ISA Dialog Manager

COBOL-Schnittstelle

A.06.03.b

In diesem Handbuch ist die Schnittstelle des ISA Dialog Managers für in COBOL entwickelte Anwendungen beschrieben. Die Datentypen, alle Funktionen der COBOL Schnittstelle sowie das Übersetzen und Linken der Anwendungen werden erläutert.

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 Programming Interface" bezeichnet), die vom Dialog Manager (DM) für in COBOL geschriebene Anwendungsprogramme angeboten wird.

Da es gravierende Unterschiede gibt, wie COBOL-Compiler an andere Sprachen - z.B. "C" - anschließen, sind mehrere APIs verfügbar.

Dieses Handbuch beschreibt die Implementierung und den Gebrauch der DM-API zu COBOL-Programmen, die mit dem MICRO FOCUS COBOL-Compiler, kompiliert worden 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.

Das Anwendungsprogramm ist also fähig, auf Daten des laufenden Dialogs zuzugreifen und diese zu modifizieren. Dies kann am besten an einem kleinen Beispiel gezeigt werden:

Der Ausdruck, der in den Dialogregeln als

```
MainWindow.title := EdittextTitle.content
```

geschrieben ist, erhält die gegenwärtigen Daten für das Attribut *.content* des Objektes *EdittextTitle*, und setzt das Attribut *.title* des Objektes *MainWindow* auf diese Daten.

Das Ganze könnte auch innerhalb eines Anwendungsprogramms durch die Verwendung der API-Aufrufe

DMcob_GetValue und **DMcob_SetValue**

erreicht werden.

Die Zielgruppe für dieses Handbuch sind Programmierende, die sowohl mit der DM-Entwicklungsumgebung als auch mit dem MICRO FOCUS COBOL-System vertraut sind. Voraussetzung sind außerdem gute Kenntnisse des verwendeten Betriebssystems und der Werkzeuge, die dieses Betriebssystem bietet.

Bitte beachten Sie insbesondere das Kapitel "Aufruf von Unterprogrammen". Sie sollten die Regeln, die das Laden der Programme im COBOL-System bestimmen, vollständig verstanden haben, da diese Features für das hier beschriebene API zentrale Bedeutung haben.

2 COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Verfügbarkeit

IDM für MICROSOFT WINDOWS ab IDM-Version A.06.01.d.

2.1 Unicode-Unterstützung

Die COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL unterstützt Unicode als Zeichencodierung. Ein Unicode String wird hierbei von MICRO FOCUS VISUAL COBOL als *National Character* dargestellt, wobei die Codierung der IDM-Codepage *CP-utf16* entspricht. Innerhalb von MICRO FOCUS VISUAL COBOL wird dies als *PIC N NATIONAL* repräsentiert.

2.1.1 Aktivierung von Unicode

Die Verwendung von Unicode wird durch Setzen der Anwendungscodepage aktiviert. Dies kann durch eine Startoption oder innerhalb der Anwendung mit der Funktion DMcob_Control geschehen:

```
...
WORKING-STORAGE SECTION.

01  ACTION  PIC 9(4) BINARY VALUE 0.
...

PROCEDURE DIVISION
...
    MOVE DMF-SetCodePage TO ACTION.
    MOVE CP-utf16 TO DM-options.
    CALL "DMcob_Control" USING DM-STDARGS NULL-OBJECT ACTION.
...
```

Hinweise

- Über COBOL Compiler-Direktiven lässt sich die Codepage von *National Character* steuern. Die gesetzte Anwendungscodepage muss dieser Codierung entsprechen. Ein *National Character* beansprucht 2 Byte. Derzeit benutzen die IDM-Codepages *CP-utf16*, *CP-utf16b*, *CP-utf16l* und *CP-utfwin* eine 2-Byte-Repräsentation.
- Es gibt Erweiterungen im IDM um die verwendete Codepage selektiv für einzelne Funktionen einer Anwendung zu ändern. Hierdurch ist ein inkrementeller Umstieg auf Unicode möglich.

2.1.2 Erweiterung der Copy-Strecken

Die Copy Strecken sind so definiert, dass Texte wahlweise als Character oder als National Character abgelegt werden können. Hierbei bleibt der Name für den Character-Eintrag erhalten. Möchte man auf den National-Character-Eintrag zugreifen, muss dem Namen ein „-u“ angehängt werden.

Character		National Character (UTF-16)	
`DM-setsep`	`pic X.`	`DM-setsep-u`	`pic N national.`
`DM-getsep`	`pic X.`	`DM-getsep-u`	`pic N national.`
`DM-usercodepage`	`pic X(32).`	`DM-usercodepage-u`	`pic N(32) national.`
`DM-value-string`	`pic X(80).`	`DM-value-string-u`	`pic N(80) national.`
`DM-va-value-string`	`pic X(80).`	`DM-va-value-string-u`	`pic N(80) national.`

Bei folgenden Strukturelementen des ValueRecords wird die Länge bzw. Größe als Anzahl der Zeichen, nicht der Bytes, angegeben:

- `DM-value-string-putlen`
- `DM-value-string-getlen`
- `DM-value-string-size`

Dasselbe gilt für alle anderen auf dem ValueRecord basierenden Strukturen.

Hinweis

Der IDM liest bzw. schreibt die entsprechenden Werte entsprechend der aktuell geltenden Anwendungscodepage.

2.1.3 Aufrufparameter von COBOL-Funktionen

Bei allen Texten kann wahlweise Character oder National Character verwendet werden. Der IDM interpretiert den Text entsprechend der aktuell geltenden Anwendungscodepage. Die Längenangabe bezieht sich auf die Anzahl der Zeichen, nicht der Bytes. Dies gilt sowohl für die IDM-Schnittstellenfunktionen als auch für die vom IDM aufgerufenen COBOL-Funktionen. Es ist wichtig, dass die aktuell verwendete Anwendungscodepage der Textdefinition entspricht.

2.1.4 Funktionen mit Records als Parametern

Für Funktionen mit ***Record***-Objekten als Parametern wird von der Anwendung **pidm** mit der Startoption **+writetrampolin** eine Copy-Strecke erstellt. Um bestehende Anwendungen nicht zu beeinträchtigen, werden diese Copy-Strecken standardmäßig ohne Unterstützung für *National Character*

erzeugt. Die Unterstützung dafür muss explizit angefordert werden, indem an Stelle der COBOL Compiler-Option **-mfviscob** die Option **-mfviscob-u** verwendet wird:

```
pidm mydlg.dlg -mfviscob-u +writetrampolin myappl_tr
```

Die erzeugte Copy-Strecke kann dann wahlweise Character oder National Character enthalten. Der IDM greift dann wieder entsprechend der aktuell eingestellten Anwendungscodepage zu.

2.2 Datentyp DT-anyvalue

Der Datentyp *DT-anyvalue* wird über den Zeiger-Datentyp (*POINTER*) von MICRO FOCUS VISUAL COBOL unterstützt.

Siehe auch

Kapitel „Verwendung des Datentyps anyvalue“

2.3 Unterstützung von Sammlungen

Die Sammlungsdatentypen des IDM und die damit zusammenhängenden Funktionen für gemanagte (verwaltete) IDM-Werte (**Managed DM-Values**) werden über den Zeiger-Datentyp (POINTER) von MICRO FOCUS VISUAL COBOL unterstützt.

DM-Datentyp	Visual-COBOL-Datentyp
hash	POINTER
list	POINTER
matrix	POINTER
refvec	POINTER
vector	POINTER

Ein **Managed DM-Value** wird als Zeiger an MICRO FOCUS VISUAL COBOL übergeben:

```
01 ManagedValue pointer.
ENTRY "GetAnyValue" using DM-COMMON-DATA ManagedValue.
```

Um einen solchen Wert innerhalb der **DM-Value**-Struktur zu verwenden, wird der Wert nach *DM-value-pointer* kopiert und *DM-datatype* auf *DT-anyvalue* gesetzt.

```
MOVE DT-anyvalue TO DM-datatype.
MOVE ManagedValue To DM-value-pointer.
```

2.3.1 Datenfunktionen in COBOL

Mit der Unterstützung der Sammlungsdatentypen ist auch eine Implementierung von Datenfunktionen in COBOL möglich.

Siehe auch

Kapitel „Datenfunktionen“

Kapitel „Datenfunktions-Struktur DM-Datafunc-Data“

Funktion DMcob_DataChanged

2.3.2 Struktur DM-ValueIndex

Diese Struktur ermöglicht ein vollwertiges Index-Argument. Sie wird von den **DMcob_Value*** Funktionen der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL für die Verarbeitung der **Sammlungsdatentypen** genutzt. Die Struktur steht aber für alle unterstützten COBOL-Varianten zur Verfügung und ist nicht auf diese Anwendungsbereiche beschränkt. Sie ist identisch mit der Struktur DM-Value.

2.3.3 Funktionen für die Bearbeitung von Sammlungen

- DMcob_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.
- DMcob_ValueChangeBuffer
  Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung. Im Unterschied zu **DMcob_ValueChange** kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.
- DMcob_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.
- DMcob_ValueGet
  Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört.
- DMcob_ValueGetBuffer
  Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört. Im Unterschied zu **DMcob_ValueGet** kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.
- DMcob_ValueGetType
  Diese Funktion dient zur Abfrage des Datentyps eines vom IDM verwalteten Wertes.

» DMcob_ValueIndex
Mit dieser Funktion kann der zu einer Position zugehörige Index einer Sammlung ermittelt werden.

» DMcob_ValueInit
Mit dieser Funktion kann ein Wert in eine vom IDM gemanagte lokale oder globale Wertereferenz umgewandelt werden. Dadurch ist die weitere Manipulation des Wertes durch **DMcob_Value…()**-Funktionen möglich sowie die Rückgabe als Parameter bzw. Rückgabewert.

3 Prinzipielle Arbeitsweise

In diesem Kapitel wird die prinzipielle Arbeitsweise des COBOL-Interfaces vorgestellt. Dabei soll die Philosophie deutlich gemacht werden, in der mit Hilfe des Dialog Managers geschriebene Programme realisiert werden sollen.

