 1;

    /* loop through [0,1],[1,1],[2,1],... */
    while(index.value.index.first<=count.value.index.second)
    {
      if (DM_ValueGet(List, &index, &value, 0)
          && value.type == DT_integer)
        icount++;
      index.value.index.first++;
    }
  }
  return icount;
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_ValueChange, DM_ValueGet, DM_ValueIndex

Eingebaute Funktionen countof, itemcount

3.80 DM_ValueGet

Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört.

Ist der definierte Index *NULL*, so wird der Gesamtwert zurückgegeben, was normalerweise einem Kopieren des Wertes entspricht.

Handelt es sich beim *retvalue*-Parameter um einen ungemanagten Wert, so bleibt dieser auch ungemanagt. Bei der Rückgabe von Strings werden diese nur in einem temporären Puffer gemerkt, der beim nächsten Aufruf einer DM-Funktion ohne *DMF_DontFreeLastStrings*-Option wieder gelöscht bzw. überschrieben werden kann.

```
DM_Boolean DML_default DM_EXPORT DM_ValueGet
(
  DM_Value   *value,
  DM_Value   *index,
  DM_Value   *retvalue,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Dieser Parameter verweist auf die Wertereferenz, von welcher der Elementwert geholt wird. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_Value * index

Dieser Parameter definiert den Index von dem der Elementwert geholt wird. Dieser Parameter muss kein gemanagter Wert sein. Falls der Parameter auf *NULL* gesetzt ist, wird der *value*-Parameter in den Rückgabeparameter *retvalue* kopiert.

-> DM_Value * retvalue

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln. Ist dieser Wert *NULL*, so wird der Wert bzw. das Element auf *DT_undefined* gesetzt.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *value*-Parameters in eine gemanagte Wertereferenz wird diese als statische bzw. globale Wertereferenz behandelt.

Option	Bedeutung
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

DM_TRUE	Das Holen des Wertes war erfolgreich.
DM_FALSE	Der Wert konnte nicht geholt werden. Das kann auf fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_ValueChange, DM_ValueCount, DM_ValueIndex

3.81 DM_ValueIndex

Mit dieser Funktion kann der zu einer Position zugehörige Index einer Sammlung ermittelt werden. Dies ist vor allem wichtig für Werte vom Typ *DT_hash* und *DT_matrix* um so auf einfachste Weise auf alle zugehörigen Indizes zugreifen zu können. Die Funktion erlaubt nur den Zugriff auf Indizes die nicht zu Standardwerten gehören.

Der zurückgelieferte Index kann in einem gemanagten *retvalue*-Parameterwert abgelegt werden oder in einem ungemanagten. Bei letzterem wird wenn nötig eine Allokierung des String-Wertes im temporären Puffer ausgeführt.

```
DM_UInt DML_default DM_EXPORT DM_ValueIndex
(
  DM_Value   *value,
  DM_UInt     indexpos,
  DM_Value   *retvalue,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Diese Parameter verweist auf die Wertereferenz, von welcher der Index an der Position geholt wird. Es muss eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_UInt indexpos

Dieser Parameter definiert die Position des Index und sollte zwischen *0 < indexpos <= DM_ValueCount()* liegen.

-> DM_Value * retvalue

Wenn dieser Parameter nicht *NULL* ist, wird hier der entsprechende Index abgelegt. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

If this parameter is not *NULL*, the retrieved index is stored here. This may be a managed or an unmanaged value reference.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.

Option	Bedeutung
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

DM_TRUE	Der Wert hat einen Index an dieser Position, der geholte Index steht danach in **retvalue* falls *retvalue != NULL* ist.
DM_FALSE	Der Index konnte nicht ermittelt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Position hinweisen.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue FindData(hash DataHash, string Pattern,
                           anyvalue FirstIndex output);

on dialog start
{
  variable hash Stations := ["1"=>"ARD","2"=>"ZDF","9"=>"SWR3"];
  variable anyvalue Idx;

  print "Found(D)=" + FindData(Stations,"D",Idx);
  print " at " + Idx;
  exit();
}
```

C-Teil

```
...

static DM_Value InvalidIndex;
static DM_Value InvalidValue;
```

```
DM_Value * DML_default DM_ENTRY FindData(DM_Value *DataHash, DM_String
Pattern,
                                        DM_Value *FirstIndex)
{
  DM_Value index;
  DM_Value value;