Bei der Realisierung von Anwendungsprogrammen sollte man immer das Schichtenmodell nach Seeheim im Kopf behalten, das eine strenge Trennung der verschiedenen Softwareschichten vorschreibt. Bei Benutzung des Dialog Managers zur Programmierung von Benutzeroberflächen können dabei die Präsentations - und die Dialogschicht fast vollständig in der Regelsprache realisiert werden. Lediglich die Zugriffe auf Objekte mit Listencharakter (Tablefield, Listbox und Poptext) sollten aus Gründen der Effizienz in COBOL realisiert werden. Die restlichen COBOL-Funktionen sollten im Idealfall keinerlei Wissen über die Oberfläche haben. Sie sollten daher immer alle benötigten Informationen vom Dialog in Form von Parametern übergeben bekommen und dann keinerlei Aufrufe an den Dialog Manager durchführen; sie sollten also keine "DMcob_"-Funktionen enthalten. Wenn Funktionen auf diese Art und Weise realisiert werden, ist die eigentliche Anwendungslogik und -verarbeitung in den COBOL-Funktionen enthalten, die unabhängig von der verwendeten Oberfläche sind. Diese Vorgehensweise ist auf jeden Fall beim Einsatz des verteilten Dialog Managers notwendig, da Funktionsaufrufe über das Netzwerk vergleichsweise teuer und daher ineffizient sind.

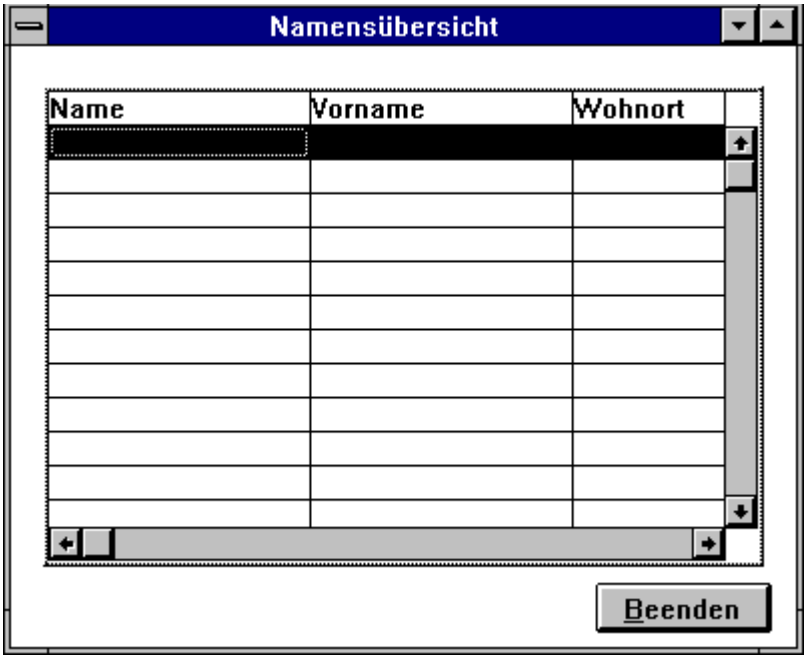
Eine Ausnahme hierzu stellt nur das eigentliche Hauptprogramm der Anwendung dar, da hier Zugriffe auf den Dialog Manager gemacht werden und daher Wissen über die Oberfläche notwendig ist.

Um das COBOL-Programm passend zu der entwickelten Oberfläche schreiben zu können, müssen zunächst einige Datenstrukturen im COBOL-Interface geklärt und die Abbildung der im Dialog vorhandenen Datentypen auf die in COBOL vorhandenen Datentypen definiert werden. Anhand eines Beispiels, das über die nachfolgenden Kapitel verteilt ist, soll die prinzipielle Vorgehensweise gezeigt werden.

3.1 Definition des Dialogs und der Ablaufsteuerung

Im Dialog Manager Editor soll ein Dialogsystem gebaut werden, das zunächst dem Anwender eine Übersicht über Namen anbietet, die er dann in einem Unterfenster modifizieren kann. Da die Übersicht sehr viele Namen enthalten kann, sollen diese immer nur teilweise geladen und angezeigt werden. Dazu muss dann eine Nachladefunktion am Tabellenelement definiert werden. Die vollständigen zu einem Namen gehörenden Daten werden in einem separatem, als Dialogbox definierten Unterfenster dargestellt, in dem sie vom Anwender verändert werden können.

Das Übersichtsfenster hat folgendes Aussehen:



Im Dialogskript ist dieses Fenster wie folgt definiert:

```
window WnUebersicht
{
  .visible false;
  .active false;
  .xleft 359;
  .width 50;
  .ytop -2;
  .height 16;
  .iconic false;
  .iconifyable true;
  .title "Namens\374bersicht";

  child tablefield T1
  {
    .visible true;
    .xauto 0;
    .xleft 1;
    .xright 1;
    .yauto 0;
    .ytop 0;
    .ybottom 1;
    .posraster true;
    .sizeraster true;
    .fieldshadow false;
    .contentfunc TABFUNC;
    .selection[sel_row] true;
    .selection[sel_header] false;
```

```
    .selection[sel_single] false;
    .colcount 3;
    .rowcount 30;
    .rowheader 1;
    .colheadshadow false;
    .colfirst 1;
    .rowfirst 2;
    .colwidth[0] 12;
    .rowheight[0] 1;
    .content[1,1] "Name";
    .content[1,2] "Vorname";
    .content[1,3] "Wohnort";
    .xraster 10;
    .yraster 16;
  }
  child pushbutton PENDE
  {
    .xleft 36;
    .width 11;
    .yauto -1;
    .text "&Beenden";
  }
}
```

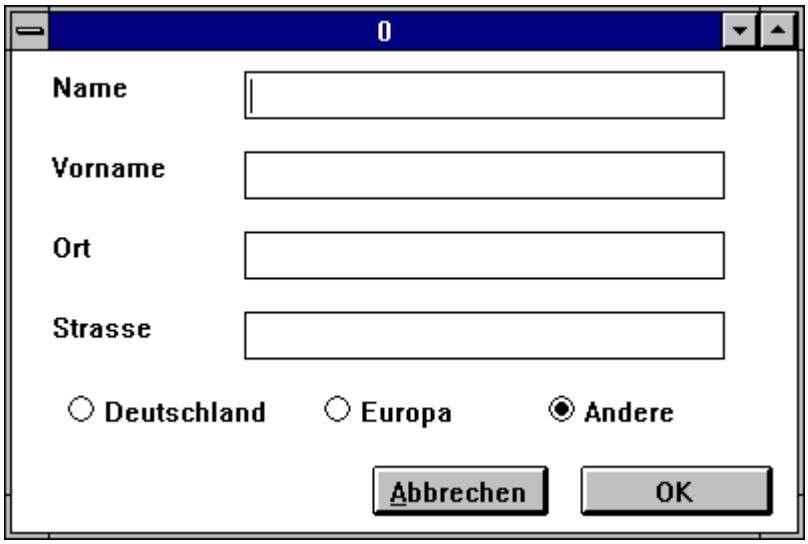
Die Definition für das Fenster, in dem die Daten geändert werden können, sieht wie folgt aus:

```
window WnName
{
  .visible false;
  .xleft 257;
  .width 50;
  .ytop 223;
  .height 13;
  .dialogbox true;

  child Eintrag Lname
  {
    .ytop 0;
    .S.text "Name";
    .E.active false;
    .E.content "";
  }

  child Eintrag Lfirstname
  {
    .ytop 2;
    .S.text "Vorname";
```

```
  .E.content "";
}

child Eintrag Lcity
{
  .ytop 4;
  .S.text "Ort";
  .E.content "";
}

child Eintrag Lstreet
{
  .ytop 6;
  .S.text "Strasse";
  .E.content "";
}

child pushbutton PbAbbrechen
{
  .xleft 22;
  .width 11;
  .ytop 10;
  .text "&Abbrechen";
}

child pushbutton PbOk
{
  .xauto -1;
  .width 12;
  .xright 1;
  .ytop 10;
  .text "OK";
}

child radiobutton XX
{
  .userdata 0;
  .visible false;
  .text "Dummy";
}

child radiobutton GERMANY
{
  .userdata 1;
  .active true;
  .xleft 3;
```

```
    .ytop 8;
    .posraster true;
    .sizeraster true;
    .text "Deutschland";
    .LandesCode := 1;
  }

  child radiobutton EUROPE
  {
    .userdata 2;
    .xleft 19;
    .ytop 8;
    .posraster true;
    .sizeraster true;
    .text "Europa";
    .LandesCode := 2;
  }

  child radiobutton OTHER
  {
    .userdata 3;
    .xleft 33;
    .ytop 8;
    .text "Andere";
    .LandesCode := 3;
  }
}
```

Dieses Fenster hat dann folgendes Aussehen:



Zur einfacheren Definition dieses Fensters wurde ein Modell eingeführt, das aus einer Groupbox mit einem statischen und editierbaren Text als Kind besteht. Die entsprechende Definition im Dialogskript sieht wie folgt aus:

```
model groupbox Eintrag
{
  .xleft 2;
  .width 45;
  .height 2;
  .borderwidth 0;

  child statictext S
  {
    .sensitive false;
    .xleft 0;
    .ytop 0;
  }

  child edittext E
  {
    .xleft 12;
    .width 30;
    .ytop 0;
  }
}
```

Für die Kommunikation mit dem COBOL-Programm wurden folgende Funktionen definiert:

- TABFUNC
  Diese Funktion ist die Nachladefunktion, die die Daten in das Tablefield nachladen soll.
  ```
  function cobol contentfunc TABFUNC();
  ```
- FILLTAB
  Mit Hilfe dieser Funktion soll das Tablefield initial teilweise gefüllt werden.
  ```
  function cobol void FILLTAB(object Table input, integer Count input);
  ```
- GETADDR
  Diese Funktion soll die zu einer Zeile gehörenden vollständigen Daten laden und an den Dialog übergeben.
  ```
  function cobol void GETADDR(record Address input output);
  ```
- PUTADDR
  Diese Funktion soll die vom Dialog geänderten Daten übernehmen und abspeichern.
  ```
  function cobol void PUTADDR(record Address input);
  ```

Die beiden letzten Funktionen erhalten jeweils einen Record als Parameter, dessen Elemente als Verweise (Links) auf die entsprechenden Elemente im Fenster definiert sind.

```
record Address
{
  string[25] NName      shadows  Lname.E.content;
```

```
  string[15] FirstName  shadows  Lfirstname.E.content;
  string[25] City       shadows  Lcity.E.content;
  string[30] Street     shadows  Lstreet.E.content;
  integer    Country    shadows  WnName.AktivesLand;
}
```

Damit die Funktionen aufgerufen werden, wurden folgende Regeln definiert:

- Beim Programmstart wird das Tablefield teilweise gefüllt und das zugehörige Fenster sichtbar gemacht.

```
on dialog start
{
    FILLTAB(T1, 20);
    T1.rowcount := 300;
    WnUebersicht.visible := true;
}
```

- Durch Selektion des Ende-Pushbuttons wird das Programm beendet. Das Programm wird auch beendet, wenn das Hauptfenster über den Schließen-Eintrag im Systemmenü geschlossen wird.

```
on PENDE select
{
    exit();
}
on WnUebersicht close
{
    exit();
}
```

- Wenn einer der Radiobuttons selektiert wird, wird dessen Wert am zugehörigen Fenster gemerkt.

```
on TABLEDEMO.RADIOBUTTON select
{
    this.window.AktivesLand := this.LandesCode;
}
```

- Bei Selektion des Abbrechen-Pushbuttons wird das zugehörige Fenster geschlossen.

```
!!Wenn der Abbrechen-Pushbutton selektiert wird,
!!einfach das zugehörige Fenster schließen.
on PbAbbrechen select
{
    this.window.visible := false;
}
```

- Wird der OK-Pushbutton selektiert, wird die Funktion PUTADDR aufgerufen und ihr die Werte übergeben.

```
!!Wenn der OK Button selektiert wird, das Fenster schließen und die
!!Änderungen in das COBOL-Pogramm übernehmen.
on PbOk select
{
    PUTADDR(Address);
```

```
    this.window.visible := false;
}
```

» Wenn das Tablefield mit einem Doppelklick selektiert wird, werden alle Daten, die zu dieser Zeile gehören, mit Hilfe der Funktion GETADDR und das Detailfenster geöffnet.

```
!! Wenn ein Doppelklick in das Innere des Tabellenelementes
!! erfolgt, soll der entsprechende Eintrag in dem separaten
!! Fenster bearbeitet werden können. Dazu muss die COBOL-
!! Funktion aufgerufen werden, die die
!! Daten holt und dann an das neue Fenster übergibt.
on T1 dbselect
{
    !! Zuerst nachschauen, ob in der Tabelle wirklich
    !! etwas selektiert ist.
    if (first(this.activeitem) > 1) then
        Address.NName := this.content
                [first(this.activeitem), 1];
        Address.FirstName := this.content
                [first(this.activeitem), 2];
        Address.City := this.content
                [first(this.activeitem), 3];
        !! Diesen Eintag nehmen und an das
        !! COBOL-Pogramm übergeben
        GETADDR(Address);
        WnName.title := ("Name Nummer: "
            + itoa((first(this.activeitem) - 1)));
        WnName.visible := true;
    else
        !! kein sinnvoller Eintrag getroffen. Ton ausgeben
        beep();
    endif
}
```

3.2 Definition der COBOL-Programme

Um nun zu dieser definierten Oberfläche ein COBOL-Programm schreiben zu können, muss man folgendes wissen:

» Wie werden Dialog-Datentypen auf COBOL-Datentypen abgebildet.

» Welche wichtigen Datenstrukturen gibt es im Dialog Manager, die zur Kommunikation zwischen dem COBOL-Programm und dem Dialog Manager dienen.

» Wie sieht ein Hauptprogramm aus, wenn die Anwendung mit Dialog Manager geschrieben werden soll.

» Wie sehen vom Dialog Manager aus aufgerufene COBOL-Unterprogramme aus.

Diese einzelnen Punkte werden in den folgenden Kapiteln erläutert.

3.2.1 Abbildung der Dialogdatentypen

Ausgehend von der Beschreibung des Dialogs können die dort verwendeten Datentypen auf in COBOL benutzbare Datentypen abgebildet werden. Diese Abbildung vom Dialog nach COBOL ist eindeutig und kann der nachfolgenden unvollständigen Tabelle entnommen werden.

Dialogdatentyp	COBOL-Datentyp
object	PIC 9(9) binary.
integer	PIC 9(9) binary.
string[??]	PIC X(??).
boolean	PIC 9(4) binary.

3.2.2 Zentrale Datenstrukturen

Im COBOL-Interface des Dialog Managers gibt es zwei zentrale Datenstrukturen, über die die Kommunikation zwischen Anwendung und Dialog Manager erfolgt.

Die wichtigste Datenstruktur ist eine Struktur mit Namen **DM-StdArgs**, die von der Anwendung an jede im COBOL-Interface aufgerufene Funktion mit übergeben werden muss. In dieser Struktur teilt der Dialog Manager der Anwendung mit, ob Fehler bei dem Funktionsaufruf aufgetreten sind. Zusätzlich kann die Anwendung dem Dialog Manager Optionen mitteilen, die bei der Ausführung einzelner Funktionen beachtet werden sollen. Die Felder für die Stringbehandlung *DM-truncspaces*, *DM-getsep* und *DM-setsep* sollten nur vor dem Aufruf der Initialisierungsfunktion gesetzt und damit global eingestellt werden, und nicht im Ablauf des Programms verändert werden. Siehe hierzu Kapitel „Behandlung von String-Parametern“.

Damit hat diese DM-StdArgs-Struktur folgendes Aussehen:

```
02 DM-StdArgs.
03 DM-version-type              pic X value "A".
03 DM-major-version             pic 9(4) binary value 6.
03 DM-minor-version             pic 9(4) binary value 3.
03 DM-patch-level               pic 9(4) binary value 2.
03 DM-patch-sublevel            pic 9(4) binary value 0.
03 DM-version-string            pic X(12) value "A.06.03.b".
03 DM-version                   pic 9(4) binary value 603.
03 DM-protocol-version          pic 9(4) binary value 1.
   03 DM-status                 pic 9(9) binary value 0.
   03 DM-options                pic 9(9) binary value 0.
   03 DM-rescode                pic 9(9) binary value 0.
   03 DM-setsep-S.
      04 DM-setsep              pic X value "@".
      04 filler                 pic X value low-value.
   03 DM-getsep-S.
      04 DM-getsep              pic X value space.
```

```
      04 filler                  pic X value low-value.
   03 DM-truncspaces             pic 9(4) binary value 1.
   03 DM-usercodepage-S.
      04 DM-usercodepage         pic X(32) value spaces.
      04 filler                  pic X(32) value low-values.
```

Die zweite wichtige Struktur ist die DM-Value-Struktur, denn in ihr werden die Werte vom COBOL-Programm an den Dialog Manager übergeben. Diese Struktur beinhaltet das Objekt, das Attribut und die Werte, die ein Attribut annehmen kann. In dem Beispiel wird diese Struktur zum Füllen des Tablefields benötigt und wird erst dort beschrieben.

3.2.3 Das Hauptprogramm

Bei Programmen, die mit dem Dialog Manager arbeiten sollen, sind spezielle Hauptprogramme notwendig, die vom Prinzip immer gleich sind und daher aus den mitgelieferten Beispielen kopiert werden können.

Beim Aufbau des Hauptprogramms muss man allerdings unterscheiden, ob man eine lokale Anwendung oder einen Server in einer verteilten Umgebung entwickelt. Bei einer lokalen Anwendung muss eine Funktion mit Namen COBOLMAIN, bei einer verteilten Anwendung müssen zwei Funktionen mit Namen COBOLAPPINIT und COBOLAPPFINISH bereitgestellt werden.

Diese Hauptprogramme müssen in ihrer WORKING-STORAGE-Section die vom Dialog Manager bereitgestellte COPY-Strecke "IDMcobws.cob" verwenden, damit auf die Definitionen des Dialog Manager zugegriffen werden kann.

3.2.3.1 Lokale Anwendungen

Der Aufbau der Funktion COBOLMAIN sieht dabei wie folgt aus:

- Zunächst muss die Funktion DMcob_Initialize zum Initialisieren des Dialog Managers aufgerufen werden. Diese muss unbedingt die erste Funktion im Dialog Manager sein, die aufgerufen wird. Alle anderen Funktionen führen zu Fehlern.
- Nach der Initialisierung des Dialog Managers kann der zu der Anwendung gehörende Dialog mit Hilfe der Funktion DMcob_LoadDialog geladen werden.
- Wenn der Dialog erfolgreich geladen wurde, müssen die im Dialog enthaltenen Funktionen vom COBOL-Programm an den Dialog Manager übergeben werden. Dieses erfolgt über den Aufruf der mit Hilfe des Simulationsprogramms generierten C-Funktion CobRecMInit<Name des Dialogs oder Moduls> für Funktionen, die einen Record als Parameter haben, und durch Aufruf der Funktion BindFuncs für Funktionen, die keine Records als Parameter haben. Diese Funktion BindFuncs kann über das Programm gencobfx generiert werden. Bei dem MICRO FOCUS COBOL-Compiler auf UNIX-Systemen und bei MICRO FOCUS VISUAL COBOL können die Funktionen auch dynamisch über das Laufzeitsystem angezogen und aufgerufen werden. Damit dieses auch bei Aufrufen vom Dialog Manager an die Anwendung genutzt werden kann, muss hierzu die Funktion **DMufcob_BindThruLoader** bzw. **DMmfviscob_BindThruLoader** bei MICRO FOCUS VISUAL

COBOL aufgerufen werden. Dadurch werden intern alle noch nicht mit Funktionspointer versehenen COBOL-Funktionen dynamisch über das COBOL-Laufzeitsystem aufgerufen.

» Nach der Anbindung der Funktionen an den Dialog sollten anwendungsspezifische Initialisierungen durchgeführt werden.

» Nach dem Laden des Dialogs und der Anbindung der Funktionen an den Dialog muss der Dialog für den Anwender mit Hilfe der Funktion DMcob_StartDialog gestartet werden. Beim Aufruf dieser Funktion wird die Startregel im Dialog "on dialog start" ausgeführt und alle im Dialog als sichtbar definierten Fenster sichtbar gemacht.

» Danach wird die Kontrolle an den Dialog Manager übergeben, indem die Funktion DMcob_EventLoop aufgerufen wird. Diese Funktion kehrt normalerweise erst beim Programmende zurück, so dass Anweisungen nach dem Aufruf dieser Funktion erst beim Beenden der Anwendung ausgeführt werden.