  /* initialize the managed values */
  DM_ValueInit(&index, DT_void, NULL, 0);
  DM_ValueInit(&value, DT_void, NULL, 0);
  DM_StringInit(&retString, 0);

  count = DM_ValueCount(DataHash, NULL, 0);
  while(count>0)
  {
    /* loop through the hash */
    if (DM_ValueIndex(DataHash, count--, &index, 0)
        && DM_ValueGet(DataHash, &index, &value, DMF_GetLocalString)
        && value.type == DT_string)
    {
      if (strstr(value.value.string, Pattern))
      {
        /* return the first found index & value */
        DM_ValueChange(FirstIndex, NULL, &index, 0);
        return DM_ValueReturn(&value, 0);
      }
    }
  }

  /* return the invalid index & value */
  DM_ValueChange(FirstIndex, NULL, &InvalidIndex, 0);
  return &InvalidValue;
}

...

int DML_c AppMain __2((int, argc), (char **,argv))
{
  DM_Value data;

  ...

  data.type = DT_string;
  data.value.string = "NO-VALUE";
  DM_ValueChange(&InvalidValue, NULL, &data, DMF_StaticValue);
  data.type = DT_string;
```

```
data.value.string = "INVALID-INDEX";
DM_ValueChange(&InvalidIndex, NULL, &data, DMF_StaticValue);

...

DM_StartDialog(...)
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_ValueChange, DM_ValueCount, DM_ValueGet

Methode index

Eingebaute Funktionen countof, itemcount

3.82 DM_ValueInit

Mit dieser Funktion kann eine Wertereferenz in eine vom IDM gemanagte lokale oder globale Wertereferenz umgewandelt werden. Dadurch ist die weitere Manipulation des Wertes durch **DM_Value…()**-Funktionen möglich sowie die Rückgabe als Parameter bzw. Rückgabewert.

Die Wertereferenz wird mit dem entsprechenden Typ initialisiert. Erlaubt sind auch die Sammlungsdatentypen *DT_list*, *DT_vector*, *DT_hash*, *DT_matrix* und *DT_refvec*.

Wird die Wertereferenz über die Option *DMF_StaticValue* als statisch bzw. global initialisiert, so ist ein Zugriff auch außerhalb des Funktionsaufrufs möglich. Eine Freigabe von Wertelisten und Strings findet beim Funktionsende nicht statt. Nicht erlaubt ist die Initialisierung von Argumenten zu einer statischen bzw. globalen gemanagten Wertereferenz.

Initialisiert werden String-Werte mit dem *NULL*-Pointer. Sammlungen werden ohne Elementwerte angelegt. Auch alle anderen Wertetypen werden zu einem *0*-Wert initialisiert.

Das Hinzufügen oder Ändern von Werten oder Teilwerten bzw. Elementen kann über die Funktion DM_ValueChange geschehen. Eine Reinitialisierung über **DM_ValueInit()** ist ebenfalls möglich.

```
DM_Boolean DML_default DM_EXPORT DM_ValueInit
(
  DM_Value   *value,
  DM_Type     type,
  DM_Value   *count,
  DM_Options  options
)
```

Parameter

-> DM_Value *value

Dieses ist die Wertereferenz, welche initialisiert werden soll.

-> DM_Type type

Dieser Parameter bezeichnet den geforderten initialen Typ.

-> DM_Value *count

In diesem Parameter kann die initiale Größe von Sammlungen wie *list* oder *matrix* angegeben werden oder der zugehörige Wertetyp bei *vector*-Werten.

-> DM_Options options

Option	Bedeutung
0	Wertereferenz wird als lokaler Wert initialisiert.
DMF_StaticValue	Wertereferenz wird als globaler, statischer Wert initialisiert.

Rückgabewert

DM_TRUE	Funktion konnte erfolgreich ausgeführt werden, die Wertereferenz ist damit initialisiert.
DM_FALSE	Wertereferenz konnte nicht initialisiert werden.

Beispiel

Dialogdatei

```
dialog Dialog

function void     AppendToList(anyvalue List input output, anyvalue Value);
function anyvalue FindMinMax(anyvalue IntegerList);
function string   ListToString(anyvalue List);
function void     SetElemSep(string Sep);

on dialog start
{
   variable vector[string] WeekDays:=["Mo","Tu","Wed","Thu","Fri","Sat"];
   variable list DaysPerWeek := [31,30,28,27];

   SetElemSep(" , ");
   AppendToList(WeekDays,"Sun");
   print ListToString(WeekDays);
   print FindMinMax(DaysPerWeek);
   exit();
}
```