Beispiel

```
 identification division.
 program-id. COBOLMAIN.

 data division.
 working-storage section.
*Das ist die Copy-Strecke, die dafür sorgt, dass auf die
*vom Dialog Manager definierten Werte zugegriffen werden
*kann
 copy "IDMcobws.cob".

*Definition einer Variablen zum Speichern der Dialog ID.
 77  DM-dialogid    pic 9(9) binary value 0.

* Variable zum Zwischenspeichern der aktuellen Funktion
* Dadurch wird die Fehlerausgabe erleichtert.
 77  func-name        pic X(30) value spaces.

 linkage section.
* Definition der Parameter, die an das Hauptprogramm
* übergeben werden.
 01  exit-status pic 9(4) binary.

 procedure division using exit-status.
 startup section.

* Initialisierung des Dialog Managers
 initialize-IDM.
* Hier kann noch der Separator gesetzt werden, mit
* dem die Strings in COBOL beendet werden sollen
   move "@" to DM-setsep.
   call "DMcob_Initialize" using DM-StdArgs
```

```
          DM-Common-Data.
     perform error-check.

* Laden des Dialogs, der zu der Anwendung gehört
 load-dialog.
     call "DMcob_LoadDialog" using DM-StdArgs DM-dialogid
            by content "table.dlg@"
     perform error-check.

* Anbinden der Funktionen mit Records als Parameter
 bind-records.
     call "CobRecMInitCobolBeispiel" using DM-dialogid
                 by content 0
                 DM-StdArgs.
* Anbindung der Funktionen ohne Records als Parameter
     call "BindFuncs" using DM-dialogID DM-StdArgs.

* Starten des Dialogs
 start-dialog.
     call "DMcob_StartDialog" using DM-StdArgs DM-dialogid.
     perform error-check.

* Starten der Verarbeitung in dem Dialog
 event-loop.
     call "DMcob_EventLoop" using DM-StdArgs.
     perform error-check.

* Beenden der Anwendung
 dialog-done.
     display "Application finishes successfully.".
     goback.

* Abfragen auf Fehler
 error-check.
     if DM-ErrorOccurred
       display "Error occurred in " func-name upon sysout
       display "Application terminates due to error."
       move 1 to exit-status
       goback.
```

3.2.3.2 Verteilte Anwendungen

Bei verteilten Anwendungen müssen zwei COBOL-Funktionen in der Anwendung bereitgestellt werden, die die Anwendung initialisieren bzw. beenden können. Die Initialisierungsfunktion heißt CobolAppInit, die Endefunktion CobolAppFinish.

Der Aufbau der Funktion CobolAppInit sieht dabei wie folgt aus:

- Zunächst muss die Funktion DMcob_Initialize zum Initialisieren des Dialog Managers aufgerufen werden. Dies ist ein gravierender Unterschied zu in C realisierten verteilten Anwendungen, denn dort darf DM_Initialize nicht aufgerufen werden. Mit dem Aufruf von DMcob_Initialize werden aber wichtige Einstellungen im COBOL-Interface vorgenommen, so dass bei COBOL-Server-Anwendungen auf diesen Aufruf nicht verzichtet werden kann.
- Nach der Initialisierung des COBOL-Interfaces müssen die im Dialog enthaltenen Funktionen vom COBOL-Programm an den Dialog Manager übergeben werden. Dieses erfolgt über den Aufruf der mit Hilfe des Simulationsprogramms generierten C-Funktion CobRecMInit<Name des Dialogs oder Moduls> für Funktionen, die einen Record als Parameter haben, und durch Aufruf der Funktion BindFuncs für Funktionen, die keine Records als Parameter haben. Diese Funktion BindFuncs kann über das Programm gencobfx generiert werden. Bei dem MICRO FOCUS COBOL-Compiler auf UNIX-Systemen und bei MICRO FOCUS VISUAL COBOL können die Funktionen auch dynamisch über das Laufzeitsystem angezogen und aufgerufen werden. Damit dieses auch bei Aufrufen vom Dialog Manager an die Anwendung genutzt werden kann, muss hierzu die Funktion **DMufcob_BindThruLoader** bzw. **DMmfviscob_BindThruLoader** bei MICRO FOCUS VISUAL COBOL aufgerufen werden. Dadurch werden intern alle noch nicht mit Funktionspointer versehenen COBOL-Funktionen dynamisch über das COBOL-Laufzeitsystem aufgerufen. Im Gegensatz zu der nicht verteilten Variante muss hier immer die ApplikationsID als Parameter übergeben werden, zu der die Funktionen gehören.
- Nach der Anbindung der Funktionen an die Applikation sollten anwendungsspezifische Initialisierungen durchgeführt werden.

In der Funktion CobolAppInit müssen dann die Aktionen durchgeführt werden, die ein kontrolliertes Beenden der Anwendung ausführen. Aufrufe an den Dialog Manager sind hierbei nicht notwendig, die Server-Anwendung wird automatisch nach der Rückkehr der Funktion beendet.

Beispiel

```
 identification division.
 program-id. COBOLAPPINIT.

 data division.
 working-storage section.
* Verwenden der vom Dialog Manager bereitgestellten
* Copy Strecke.
 copy "IDMcobws.cob".

 linkage section.
* Deklaration der Parameter der Funktion
* Im Exit-Status wird das Ergebnis zurückgeliefert,
* in DM-appl-id wird die ApplikationsID und in
* DM-dialogID wird die Dialog ID übergeben.
 01  exit-status    pic 9(4) binary.
 77  DM-appl-id    pic 9(9) binary.
```

```
 77  DM-dialogid    pic 9(9) binary.

 procedure division using exit-status dm-appl-id
     DM-dialog-id.
 startup section.

* Initialisierung des COBOL-Interfaces
 initialize-IDM.
   call "DMcob_Initialize" using DM-StdArgs
     DM-Common-Data.
   PERFORM ERROR-CHECK.

* Dynamische Anbindung der COBOL-Funktionen, nachdem die
* Recordfunktionen angebunden worden sind.
 BIND-FUNCTIONS.
   call "CobRecMInitCobolBeispiel" using
               DM-appl-id
               by content 0
               DM-StdArgs.
   call "DMufcob_BindThruLoader" using DM-StdArgs
               DM-appl-id.

* Setzen des Rückgabewertes.
   MOVE DM-SUCCESS TO EXIT-STATUS.
   GOBACK.
```

3.2.4 Hilfsmittel für die COBOL-Programmierung

Damit die im Dialog getroffenen Definitionen für Parameter der COBOL-Funktionen nicht zweimal eingegeben werden müssen, kann der Simulator des Dialog Managers aus einer Dialogdatei die für das COBOL-Programm notwendige Copy-Strecke generieren. Dieses erfolgt durch die Startoption

+writetrampolin <Basis-Name>

Dabei wird mit *Basis-Name* der Name einer Datei angegeben, an die die Endungen ".c", ".cob" und ".cpy" angehängt werden, um die für den Aufruf der COBOL-Funktionen notwendigen Dateien zu generieren.

Für das Beispiel sieht die Kommandozeile wie folgt aus:

```
pidm +writetrampolin tablec table.dlg
```

Durch diesen Aufruf entstehen die Dateien

- tablec.c
- tablec.cpy
- table_.cob

Die generierten C und COBOL-Dateien müssen mit übersetzt und zu der Anwendung hinzu gelinkt werden. Die CPY-Datei sollte benutzt werden, um auf die im Dialog getroffenen Definitionen der Übergabestruktur zugreifen zu können.

Diese Datei hat für das Beispiel folgendes Aussehen:

```
01 RecAddress.
   05 NName      pic X(25).
   05 FirstName  pic X(15).
   05 City       pic X(25).
   05 Street     pic X(30).
```

Hinweis

Bei MICRO FOCUS VISUAL COBOL können Texte auch in Unicode (National Character) formatiert sein. Um eine entsprechende Copy-Strecke zu erzeugen, muss folgende Kommandozeile verwendet werden:

```
pidm -mfviscob-u +writetrampolin tablec table.dlg
```

3.2.5 Die Funktionen für das Tablefield

Funktionen, die den Inhalt von Objekten mit Listencharakter füllen oder abfragen, sind in Bezug auf das Schichtenmodell eine Ausnahme, denn diese Funktionen können nur mit sehr weitreichenden Kenntnissen über den Dialog geschrieben werden. Dies ist notwendig, damit diese Funktionen performant ablaufen können. Das Füllen eines Tablefields muss mit Hilfe von sogenannten temporären Bereichen erfolgen. Diese temporären Bereiche werden im Dialog Manager angelegt, dann von der Anwendung gefüllt und anschließend einem Objekt zugewiesen. Durch diese Vorgehensweise wird bei lokalen Anwendungen das ständige Flackern des Tablefields beim Füllen verhindert und bei verteilten Anwendungen zusätzlich eine sehr hohe Performancesteigerung gegenüber der Einzelzuweisung erreicht. Im einzelnen sieht die Vorgehensweise wie folgt aus:

» Zunächst muss dem Dialog Manager mitgeteilt werden, dass ein temporärer Bereich angelegt werden soll. Dieses erfolgt mit Hilfe der Funktion DMcob_InitVector. Wenn möglich, sollte man hier die gewünschte Größe des Bereiches angeben, also wie viele Elemente man in diesen Bereich stellen möchte. Diese Größe ist aber nur ein Hilfswert für den Dialog Manager, bei Zuweisungen auf höhere Elemente wird der Bereich entsprechend vergrößert. Zusätzlich muss man auch noch den Datentyp angeben, der in dem anzulegenden Bereich gespeichert werden soll. Dabei können die vom Dialog Manager definierten Datentypen wie *DT-string*, *DT-integer* oder *DT-boolean* verwendet werden. Der Datentyp *DT-void* hat dabei die besondere Bedeutung, dass hier ein Bereich angelegt werden soll, der die Attribute *AT-content*, *AT-userdata*, *AT-active* und *AT-sensitive* aufnehmen kann. Dieser Bereich kann dann einem Tablefield oder einer Listbox zugewiesen werden. Diese Funktion liefert als Ergebnis eine ID des von ihr angelegten temporären Bereiches zurück. Diese ID muss an alle nachfolgenden Funktionen übergeben werden, wenn diese auf diesen Bereich zugreifen wollen.

» Nach dem Anlegen eines temporären Bereiches kann dieser Bereich mit Hilfe der Funktion DMcob_SetVectorValue gefüllt werden. Dazu muss in der DM-Value Struktur das Feld DM-Index die Nummer des Elementes enthalten sein, das gerade gesetzt werden soll. In den DM-Value-Teilen der Struktur wird der eigentliche Wert übergeben. In DM-Datatype muss zur Sicherheit der Datentyp des Wertes angegeben werden. Wurde der Bereich mit dem Datentyp *DT-void* angelegt, so muss zusätzlich noch in DM-Attribute das Attribut hinterlegt werden, das gefüllt werden soll.

» Nach dem Füllen des Bereiches kann der Bereich mit Hilfe der Funktion DMcob_SetVector einem Objekt zugewiesen werden. Dabei kann über Parameter gesteuert werden, ob in dem Objekt der neue Inhalt hinten angefügt oder alten Inhalt überschreiben soll.

» Wird der temporäre Bereich nicht mehr benötigt, so muss dieser Bereich unbedingt über die Funktion DMcob_FreeVector freigegeben werden.

Das Auslesen von Werten läuft analog zum Füllen des Objektes:

» Abholen der Werte aus dem Objekt mit Hilfe der Funktion DMcob_GetVector

» Auslesen der einzelnen Werte mit der Funktion DMcob_GetVectorValue.

» Freigeben des temporären Bereiches über die Funktion DMcob_FreeVector.

3.2.5.1 Funktion FILLTAB

Diese Funktion übernimmt das initiale Füllen des Tablefields. Dazu wird dieser Funktion die ID des Tablefields und die Anzahl der zu füllenden Elemente übergeben.

Zunächst erfolgt in üblicherweise die Definition des Moduls (sollte genauso heißen wie die Funktion im Dialog).

```
*SET OSVS
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILLTAB.
    AUTHOR. "ISA-DEMO".

    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    DATA DIVISION.
    FILE SECTION.

    WORKING-STORAGE SECTION.
    01  STR-TAB.
    05 STRFIELD PIC X OCCURS 80.
    01  INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.

    77  COUNTER PIC 9(4) VALUE 0.
    77  BASE PIC 9(4) VALUE 0.
    77  DM-POINTER  PIC 9(4) BINARY VALUE 0.
    77  ICOUNT  PIC 9(4) BINARY VALUE 0.
```

```
 77  IDUMMY  PIC 9(4) VALUE 0.

 77  I    PIC 99 VALUE ZERO.
 77  J    PIC 99 VALUE ZERO.
 77  KL      PIC 9(4) VALUE ZERO.
 77  DLG-NAME PIC X(80) value "Name ".
 77  DLG-VORNAME PIC X(80) value "Vorname ".
 77  DLG-WOHNORT PIC X(80) value "Ort  ".
```

In der Linkage-Section wird ein Copy auf die vom Dialog Manager bereitgestellte Datei IDMcobls.cob durchgeführt, damit auf diese Definitionen im COBOL-Programm zugegriffen werden kann.

```
 LINKAGE SECTION.
*Über diese Copy-Strecke werden die Definitionen im
* Dialog Manager
*in diesem COBOL Modul verfügbar.
 COPY "IDMcobls.cob".
*In diesem Parameter wird die Anzahl der Zeilen
*Übergeben, die
*initial gefüllt werden sollen.
 77  DLG-CNT PIC 9(9) binary.
*In dieser Variablen wird die Tabelle übergeben
*die gefüllt werden soll.
 77  DLG-OBJECT PIC 9(9) binary.
```

Bei der Definition der Procedure Division muss man beachten, dass diese einen Parameter mehr als im Dialog definiert hat. Der erste Parameter einer vom Dialog Manager aufgerufenen COBOL-Funktion ist immer die DM-Common-Data und diese wird im Dialog nicht definiert.

```
*Diese COBOL-Funktion legt einen temporaeren Speicherbereich
* im Dialog Manager an und weist diesen dann einem
* Tablefield zu.

 PROCEDURE DIVISION USING DM-COMMON-DATA
     DLG-OBJECT DLG-CNT.
 ORGANIZE-IN SECTION.
   MOVE 0 TO BASE.
   MOVE DLG-CNT TO ICOUNT.
*Initialisierung eines Speicherplatzes im DM
```

In dieser Funktion wird dann ein temporärer Bereich angelegt, gefüllt und dann dem Tablefield zugewiesen.

```
    CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
      DT-String
      ICOUNT.
*Initialisierung der DM-Value Struktur
   MOVE DT-STRING TO DM-DATATYPE.
*Auffüllen des temporären Bereichs, der dann in der
*Tabelle angezeigt werden soll.
```

```
 PREPARE-DATA SECTION.
*Setzen der einzelnen Inhalte
   MOVE 0 TO COUNTER.
   MOVE 0 TO KL.

   PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL COUNTER > ICOUNT
         ADD 1 TO KL
     MOVE KL TO DM-INDEX
     MOVE DLG-NAME TO STR-TAB
*Zunächst wird der Name gesetzt.
         COMPUTE IDUMMY = COUNTER + BASE
     MOVE IDUMMY TO INT-TAB
     MOVE 5 TO J
     MOVE SPACE TO STRFIELD(J)
     ADD 1 TO J
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
         MOVE INTFIELD(I) TO STRFIELD(J)
         ADD 1 TO J
     END-PERFORM

     MOVE STR-TAB TO DM-VALUE-STRING
     CALL "DMcob_SetVectorValue" USING DM-STDARGS
         DM-VALUE
         DM-POINTER

*Danach wird der Vorname gesetzt.
         ADD 1 TO KL
     MOVE KL TO DM-INDEX
     MOVE DLG-VORNAME TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
     COMPUTE IDUMMY = COUNTER + BASE
     MOVE IDUMMY TO INT-TAB
     MOVE 8 TO J
     MOVE SPACE TO STRFIELD(J)
     ADD 1 TO J
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
         MOVE INTFIELD(I) TO STRFIELD(J)
         ADD 1 TO J
     END-PERFORM

*Zuletzt wir der Wohnort gesetzt.
     MOVE STR-TAB TO DM-VALUE-STRING
     CALL "DMcob_SetVectorValue" USING DM-STDARGS
         DM-VALUE
         DM-POINTER
```

```
           ADD 1 TO KL
           MOVE KL TO DM-INDEX
           MOVE DLG-WOHNORT TO STR-TAB
       *Aufbereiten eines veraenderten Strings fuer die Anzeige
           COMPUTE IDUMMY = COUNTER + BASE
           MOVE IDUMMY TO INT-TAB
           MOVE 4 TO J
           MOVE SPACE TO STRFIELD(J)
           ADD 1 TO J
           PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
               MOVE INTFIELD(I) TO STRFIELD(J)
               ADD 1 TO J
           END-PERFORM

           MOVE STR-TAB TO DM-VALUE-STRING
           CALL "DMcob_SetVectorValue" USING DM-STDARGS
               DM-VALUE
               DM-POINTER
       *Ende der Schleife zum Aufbereiten der Daten.
          END-PERFORM.

       *Uebergeben der gespeicherten Werte an die Anzeige
          MOVE DLG-OBJECT TO DM-OBJECT.
          MOVE AT-FIELD TO DM-ATTRIBUTE.
          MOVE 2 TO DM-INDEXCOUNT.
          MOVE 1 TO DM-index.
          MOVE 1 TO DM-second.
       * Es soll der gesamte Inhalt der Tabelle durch den
       * Inhalt des temporären Bereichs ersetzt werden.
       * Daher die letzten drei Parameter mit dem Wert 0.
          CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
                DM-POINTER
                by content 0
                by content 0
                by content 0.
```

Abschließend wird dieser Bereich wieder im Dialog Manager freigegeben.

```
       *Freigeben des Speicherbereiches
          CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.

          GOBACK.
```

3.2.5.2 Funktion TABFUNC

Mit Hilfe dieser Funktion soll das Nachladen des Tablefields erreicht werden. Immer wenn der Benutzer in einen Bereich scrollt, der noch nicht gefüllt ist, wird vom Dialog Manager diese Funktion aufgerufen.

Die Parameter sind fest vom Dialog Manager definiert und können daher nicht geändert werden. Diese Nachladefunktionen erhalten eine Struktur DM-Content-Data als Parameter, in der die notwendigen Informationen gespeichert sind.

```
01  DM-Content-Data
    02  DM-CO-OBJECT     pic 9(9) binary.
    02  DM-CO-REASON     pic 9(4) binary.
    02  DM-CO-VISFIRST   pic 9(4) binary.
    02  DM-CO-VISLAST    pic 9(4) binary.
    02  DM-CO-LOADFIRST  pic 9(4) binary.
    02  DM-CO-LOADLAST   pic 9(4) binary.
    02  DM-CO-COUNT      pic 9(4) binary.
    02  DM-CO-HEADER     pic 9(4) binary.
```

In DM-Co-Object wird die ID des Tablefields übergeben. Die Felder DM-Co-Visfirst und DM-Co-Vislast enthalten die erste und letzte sichtbare Zeile, die Felder DM-Co-Loadfirst und DM-Co-Loadlast die erste und letzte zu ladende Zeile. Dieses sind jedoch nur die Minimalwerte, die Anwendung darf jederzeit mehr als die angegebenen Zeilen laden.

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. TABFUNC.
    AUTHOR. "ISA-DEMO".

    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    DATA DIVISION.
    FILE SECTION.

    WORKING-STORAGE SECTION.
    01  STR-TAB.
    05 STRFIELD PIC X OCCURS 80.
    01  INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.
    77  COUNTER PIC 9(4) VALUE 0.
    77  BASE PIC 9(4) VALUE 0.
    77  DM-POINTER  PIC 9(4) BINARY VALUE 0.
    77  ICOUNT  PIC 9(4) BINARY VALUE 0.
    77  IROW    PIC 9(4) BINARY VALUE 0.
    77  ICOL    PIC 9(4) BINARY VALUE 3.
    77  IDUMMY  PIC 9(4) VALUE 0.

    77  I    PIC 99 VALUE ZERO.
```

```
77  J    PIC 99 VALUE ZERO.
77  KL      PIC 9(4) VALUE ZERO.
77  DLG-NAME PIC X(80) value "Name ".
77  DLG-VORNAME PIC X(80) value "Vorname ".
77  DLG-WOHNORT PIC X(80) value "Ort  ".
77  DLG-COUNT PIC 9(9) BINARY value 0.
```

Damit auf die DM-Content-Data Struktur zugegriffen werden kann, muss zusätzlich die Datei IDM-coboc.cob per COPY angezogen werden. Ansonsten läuft diese Funktion wie die Funktion FILLTAB ab.