C-Teil

```
...

static DM_String elemSep = NULL;

void DML_default DM_ENTRY SetElemSep(DM_String sep)
{
  DM_StringInit(&elemSep, DMF_StaticValue);
  DM_StringChange(&elemSep, sep, 0);
}

DM_Value* DML_default DM_ENTRY FindMinMax(DM_Value *IntegerList)
{
  DM_Value subval, minMaxList;  /* managed local values */
  DM_UInt  count;
  DM_UInt  minValueCount=0, maxValueCount=0;
  DM_Value index, mindex, data;  /* unmanaged values */
```

```
  /* initialize local managed values */
  DM_ValueInit(&subval, DT_void, NULL, 0);
  DM_ValueInit(&minMaxList, DT_list, NULL, 0);

  /* determine the itemcount of the list */
  count = DM_ValueCount(IntegerList, NULL, 0);
  index.type = DT_integer;
  index.value.integer = 1;

  while(count>0)
  {
    /* loop through the index 1,2,3,... */
    if (DM_ValueGet(IntegerList, &index, &subval, 0)
        && subval.type == DT_integer)
    {
      if (minValueCount==0 || subval.value.integer<minValue)
      {
        minValueCount++;
        /* store maximum value at [1] in minMaxList */
        mindex.type = DT_integer;
        mindex.value.integer = 1;
        DM_ValueChange(&minMaxList, &mindex, &subval, 0);
      }
      if (maxValueCount==0 || subval.value.integer>maxValue)
      {
        maxValueCount++;
        /* store maximum value at [2] in minMaxList */
        mindex.type = DT_integer;
        mindex.value.integer = 2;
        DM_ValueChange(&minMaxList, &mindex, &subval, 0);
      }
    }
    count--;
    index.value.integer++;
  }
  /* return the minMaxList (without compiler warnings) */
  return DM_ValueReturn(&minMaxList, 0);
}

...

void DML_default DM_ENTRY AppendToList(DM_Value *List, DM_Value *Value)
{
  DM_Value newList;

  /* demonstrate the returning of a list-value by two ways * /
```

```
  if (List->type == DT_list || List->type == DT_vector)
  {
    /* 1) returning a manipulated argument (auto-management) */
    DM_ValueChange(List, NULL, Value, DMF_AppendValue);
  }
  else
  {
    /* 2) creation of a managed local list-value */
    DM_ValueInit(&newList, DT_list, NULL, 0);
    DM_ValueChange(&newList, NULL, List, DMF_AppendValue);
    DM_ValueChange(&newList, NULL, Value, DMF_AppendValue);
    *List = newList;
  }
}

DM_String DML_default DM_ENTRY ListToString(DM_Value *List)
{
  DM_Value  index;
  DM_Value  value;
  DM_UInt   count;
  DM_String retString;

  /* initialize the managed values */
  DM_ValueInit(&index, DT_void, NULL, 0);
  DM_ValueInit(&value, DT_void, NULL, 0);
  DM_StringInit(&retString, 0);

  count = DM_ValueCount(List, NULL, 0);
  while(count>0)
  {
    if (DM_ValueIndex(List, count--, &index, 0)
        && DM_ValueGet(List, &index, &value, 0)
        && value.type == DT_string)
    {
      DM_StringChange(&retString, value.value.string, DMF_AppendValue);
      if (count>0)
        DM_StringChange(&retString, elemSep, DMF_AppendValue);
    }
  }
  return DM_StringReturn(retString, 0);
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

3.83 DM_ValueReturn

Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen **DM_Value**-Werten an den Aufrufer. Bei Verwendung von lokalen Variablen und Strukturen in einer C-Funktion sind diese nach der Rückgabe ungültig. Mit dieser Funktion kann problemlos und sicher eine lokale **DM_Value**-Variable zurückgegeben werden.

Zu diesem Zweck wird, wenn nötig, eine temporäre Kopie des zurückzugebenden Wertes angelegt (z.B. der darin liegende String kopiert). Ein Kopieren findet für gemanagte Wertereferenzen nicht statt. Insofern sollte der zurückgegebene **DM_Value**-Pointer dann mit `return` an den Aufrufer weitergegeben werden.

```
DM_Value* DML_default DM_EXPORT DM_ValueReturn
(
  DM_Value   *value,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Dieser Parameter verweist auf die Wertereferenz welche zurückgegeben werden soll. Es kann sich dabei um eine gemanagte Wertereferenz handeln, muss aber nicht.

-> DM_Options options

Hier sind keine Optionen erforderlich, ist mit *0* zu belegen.

Rückgabewert

Zurückgegeben wird ein Zeiger auf eine gültige **DM_Value**-Struktur oder *NULL* im Fehlerfall. Ein Fehler kann z.B. vorliegen, wenn die Funktion im falschen „runstate“ aufgerufen wird, der gemanagte String ungültig ist oder das Kopieren nicht ausgeführt werden konnte.