```
LINKAGE SECTION.
COPY "IDMcobls.cob".
COPY "IDMcoboc.cob".

* Diese COBOL-Funktion legt einen temporaeren Speicher
* bereich im Dialog Manager an und weist diesen dann
* einem Tablefield zu.

PROCEDURE DIVISION USING DM-COMMON-DATA DM-Content-Data.
ORGANIZE-IN SECTION.

  COMPUTE ICOUNT =  DM-co-loadlast - DM-co-loadfirst.
  COMPUTE ICOUNT = ICOUNT + 1.
  COMPUTE ICOUNT = ICOUNT* 3.
  COMPUTE BASE = DM-co-loadfirst* 3.
  MOVE ICOUNT TO DLG-COUNT.
  MOVE DM-CO-OBJECT TO DM-OBJECT.
  MOVE DT-string TO DM-DATATYPE.
  MOVE DM-co-loadfirst TO BASE.
*Initialisierung eines Speicherplatzes im DM
  CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
    DT-String
    ICOUNT.
*Initialisierung der DM-Value Struktur
  MOVE DT-STRING TO DM-DATATYPE.
  PERFORM FILLDATA.

*Uebergeben der gespeicherten Werte an die Anzeige
  MOVE DM-CO-OBJECT TO DM-OBJECT.
  MOVE AT-FIELD TO DM-ATTRIBUTE.
  MOVE 2 TO DM-INDEXCOUNT.
  COMPUTE DM-INDEX =  DM-co-loadfirst - 1.
  MOVE 1 TO DM-second.
  CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
        DM-POINTER by content 0
        by reference DM-co-loadlast ICOL.
```

```
*Freigeben des Speicherbereiches
   CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
   GOBACK.

 FILLDATA.
*Setzen der einzelnen Inhalte
   MOVE 0 TO COUNTER.
   MOVE 0 TO KL.
     MOVE 256 TO DM-INDEXCOUNT.
   PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL KL > ICOUNT
     ADD 1 TO KL
     MOVE KL TO DM-INDEX
     MOVE DLG-NAME TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
     COMPUTE IDUMMY = COUNTER + BASE
     MOVE IDUMMY TO INT-TAB
     MOVE 5 TO J
     MOVE SPACE TO STRFIELD(J)
     ADD 1 TO J
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
         MOVE INTFIELD(I) TO STRFIELD(J)
         ADD 1 TO J
     END-PERFORM

     MOVE STR-TAB TO DM-VALUE-STRING
     CALL "DMcob_SetVectorValue" USING DM-STDARGS
         DM-VALUE DM-POINTER

     ADD 1 TO KL
     MOVE KL TO DM-INDEX
     MOVE DLG-VORNAME TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
     COMPUTE IDUMMY = COUNTER + BASE
     MOVE IDUMMY TO INT-TAB
     MOVE 8 TO J
     MOVE SPACE TO STRFIELD(J)
     ADD 1 TO J
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
         MOVE INTFIELD(I) TO STRFIELD(J)
         ADD 1 TO J
     END-PERFORM

     MOVE STR-TAB TO DM-VALUE-STRING
     CALL "DMcob_SetVectorValue" USING DM-STDARGS
```

```
           DM-VALUE DM-POINTER
        ADD 1 TO KL
        MOVE KL TO DM-INDEX
        MOVE DLG-WOHNORT TO STR-TAB
    *Aufbereiten eines veraenderten Strings fuer die Anzeige
        COMPUTE IDUMMY = COUNTER + BASE
        MOVE IDUMMY TO INT-TAB
        MOVE 4 TO J
        MOVE SPACE TO STRFIELD(J)
        ADD 1 TO J
        PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
            MOVE INTFIELD(I) TO STRFIELD(J)
            ADD 1 TO J
        END-PERFORM
        MOVE STR-TAB TO DM-VALUE-STRING
        CALL "DMcob_SetVectorValue" USING DM-STDARGS
            DM-VALUE DM-POINTER
      END-PERFORM.
```

3.2.6 Funktionen mit Records als Parameter

Im Gegensatz zu den Funktionen für das Tablefield können die nachfolgenden Funktionen ohne Wissen über den Dialog auskommen und sollten das auch. Diese Funktionen werden daher ohne jeglichen Aufruf an den Dialog Manager in Standard-COBOL geschrieben. Lediglich die Copy-Strecken zeigen, dass es sich um Funktionen handelt, die vom Dialog Manager aufgerufen werden.

3.2.6.1 Funktion GETADDR

Diese Funktion soll zu einem gegebenen Namen den restlichen Datensatz an die Oberfläche liefern. Damit auf die Definitionen des Dialog zugegriffen werden kann, wird ein Copy auf die vom Dialog Manager generierte Datei tablec.cpy eingefügt.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. GETADDR.
 AUTHOR. "ISA-DEMO".
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.

 COPY "IDMcobls.cob".
*Das nachfolgende Copy gehört zu dieser Anwendung. Es
*wurde vom DM aus dem Dialog generiert und sollte in den
*COBOL-Modulen da verwendet werden, wo auf die entsprechenden
*Strukturen zugegriffen werden soll.
 COPY "tablec.cpy".
```

```
 PROCEDURE DIVISION USING DM-COMMON-DATA RECADDRESS.

 SEC-RECADDRESS SECTION.
*Hier müssten jetzt die Werte für die einzelnen Einträge
*aus der Datenbank geholt werden.
*Das entfällt hier. Stattdessen werden hier Dummy-Werte
*in die entsprechenden Strukurelemente zugewiesen.
     MOVE "NAME" TO NNAME.
     MOVE "VORNAME" TO FIRSTNAME.
     MOVE "STRASSE" TO STREET.
     MOVE 3 TO COUNTRY.
     GOBACK.
```

3.2.6.2 Funktion PUTADDR

Diese Funktion soll die vom Benutzer geänderten Daten wieder abspeichern. Damit auf die Definitionen des Dialogs zugegriffen werden kann, wird ein Copy auf die vom Dialog Manager generierte Datei tablec.cpy eingefügt.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PUTADDR.
 AUTHOR. "ISA-DEMO".

 DATA DIVISION.

 WORKING-STORAGE SECTION.

 LINKAGE SECTION.

 COPY "IDMcobls.cob".
*Das nachfolgende Copy gehört zu dieser Anwendung. Es
*wurde vom DM aus dem Dialog generiert und sollte in den
*COBOL-Modulen da verwendet werden, wo auf die entsprechenden
*Strukturen zugegriffen werden soll.
 COPY "tablec.cpy".

 PROCEDURE DIVISION USING DM-COMMON-DATA RECADDRESS.

 SEC-RECADDRESS SECTION.

*Hier müssten jetzt die Werte für die einzelnen Einträge
*in der Datenbank gespeichert werden.
*Das entfällt hier. Stattdessen werden hier Dummy-Werte
*in die entsprechenden Strukurelemente zugewiesen.
     MOVE "      " TO STREET.
     MOVE 0 TO COUNTRY.
```

```
GOBACK.
```

3.3 Übersetzen und Linken

Abschließend müssen nun die erstellten Sourcen übersetzt werden. Die einzelnen Compiler-Optionen sind dabei umgebungsabhängig und können hier nur beispielhaft angegeben werden. Es sollten die in den Beispielen ausgelieferten Makefiles genommen werden und an die örtlichen Gegebenheiten angepasst werden.

Prinzipiell sind zum Bauen einer ablauffähigen Anwendung folgende Schritte notwendig:

1. Mit Hilfe des Simulationsprogramms des Dialog Managers werden die notwendigen Copy-Strecken generiert.
   ```
   idm +writetrampolin tablec table.dlg
   ```
2. Die generierte C-Datei wird mit Hilfe des C-Compilers und den für den Dialog Manager üblichen Compiler-Optionen übersetzt. Zusätzlich muss in einer Optionen angegeben werden, für welches COBOL-System die Datei übersetzt werden soll. Dabei gibt es die Möglichkeiten
   - -DUFCOB für MICRO FOCUS COBOL
   - -DMFVISCOB für MICRO FOCUS VISUAL COBOL

   Zusätzlich muss hier die Option -DUFCOB_LINK auf Microsoft Windows gesetzt werden. Auf den UNIX-Systemen und bei MICRO FOCUS VISUAL COBOL kann dieser Schalter gesetzt werden, wenn ein ausführbares Programm, das ohne das COBOL-Laufzeitsystem ablaufen kann, zusammen gelinkt werden soll.
   ```
   cc -c -DMOTIF -DHP9800 -DHPUX -DUFCOB table_.cob
   ```
3. Die COBOL-Module müssen ganz normal mit dem COBOL-Compiler übersetzt werden.
   ```
   cob -x -c FILLTAB.cbl
   ```
4. Anschließend werden die übersetzten Dateien zu einem Executable zusammen gelinkt. Auf den UNIX-basierten Systemen erfolgt dies mit Hilfe des Shell-Skripts **ufdmlink** für MICRO FOCUS-Programme. Auf den anderen Systemen wird der Linker direkt aufgerufen.

Auf Windows-Systemen steht das Skript **idmcob.bat** zum Kompilieren und Linken zur Verfügung.

Im Gegensatz zu reinen C-Anwendungen darf die Datei "startup.obj" nicht zu dem Programm hinzugebunden werden. Dafür müssen aber die speziell für das COBOL zuständigen Dateien und Libraries zu der Anwendung dazu gelinkt werden. Beim Zusammenlinken muss die angegebene Reihenfolge unbedingt eingehalten werden.

4 Datentypen

Damit die im Dialog definierten COBOL-Funktionen realisiert werden können, muss zunächst eine Abbildungsvorschrift definiert werden, wie die im Dialog definierten Datentypen auf die in COBOL verfügbaren Datentypen abgebildet werden können. Danach werden die zur Kommunikation zwischen Dialog Manager und Applikation definierten Strukturen erklärt, die durch das Kopieren der Datei **IDMcobws.cob** verfügbar werden. Sie dienen dabei sowohl der Übergabe von Werten an den DM als auch der Rückgabe von Werten des DM an die Applikation.

4.1 Abbildung der Dialog-Datentypen auf COBOL-Datentypen

4.1.1 Grunddatentypen

Die im Dialog verwendeten Grunddatentypen werden in COBOL wie folgt abgebildet:

DM-Datentyp	COBOL-Datentyp
boolean	pic 9(4) binary
cardinal	pic 9(4) binary
enum	pic 9(4) binary
datatype	pic 9(4) binary
index	keine Entsprechung
object	pic 9(9) binary
string [?]	pic X(?) COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL: pic N(?) national
integer	pic 9(9) binary
attribute	pic 9(9) binary
method	pic 9(9) binary
class	pic XX
anyvalue	keine Entsprechung COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL: pointer

4.1.2 Records

Neben den Grunddatentypen werden auch Records zu COBOL-Datentypen konvertiert. Die Konvertierung erfolgt hierbei einzeln nach dem obigen Schema für jedes einzelne Element.

Die für das COBOL-Programm notwendigen Definitionen werden durch eine generierte COBOL Copy-Datei bereitgestellt. Diese Generierung erfolgt mit Hilfe des Simulationsprogramms und der Startoption **+/-writetrampolin**.

```
idm +writetrampolin <BasisName> <Name der Dialogdatei>
```

Beispiel

In einem Record sind ein String und eine Zahl enthalten.

```
record Test1
{
  string[25] Value1;
  integer    Value3;
}
```

Die entsprechende generierte COBOL-Definition sieht dann wie folgt aus:

```
01  Test1
    05 Value1 pic X(25).
    05 Value3 pic 9(9) binary.
```

Hinweis

Um mit MICRO FOCUS VISUAL COBOL den Datentyp *National Character* zu unterstützen, muss zusätzlich die Option **-mfviscob-u** angegeben werden. Die COBOL-Definition sieht dann so aus:

```
01  RecTest1.
    05  Value1-S.
        06 Value1 pic X(25).
        06 filler pic X(25).
    05  filler redfines Value1-S.
        06 Value1-u pic N(25) national.
    05  Value3 pic S9(9) binary.
```

4.1.3 Datentypen für Sammlungen

Die Sammlungsdatentypen des IDM und die damit zusammenhängenden Funktionen für gemanagte (verwaltete) IDM-Werte (**Managed DM-Values**) werden über den Zeiger-Datentyp (POINTER) von MICRO FOCUS VISUAL COBOL unterstützt.

DM-Datentyp	Visual-COBOL-Datentyp
hash	POINTER
list	POINTER
matrix	POINTER
refvec	POINTER
vector	POINTER

Ein **Managed DM-Value** wird als Zeiger an MICRO FOCUS VISUAL COBOL übergeben:

```
01 ManagedValue pointer.
ENTRY "GetAnyValue" using DM-COMMON-DATA ManagedValue.
```

Um einen solchen Wert innerhalb der **DM-Value**-Struktur zu verwenden, wird der Wert nach *DM-value-pointer* kopiert und *DM-datatype* auf *DT-anyvalue* gesetzt.

```
MOVE DT-anyvalue TO DM-datatype.
MOVE ManagedValue To DM-value-pointer.
```

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

4.2 Standardargument

Das Dialog Manager Standardargument **DM-StdArgs** wird an alle Funktionen übergeben. Es wird benutzt, um den Rückgabewert der aufgerufenen Funktion zu prüfen und um globale Information zu setzen.

```
02 DM-StdArgs.
03 DM-version-type              pic X value "A".
03 DM-major-version             pic 9(4) binary value 6.
03 DM-minor-version             pic 9(4) binary value 3.
03 DM-patch-level               pic 9(4) binary value 2.
03 DM-patch-sublevel            pic 9(4) binary value 0.
03 DM-version-string            pic X(12) value "A.06.03.b".
03 DM-version                   pic 9(4) binary value 603.
03 DM-protocol-version          pic 9(4) binary value 1.
   03 DM-status                 pic 9(9) binary value 0.
   03 DM-options                pic 9(9) binary value 0.
   03 DM-rescode                pic 9(9) binary value 0.
   03 DM-setsep-S.
      04 DM-setsep              pic X value "@".
      04 filler                 pic X value low-value.
```

```
   03 DM-getsep-S.
      04 DM-getsep               pic X value space.
      04 filler                  pic X value low-value.
   03 DM-truncspaces             pic 9(4) binary value 1.
   03 DM-usercodepage-S.
      04 DM-usercodepage         pic X(32) value spaces.
      04 filler                  pic X(32) value low-values.
```

Definition in der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

```
02 DM-StdArgs.
   03 DM-version-type            pic X value "T".
   03 DM-major-version           pic 9(4) binary value 6.
   03 DM-minor-version           pic 9(4) binary value 3.
   03 DM-patch-level             pic 9(4) binary value 2.
   03 DM-patch-sublevel          pic 9(4) binary value 0.
   03 DM-version-string          pic X(12) value "A.06.03.b".
   03 DM-version                 pic 9(4) binary value 603.
   03 DM-protocol-version        pic 9(4) binary value 1.
   03 DM-status                  pic 9(9) binary value 0.
   03 DM-options                 pic 9(9) binary value 0.
   03 DM-rescode                 pic 9(9) binary value 0.
   03 DM-setsep-S.
      04 DM-setsep               pic X value "@".
      04 filler                  pic X value low-value.
   03 filler redefines DM-setsep-S.
      04 DM-setsep-u             pic N national.
   03 DM-getsep-S.
      04 DM-getsep               pic X value space.
      04 filler                  pic X value low-value.
   03 filler redefines DM-getsep-S.
      04 DM-getsep-u             pic N national.
   03 DM-truncspaces             pic 9(4) binary value 1.
   03 DM-usercodepage-S.
      04 DM-usercodepage         pic X(32) value spaces.
      04 filler                  pic X(32) value low-values.
   03 filler redefines DM-usercodepage-S.
      04 DM-usercodepage-u       pic N(32) national.
```

DM-status

Beinhaltet den Fehlerstatus des Aufrufs an die Dialog Manager Funktion. Mögliche Werte sind:

- *DM-success*
  Funktion hat keinen Fehler gefunden.
- *DM-error*
  Funktion hat Fehler gefunden.

Alle Dialog Manager Funktionen setzen dieses Feld, um sagen zu können, ob der Aufruf erfolgreich gewesen ist oder nicht.

DM-options

In diesem Optionsfeld kann funktionsspezifische Information gesetzt werden. Dieses Feld wird von der Dialog Manager Funktion immer auf 0 zurückgesetzt. Die möglichen Werte werden bei den Funktionen beschrieben, da die gültigen Werte funktionsspezifisch sind.

DM-rescode

Dieses Feld enthält den Rückgabe-Code einiger Dialog Manager Funktionen.

Die nächsten drei Elemente der Struktur gehören von der Bedeutung her zusammen und sind alle durch den Einsatz zweier unterschiedlicher Programmiersprachen bedingt. In der Anwendung wird hier mit COBOL programmiert, das Strings bei der Definition in ihrer Länge festlegt; der Dialog Manager hingegen betrachtet Strings als dynamisch, passt also die Länge immer den aktuellen Gegebenheiten an.

Aus diesem Grund muss aber bei der Definition von Strings im Dialog immer eine Länge angegeben werden, wenn dieser String an ein COBOL-Programm übergeben werden soll. Dieses Zusammenspiel wird im nachfolgenden Kapitel genauer dargestellt. Prinzipiell sollten diese drei Elemente nur vor der Initialisierung des Dialog Manager gesetzt und danach nicht mehr verändert werden, sonst kann es zu Schwierigkeiten in den verschiedenen COBOL-Programmen kommen.

DM-setsep
DM-setsep-u

Das Zeichen, das in diesem Feld gespeichert wird, sagt dem Dialog Manager, dass die Applikation Strings mit diesem Zeichen beendet.

DM-getsep
DM-getsep-u

Das Zeichen, das in diesem Feld gespeichert wird, sagt dem Dialog Manager, Strings mit diesem Zeichen zu beenden.

DM-truncspaces

Wenn dieses Feld auf 1 gesetzt ist, entfernt der Dialog Manager alle Leerzeichen am Ende einer Zeichenkette.

DM-usercodepage
DM-usercodepage-u

In diesem Feld wird die Codepage für Zeichenketten beim Aufruf der Funktion DMcob_Control definiert.

Hinweis

Vor IDM-Version A.06.01.d war die Struktur **DM-StdArgs** wie folgt definiert:

```
02 DM-StdArgs.
   03 DM-status              pic 9(9) binary value 0.
   03 DM-options             pic 9(4) binary value 0.
   03 DM-rescode             pic 9(4) binary value 0.
   03 DM-setsep              pic X value "@".
   03 DM-getsep              pic X value space.
   03 DM-truncspaces         pic 9(4) binary value 1.
   03 DM-usercodepage        pic X(32) value space.
```

4.3 Behandlung von String-Parametern

Bei der Bearbeitung von String-Parametern im COBOL-Interface muss beachtet werden, dass der Dialog Manager intern in der Programmiersprache C geschrieben ist. Diese beiden Programmiersprachen haben leider eine sehr unterschiedliche interne Darstellung von Strings.

	C	COBOL
Länge	dynamisch, beliebige Länge	statisch definierte Länge
Endekennzeichen	Endekennzeichen '\0'	kein Endekennzeichen
Füllgrad	Länge entspricht in der Regel dem eigentlichen String	Länge unterschiedlich zu Definition. In der Regel mit Blanks aufgefüllt.

Um nun diese zwei unterschiedlichen Darstellungen von Strings möglichst nahe zusammen zu führen, gibt es im Dialog Manager folgende Vorgehensweise bei der Behandlung von String-Parametern:

1. Beim Initialisierungsaufruf an den Dialog Manager durch DMcob_Initialize können dem Dialog Manager drei Werte mitgegeben werden, die die Stringbehandlung steuern.
   a. `DM-SetSep of DM-StdArgs`
      Das hier enthaltene Zeichen ist das Zeichen, mit dem das COBOL-Programm seine wirklichen Strings terminiert. Der Dialog Manager sucht nach diesem Zeichen und löscht alle nachfolgenden Zeichen in dem übergebenen String.
   b. `DM-GetSep of DM-StdArgs`
      Das hier enthaltene Zeichen ist das Zeichen, mit dem der Dialog Manager Strings bis zu ihrer Definitionslänge auffüllt, bevor sie an das COBOL-Programm übergeben werden.
   c. `DM-truncspaces of DM-StdArgs`
      Über diesen Parameter kann dem Dialog Manager mitgeteilt werden, dass alle Blanks am Ende eines Strings als nicht existent betrachtet und gelöscht werden sollen.

   Diese bei DMcob_Initialize getroffenen Einstellungen gelten für alle direkt vom Dialog Manager aufgerufenen Funktionen sowie deren Rückgabewerte.
2. Bei allen Aufrufen an die Schnittstellenfunktion des Dialog Managers können diese Werte lokal temporär umdefiniert werden. Es gibt dabei wie bei 1) drei Möglichkeiten:

a. `DM-setsep of DM-StdArgs`

b. `DM-setsep of DM-StdArgs`

c. `DM-truncspaces of DM-StdArgs`

Die Bedeutung ist wie bei 1.

Daher erscheint folgende Belegung sinnvoll:

```
DM-setsep @
DM-getsep " "
DM-truncspaces 1
```

Mit dieser Belegung können die jeweiligen Werte jeweils direkt vom COBOL-Programm und im Dialog Manager verarbeitet werden.

Beispiel

Im Dialog ist ein Eingabefeld definiert mit einer Länge, die durch das Attribut *.maxchars* auf die Länge 20 begrenzt ist. Die COBOL-Funktion, die diese Eingabe überprüfen soll, erhält einen String-Parameter der Länge 20.