Zu beachten

Für die Rückgabe von *output*-Parametern können die Funktionen DM_ValueInit und DM_ValueChange verwendet werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue StringOf(integer I);

on dialog start
{
  print StringOf(123);
  print StringOf(-42);
```

```
  exit();
}
```

C-Teil

```
...

DM_Value * DML_default DM_ENTRY StringOf(DM_Integer I)
{
  char     buf[10];
  DM_Value data;
  DM_Value negative;

  if (I>=0)
  {
    /* return an unmanaged value */
    sprintf(buf, "%d", (int)I);
    data.type = DT_string;
    data.value.string = buf;
    /* wrong: return &data => data is a local structure! */
    return DM_ValueReturn(&data, 0);
  }
  else
  {
    /* return a managed value */
    DM_ValueInit(&negative, DT_string, NULL, 0);  /* can be omitted */
    data.type = DT_string;
    data.value.string = "!!negative!!";
    DM_ValueChange(&negative, NULL, &data, 0);
    /* return &negative; => possible but generates compiler warning! */
    return DM_ValueReturn(&negative);
  }
}
```

Verfügbarkeit

Ab IDM-Version A.06.01.a

Siehe auch

Funktionen DM_ValueChange, DM_ValueCount, DM_ValueGet, DM_ValueIndex

Methode index

Eingebaute Funktionen countof, itemcount

3.84 DM_WaitForInput

Mit Hilfe dieser Funktion kann auf das Eintreffen einer speziellen Nachricht gewartet werden, ohne dass das zugrundeliegende Fenstersystem blockiert wird.

Diese Funktion ist nur unter MICROSOFT WINDOWS verfügbar.

```
DM_UInt DML_default DM_EXPORT DM_WaitForInput
(
  DM_UInt msg,
  DM_Options options
)
```

Parameter

-> DM_UInt msg

In diesem Parameter wird die Nachricht übergeben, auf die gewartet werden soll.

-> DM_UInt timeout

In diesem Parameter wird in Sekunden angeben, wie lange auf das Eintreffen der Nachricht gewartet werden soll. Eine *0* bedeutet hier, dass unendlich lange gewartet werden soll.

-> DM_Options options

Für die Belegung dieses Parameters gibt es neben *0* noch folgende Möglichkeiten:

Option	Bedeutung
DMF_IgnoreExtEvent	Diese Option bedeutet, dass beim Warten auf das angegebene Ereignis externe Ereignisse ignoriert werden sollen. Ist diese Option nicht gesetzt, so beendet ein externes Ereignis das Warten auf die angegebene Nachricht.

Rückgabewert

Wert	Bedeutung
DMF_RetInput	Dieser Wert bedeutet, dass die angegebene Nachricht eingetroffen ist.
DMF_RetTimeout	Dieser Wert bedeutet, dass die angegebene Zeit überschritten worden ist und deshalb ein Timeout eingetreten ist.
DMF_RetExtEvent	Dieser Wert bedeutet, dass ein externes Ereignis eingetroffen ist.
DMF_RetError	Beim Aufruf dieser Funktion ist ein Fehler eingetroffen. Die Ursache hierfür kann sein, dass die Message ungültig ist oder dass der aktuelle Zustand der Anwendung den Aufruf nicht zulässt. Nicht zulässig sind Aufrufe an diese Funktion aus Callback-, Canvas- und Formatfunktionen.

Anmerkung

Beim Aufruf diese Funktion werden im Dialog Manager keinerlei Ereignisse mehr verarbeitet, bis die angegebene Nachricht eingetroffen wird. Damit kann also sehr einfach ein Überlaufen der Dialog Manager internen Ereignisverarbeitung erreicht werden. Daher muss diese Funktion mit besonderer Sorgfalt eingesetzt werden, sonst kann die ganze Anwendung stillgelegt werden.

Beispiel

Asynchrone Ermittlung eines Rechner-Namens über die Funktion gethostbyname.

```
/*
** Diese Funktion hat die Steuerung. Sie lässt die freie
** Message berechnen, installiert den Input-Handler und ruft
** dann die asynchrone Funktion gethostbyname auf.
*/
static struct hostent FAR * TcpWin_gethostbyname __1(
(const char FAR *, name))
{
  HANDLE h = WSAAsyncGetHostByName (TcpWinHwnd,
    TcpWinMsgGetXByY,name,TcpWinBuffer,MAXGETHOSTSTRUCT);
  TcpWinHostent = (struct hostent FAR *) 0;

  if ((h != (HANDLE) 0)
  && (DM_InputHandler (TcpWinGetXByYHandler, (FPTR) 0,
        TcpWinMsgGetXByY, DMF_ModeMsgNotify,
        DMF_RegisterHandler, DMF_DontTrace)
    != (HWND) 0)
    && DM_WaitForInput (TcpWinMsgGetXByY, 0,
    DMF_IgnoreExtEvent | DMF_DontTrace))
    {
      DM_InputHandler (TcpWinGetXByYHandler, (FPTR) 0,
        TcpWinMsgGetXByY, DMF_ModeMsgNotify,
        DMF_WithdrawHandler,
        DMF_DontTrace | DMF_CheckFuncarg);
    }

    return (TcpWinHostent);
}
```

3.85 YiRegisterUserEventMonitor

Diese Funktion gibt es nur für die Version unter MICROSOFT WINDOWS. Sie dient dazu, eine Ereignis-Monitor-Funktion zu installieren, mit deren Hilfe DM-Ereignis-Schleifen unterbrochen werden können.

Änderung ab Version A.05.01.d

Die Monitore "YI_OBJ_MONITOR" bzw. "YI_OBJFRAME_MONITOR" werden für zusätzliche Microsoft Windows Controls aufgerufen.

Wenn ein ISA Dialog Manager Objekt aus mehreren Microsoft Windows Controls aufgebaut ist, kann der Monitor "YI_OBJ_MONITOR" für jedes dieser Microsoft Windows Controls aufgerufen werden.

```
DM_Boolean DML_default DM_EXPORT YiRegisterUserEventMonitor
(
  int which,
  YiUserEventMonitor uem
)
```

Parameter

-> int which

Dieser Parameter definiert, welche Monitor-Funktion installiert wird.

Die folgenden Werte sind möglich:

» *YI_APP_MONITOR*
Ein Applikations-Ereignis-Monitor wurde installiert.

» *YI_OBJ_MONITOR*
Ein Objekt-Ereignis-Monitor wurde installiert.

» *YI_OBJFRAME_MONITOR*
Ein Frame-Objekt-Ereignis-Monitor wurde installiert.

-> YiUserEventMonitor uem

Dieser Parameter ist die Adresse zu der zu installierenden Monitor-Funktion. Definition und Gebrauch werden unten erklärt (siehe "Anmerkungen").

Rückgabewert

TRUE	Monitor-Funktion konnte erfolgreich installiert werden.
FALSE	Monitor-Funktion konnte nicht installiert werden.

Anmerkungen

1. YiRegisterUserEventMonitor installiert einen neuen Monitor-Funktions-Pointer; dadurch wird die alte Monitor-Funktion gelöscht.
2. Wenn ein *NULL*-Pointer anstatt eines Monitor-Funktions-Pointers weitergegeben wird, wird wie-

der die Default-Funktion installiert.

3. Es ist möglich, zwei verschiedene Monitor-Funktionen in Abhängigkeit von *which* zu installieren.

3.85.1 YI_APP_MONITOR

Diese Funktion ist der Applikations-Monitor, der von der Anwendung installiert werden kann. Diese Monitor-Funktion erhält jede Message, die MICROSOFT WINDOWS an die Anwendung schickt. Sie ist folgendermaßen definiert:

```
LONG DML_default DM_CALLBACK AppMonitorFunc
(
  DM_ID id,
  MSG *pMsg
)
```

Parameter

-> DM_ID id

Dieser Parameter wird zur Zeit noch nicht benutzt und muss daher mit 0 belegt sein.

-> MSG *pMsg

Dieser Parameter ist die erhaltene Message. Die Monitor-Funktion muss diese Message in jedem Fall abarbeiten! Für alle Messages, die nicht abgearbeitet werden, muss die Funktion "YiDefAppMonitor" aufgerufen werden, die die Standard-DM-Abarbeitung ausführt.

Sie ist folgendermaßen definiert:

```
LONG DML_pascal DM_EXPORT YiDefAppMonitor (DM_ID id, MSG *pMsg)
```

Rückgabewert

Rückgabewert, der von MICROSOFT WINDOWS erwartet wird oder der Rückgabewert von "YiDefAppMonitor" muss zurückgegeben werden.

3.85.2 YI_OBJ_MONITOR

Diese Monitor-Funktion erhält jede Message, die Microsoft Windows zu den DM-Objekten schickt. Sie ist folgendermaßen definiert:

```
LONG DML_pascal DM_CALLBACK ObjectMonitorFunc
(
  DM_ID id,
  MSG *pMsg
)
```

Parameter

-> DM_ID id

Dieser Parameter ist der Identifikator des DM-Objekts.

-> MSG *pMsg

Dieser Parameter ist die erhaltene Message. Die Monitor-Funktion muss diese Message in jedem Fall abarbeiten! Für alle Messages, die nicht abgearbeitet werden, muss die Funktion "YiDefObjMonitor" aufgerufen werden, die die Standard-DM-Abarbeitung ausführt.

Sie ist folgendermaßen definiert:

```
LONG DML_pascal DM_EXPORT YiDefObjMonitor (DM_ID id, MSG *pMsg)
```

Rückgabewert

Rückgabewert, der von MICROSOFT WINDOWS erwartet wird oder der Rückgabewert von "YiDefObjMonitor" muss zurückgegeben werden.

Wichtige Anmerkungen

Wenn beim Implementieren der Monitor-Funktionen ein Fehler auftritt, insbesondere wenn die Default-Funktion nicht aufgerufen wird oder die falsche Default-Funktion aufgerufen wird, kommt es zu einem Systemabsturz. Daher sollten nur selbst definierte Messages oder DDE-Messages in dem Applikationsmonitor abgearbeitet werden. Alle anderen Messages sollten an die Funktion "YiDefAppMonitor" übergeben werden.

Wenn möglich, sollte keine Objekt-Monitor-Funktion installiert werden.

Für die Definition spezieller Objekte steht das Objekt canvas zur Verfügung.

3.85.3 YI_OBJFRAME_MONITOR

Diese Monitor-Funktion erhält jede Message, die Microsoft Windows zu den MSW -Frame-Fenstern schickt; dies betrifft nur die DM-Objekte, die aus mehreren MSW -Fenstern zusammengesetzt sind.

Diese Funktion ist folgendermaßen definiert:

```
LONG DML_pascal DM_CALLBACK ObjectFrameMonitorFunc
(
  DM_ID id,
  MSG *pMsg
)
```

Parameter

-> DM_ID id

Dieser Parameter ist der Identifikator des DM-Objekts.

-> MSG *pMsg

Dieser Parameter ist die erhaltene Message. Die Monitor-Funktion muss diese Message in jedem Fall abarbeiten! Für alle Messages, die nicht abgearbeitet werden, muss die Funktion "YiDefObjFrameMonitor" aufgerufen werden, die die Standard-DM-Abarbeitung ausführt.

Sie ist folgendermaßen definiert:

```
LONG DML_pascal DM_EXPORT YiDefObjFrameMonitor (DM_ID id, MSG *pMsg)
```

Rückgabewert

Rückgabewert, der von MICROSOFT WINDOWS erwartet wird oder der Rückgabewert von "YiDefObjFrameMonitor" muss zurückgegeben werden.

Wichtige Anmerkungen

Wenn beim Implementieren der Monitor-Funktionen ein Fehler auftritt, insbesondere wenn die Default-Funktion nicht aufgerufen wird oder die falsche Default-Funktion aufgerufen wird, kommt es zu einem Systemabsturz. Daher sollten nur selbst definierte Messages oder DDE-Messages in dem Applikationsmonitor abgearbeitet werden. Alle anderen Messages sollten an die Funktion "YiDefObjFrameMonitor" übergeben werden.

Wenn möglich, sollte keine Objekt-Frame-Monitor-Funktion installiert werden.

Für die Definition spezieller Objekte steht das Objekt canvas zur Verfügung.

4 Optionen der Schnittstellen-Funktionen

In der nachfolgenden Tabelle werden alle Optionen aufgeführt, die im Parameter *options* der DM-Schnittstellen-Funktionen angegeben werden können. Im Normalfall können diese mit einem logischen „oder“ verknüpft werden, so dass eine Funktion mit mehr als einer gültigen Option aufgerufen werden kann.

Option	Funktion	Bedeutung
DMF_AcceptChild	DM_GetValue DM_SetValue	Ist gültig beim Attribut AT_options der Canvas unter Motif. Damit wird eine Canvas bezeichnet, die Kinder haben kann.
DMF_AppendValue	DM_StringChange DM_ValueChange	Anhängen eines Datenwerts an eine Sammlung (wenn *index* == *NULL*) bzw. String.
DMF_CheckFuncarg	DM_InputHandler	Es sollen alle Funktionsargumente genommen werden, um die Funktion zu suchen, die deinstalliert werden soll.
DMF_CreateInvisible	DM_CreateObject	Das neu generierte Objekt soll unsichtbar generiert werden; unabhängig davon, wie das Attribut AT_visible in dem Kopiermodell gesetzt ist.
DMF_CreateModel	DM_CreateObject	Das neu zu generierende Objekt soll als Modell generiert werden.
DMF_DisableHandler	DM_DispatchHandler DM_ErrorHandler DM_InputHandler DM_NetHandler	Deaktiviert eine registrierte Handler-Funktion.

Option	Funktion	Bedeutung
DMF_DontFreeLastStrings	DM_GetContent DM_GetValue DM_GetValueIndex DM_GetVectorValue DM_ValueCount DM_ValueGet DM_ValueIndex	Normalerweise werden bei jedem Aufruf an **DM_Get***- bzw. **DM_Value***-Funktionen die internen Strings überschrieben. Durch diese Option können die Strings länger gültig gehalten werden.
DMF_DontTrace	DM_DispatchHandler DM_InputHandler DM_QueueExtEvent DM_SendEvent DM_WaitForInput	Funktionsaufruf soll nicht mitprotokolliert werden.
DMF_DontWait	DM_EventLoop	Die Ereignisverarbeitung soll nicht "blockierend" durchgeführt werden, d.h. ist ein Ereignis da, soll es verarbeitet werden; ist kein Ereignis da, soll sofort in die aufrufende Funktion zurückgesprungen werden.
DMF_EnableHandler	DM_DispatchHandler DM_ErrorHandler DM_InputHandler DM_NetHandler	Reaktiviert eine zuvor deaktivierte Handler-Funktion.
DMF_FatalNetErrors	DM_Initialize	Stellt ein kompatibles Verhalten des DISTRIBUTED DIALOG MANAGERS (DDM) zu den IDM-Versionen vor A.05.01.d ein, mit einem sofortigen Abbruch auf Client- und Server-Seite bei Netzwerk-, Protokoll- und Versionsfehlern.
DMF_ForceDestroy	DM_Destroy DM_StopDialog	Diese Option zeigt an, dass das angegebene Objekt gelöscht werden soll. Bei DM_Destroy muss diese Option gesetzt sein, wenn das Objekt wirklich gelöscht werden soll.

Option	Funktion	Bedeutung
DMF_GetLocalString	DM_GetValue DM_GetValueIndex DM_GetVectorValue DM_ValueCount DM_ValueGet DM_ValueIndex	Text soll als String in der aktuell eingestellten Sprache zurückgeliefert werden.
DMF_GetMasterString	DM_GetValue DM_GetValueIndex DM_GetVectorValue DM_ValueCount DM_ValueGet DM_ValueIndex	Text soll als String in der Originalsprache zurückgeliefert werden.
DMF_GetTextID	DM_GetValue DM_GetValueIndex DM_GetVectorValue	Text soll als Textid zurückgeliefert werden.
DMF_IgnoreExtEvent	DM_WaitForInput	Externe Ereignisse sollen ignoriert werden.
DMF_IncludeIdent	DM_ErrMsgText	Der Name des Teils, der den Fehler erzeugt hat, soll in der Fehlermeldung enthalten sein. Dies kann das Betriebssystem, das Fenstersystem oder der DM sein.
DMF_IncludeModule	DM_ErrMsgText	Der Name des Moduls, in dem der Fehler aufgetreten ist, soll in der Fehlermeldung enthalten sein.
DMF_IncludeSeverity	DM_ErrMsgText	Der Grad des Fehlers (Warnung, Fehler, fataler Fehler) soll in dem Fehlertext enthalten sein.
DMF_IncludeText	DM_ErrMsgText	Der eigentliche Fehlertext soll in der Fehlermeldung enthalten sein.
DMF_InheritFromModel	DM_CreateObject	Das Objekt soll einschließlich der bei dem Modell angegebenen Kinder generiert werden.

Option	Funktion	Bedeutung
DMF_Inhibit	DM_SetContent DM_SetValue DM_SetValueIndex DM_SetVectorValue	Für das Setzen des Attributwerts soll kein Ereignis erzeugt werden und damit keine Regel "on Objekt .Attribute changed" ausgelöst werden.
DMF_InhibitTag	DM_TraceMessage	Beim Tracing soll der Zeilenanfang "[UM]" nicht gedruckt werden.
DMF_InstallHead	DM_DispatchHandler	Installiert die Handler-Funktion am Anfang der Liste gleichartiger Handler-Funktionen.
DMF_InstallTail	DM_DispatchHandler	Installiert die Handler-Funktion am Ende der Liste gleichartiger Handler-Funktionen.
DMF_LogFile	DM_TraceMessage	Ausgabe erfolgt nicht in das Tracefile, sondern in das Logfile.
DMF_NoCriticalSection	DM_QueueExtEvent DM_SendEvent	Verhindert unter MICROSOFT WINDOWS die Benutzung einer „Critical Section“.
DMF_NoFocusFrame	DM_GetValue DM_SetValue	Ist gültig bei Attribut AT_options der Canvas unter Motif. Damit wird eine Canvas bezeichnet, die keinen Fokusrahmen haben soll.
DMF_OmitActive	DM_GetContent DM_SetContent	Das Attribut AT_active soll nicht mit übergeben werden.
DMF_OmitSensitive	DM_GetContent DM_SetContent	Das Attribut AT_sensitive soll nicht mit übergeben werden.
DMF_OmitStrings	DM_GetContent DM_SetContent	Die Strings sollen nicht mit übergeben werden.
DMF_OmitUserData	DM_GetContent	Das Attribut AT_userdata soll nicht gefüllt werden.
DMF_Printf	DM_TraceMessage	Parameter "string" wird wie bei der C-Funktion "printf" interpretiert.
DMF_RegisterHandler	DM_DispatchHandler DM_ErrorHandler DM_InputHandler DM_NetHandler	Über diese Option wird ein neuer Handler installiert.

Option	Funktion	Bedeutung
DMF_ReplaceFunctions	DM_BindFunctions	Eine bei dem Objekt vorhandene Funktionstabelle wird durch die neue vollständig ersetzt.
DMF_SaveAll	DM_SaveProfile	Speichert auch geerbte Werte von konfigurierbaren Records in der Konfigurationsdatei.
DMF_SetCodePage	DM_Control DM_ControlEx	Die angegebene Codepage soll in der Anwendung verwendet werden und alle Texte in diese Codepage umgewandelt werden.
DMF_SetFormatCodePage	DM_Control DM_ControlEx	Definiert die Codepage, in der Formatfunktionen Strings interpretieren und zurückgeben.
DMF_SetUserCodePage	DM_ControlEx	Stellt den Zeichencode für **iconv** ein, und beeinflusst damit indirekt die IDM Codepage **CP_ucp**. Diese wird mittels **DMF_SetCodePage** aktiviert. (Nur auf Plattformen, die iconv unterstützen).
DMF_ShipEvent	DM_SetContent DM_SetValue DM_SetValueIndex DM_SetVectorValue	Für das Setzen des Attributs soll ein Ereignis erzeugt werden und damit die möglicherweise vorhandene Regel ".attribute changed" ausgelöst werden.
DMF_SignalMode	DM_Control DM_ControlEx	Über diese Option wird eingestellt, auf welche Art und Weise der Dialog Manager Signale abfangen soll.
DMF_Silent	DM_BindCallBacks	Es sollen keine Fehlermeldungen über fehlende oder zu viel definierte Funktionen ausgedruckt werden.
DMF_SortBinary	DM_ValueChange	Binäres Sortieren einer Sammlung.
DMF_SortLinguistic	DM_ValueChange	Sortieren einer Sammlung nach linguistischen Regeln.

Option	Funktion	Bedeutung
DMF_SortReverse	DM_ValueChange	Sortieren einer Sammlung in umgekehrter Reihenfolge.
DMF_StaticValue	DM_StringChange DM_StringInit DM_ValueChange DM_ValueGet DM_ValueInit	Umwandlung in eine statische bzw. globale Wertereferenz bei der automatischen Umwandlung in eine gemanagte Wertereferenz.
DMF_Synchronous	DM_QueueExtEvent DM_SendEvent	Kann gesetzt werden, um interne Prozesse zu optimieren, wenn die Funktion nicht aus einem „Signal Handler“ aufgerufen wird.
DMF_UpdateScreen	DM_Control DM_ControlEx	Alle intern abgelaufenen Aktionen sollen auf dem Bildschirm sichtbar gemacht werden.
DMF_UseUserData	DM_SetContent	In dem DM_Content-Vektor soll die Userdata beachtet und dem Objekt zugewiesen werden.
DMF_Verbose	DM_BindCallBacks DM_DataChanged	DM_BindCallbacks: Es sollen Fehlermeldungen über fehlende oder zu viel definierte Funktionen ausgedruckt werden. DM_DataChanged: Aktiviert das Tracing dieser Funktion.
DMF_WaitForEvent	DM_EventLoop	Die Funktion soll auf genau ein Ereignis warten und dann zurückkehren.
DMF_WithDrawHandler	DM_DispatchHandler DM_ErrorHandler DM_InputHandler DM_NetHandler	Diese Option deinstalliert eine Handler-Funktion.
DMF_XlateString	DM_SetValue DM_SetValueIndex DM_SetVectorValue	Der angegebene String soll, bevor er dem Objekt zugewiesen wird, in die aktuell eingestellte Sprache übersetzt werden. Dies geht natürlich nur, wenn der Text bereits intern vorhanden ist und ebenfalls eine Übersetzung vorliegt.

Index

E

F

T

U

V

W

X

Y

Z