```
function cobol void TestFkt (string[20] InString input output);

edittext EtEingabe
{
  .maxchars 20;
}
```

Die nachfolgende Tabelle stellt den Einfluss der einzelnen Elemente auf die Übergabe des Strings von und nach COBOL dar.

Dialog	COBOL	
	DM-GetSep Space	**Ergebnis**
AB im Eingabefeld	AB wird mit 18 Leerzeichen aufgefüllt	AB + 18 Leerzeichen

Im COBOL-Programm wird nun dieser String verarbeitet und ein neuer Inhalt in den String gesetzt.

COBOL	Dialog		
Befehl	**Inhalt des Strings**	**DM-TruncSpaces 1**	**Ergebnis**
`MOVE "NEU" TO InString`	NEU + 17 Leerzeichen	NEU	NEU

4.4 Verwendung des Datentyps *anyvalue*

Wird im IDM-Dialogskript der Typ *anyvalue* als Parameter einer COBOL-Funktion definiert, dann ist für diesen Parameter jeder Datentyp erlaubt. Im COBOL-Programm wird ein solcher Parameter als Zeiger (*pointer*) an das Programm übergeben. Um einen solchen Wert in Schnittstellenfunktionen zu verwenden, die eine **DM-Value**-Struktur erwarten, kann der Parameter dem *DM-value-pointer* zugewiesen werden und der *DM-datatype* auf *DT-anyvalue* gesetzt werden. Man kann dies so interpretieren, dass die **DM-Value**-Struktur jeden Datentyp beinhalten kann und dass der tatsächliche Wert als Zeiger hinterlegt ist.

Unabhängig davon, wie ein Eingabeparameter definiert war, wird ein Rückgabewert oder Ausgabeparameter, dessen Datentyp direkt in der **DM-Value**-Struktur gespeichert werden kann, auch direkt gespeichert. Aus *DT-anyvalue* wird also zum Beispiel *DT-string*.

Nicht direkt gespeichert werden können Sammlungsdatentypen, diese werden also immer als *DT-anyvalue* zurückgegeben. Ein solcher *DT-anyvalue*-Wert kann mit den **DMcob_Value***-Funktionen ausgewertet werden.

Zu beachten

Der Datentyp *anyvalue* gibt im IDM-Dialogskript an, dass jeder Datentyp erlaubt ist. Zur Laufzeit haben aber ein Attribut, eine Variable, ein Parameter oder ein Rückgabewert immer einen Wert mit einem bestimmten Datentyp, der ungleich *anyvalue* ist.

Beispiel

Die folgende Definition einer Regel besagt, dass der Rückgabewert jeden Datentyp annehmen kann:

```
rule anyvalue RuleAny(integer I) {
  case I
    in 1: return "Hallo";
    in 2: return 42;
    otherwise:
      return void;
  endcase
}
```

Zur Laufzeit wird aber nicht *anyvalue* zurückgegeben, sondern immer der Typ aus der `return`-Anweisung.Zum Beispiel wird `print typeof(RuleAny(1));` den Wert *„string“* ausgeben.

Eine Ausnahme von den obigen Regeln stellt ein Managed Value dar, der mit **DMcob_ValueInit** erzeugt wurde. Ein Managed Value bleibt immer erhalten. Wird ein Managed Value in einer **DM-Value** Struktur als *DM-value-pointer* verwendet (*DM-datatype* muss auf *DT-anyvalue* gesetzt sein), dann bleibt der Datentyp *DT-anyvalue* erhalten. Dies gilt auch für Rückgabewerte und Ausgabeparameter.

Gültigkeitsdauer der DM-value-pointer-Werte

Ein Managed Value, der nicht global angelegt wurde, ist nur bis zum Ende der Funktion, in der er angelegt wurde gültig. Er kann aber noch an den IDM zurückgegeben werden.

Das gleiche gilt für Zeigerwerte, die als *anyvalue*-Parameter an eine Funktion übergeben wurden. Sie sind auch nur bis zum Ende der Funktion gültig. Eine Rückgabe an den IDM ist erlaubt.

Alle anderen Zeigerwerte, die zum Beispiel angelegt wurden um einen Sammlungsdatentyp zurückzugeben, sind nur bis zum Aufruf der nächsten **DMcob***-Funktion gültig. Sie dürfen auch nicht an den IDM zurückgegeben werden. Damit solche Werte ausgewertet werden können, zerstören die **DMcob_Value***-Funktionen die Gültigkeit nicht.

Es wird empfohlen für Sammlungsdatentypen von vorneherein Managed Values zu verwenden.

4.5 Erfragen und Ändern von Attributen

Zum Setzen und Abfragen von Attributwerten im Dialog Manager werden drei unterschiedliche Datenstrukturen verwendet. In der Struktur **DM-Value** kann genau ein Wert erfragt bzw. gesetzt werden, in der Struktur **DM-ValueArray** hingegen können bis zu 16 Werte mit einem Aufruf erfragt bzw. gesetzt werden. Die Struktur **DM-ValueIndex** dient zur Übergabe eines zusätzlichen Indexwertes.

Diese Strukturen sind in **IDMcobls.cob** und **IDMcobws.cob** definiert.

4.5.1 DM-Value

Mit Hilfe dieser Struktur kann immer genau ein Wert im Dialog Manager gesetzt oder erfragt werden. Diese Struktur ist wie folgt definiert:

```
02 DM-Value.
   03 DM-object                 pic 9(9) binary value 0.
   03 DM-attribute              pic 9(9) binary value 0.
   03 DM-indexcount             pic 9(4) binary value 0.
   03 DM-index                  pic 9(4) binary value 0.
   03 DM-second                 pic 9(4) binary value 0.
   03 DM-datatype               pic 9(4) binary value 0.
   03 DM-long-first             pic 9(9) binary value 0.
   03 DM-long-second            pic 9(9) binary value 0.
   03 DM-value-object           pic 9(9) binary value 0.
   03 DM-value-boolean          pic 9(4) binary value 0.
   03 DM-value-cardinal         pic 9(4) binary value 0.
   03 DM-value-classid          pic XX value "??".
   03 filler                    pic XX value low-values.
   03 DM-value-integer          pic S9(9) binary value 0.
   03 DM-value-index.
      04 DM-value-long-first    pic 9(9) binary value 0.
      04 filler redefines DM-value-long-first.
         05 filler              pic XX.
```

```
         05 DM-value-first       pic 9(4) binary.
      04 DM-value-long-second    pic 9(9) binary value 0.
      04 filler redefines DM-value-long-second.
         05 filler               pic XX.
         05 DM-value-second      pic 9(4) binary.
   03 DM-value-attribute         pic 9(9) binary value 0.
   03 DM-value-method            pic 9(9) binary value 0.
   03 DM-value-pointer           pic X(8) value low-values.
   03 DM-value-string-putlen     pic 9(4) binary value 0.
   03 DM-value-string-getlen     pic 9(4) binary value 0.
   03 DM-value-string-size       pic 9(4) binary value 80.
   03 DM-value-string-S.
      04 DM-value-string         pic X(80) value low-values.
      04 filler                  pic X(80) value low-values.
   03 filler                     pic XX value low-values.
```

Definition in der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

```
02 DM-Value.
   03 DM-object                  pic 9(9) binary value 0.
   03 DM-attribute               pic 9(9) binary value 0.
   03 DM-indexcount              pic 9(4) binary value 0.
   03 DM-index                   pic 9(4) binary value 0.
   03 DM-second                  pic 9(4) binary value 0.
   03 DM-datatype                pic 9(4) binary value 0.
   03 DM-long-first              pic 9(9) binary value 0.
   03 DM-long-second             pic 9(9) binary value 0.
   03 DM-value-object            pic 9(9) binary value 0.
   03 DM-value-boolean           pic 9(4) binary value 0.
   03 DM-value-cardinal          pic 9(4) binary value 0.
   03 DM-value-classid           pic XX value "??".
   03 filler                     pic XX value low-values.
   03 DM-value-integer           pic S9(9) binary value 0.
   03 DM-value-index.
      04 DM-value-long-first     pic 9(9) binary value 0.
      04 filler redefines DM-value-long-first.
         05 filler               pic XX.
         05 DM-value-first       pic 9(4) binary.
      04 DM-value-long-second    pic 9(9) binary value 0.
      04 filler redefines DM-value-long-second.
         05 filler               pic XX.
         05 DM-value-second      pic 9(4) binary.
   03 DM-value-attribute         pic 9(9) binary value 0.
   03 DM-value-method            pic 9(9) binary value 0.
$IF P64 SET
   03 DM-value-pointer           pointer value null.
$ELSE
```

```
   03 DM-value-pointer           pointer value null.
   03 filler                     pic X(4) value low-values.
$END
   03 DM-value-string-putlen     pic 9(4) binary value 0.
   03 DM-value-string-getlen     pic 9(4) binary value 0.
   03 DM-value-string-size       pic 9(4) binary value 80.
   03 DM-value-string-S.
      04 DM-value-string         pic X(80) value low-values.
      04 filler                  pic X(80) value low-values.
   03 filler redefines DM-value-string-S.
      04 DM-value-string-u       pic N(80) national.
   03 filler                     pic XX value low-values.
```

Bedeutung der Strukturelemente

DM-object

Objekt, dessen Attribut verändert oder zurückgegeben werden soll.

DM-attribute

Attribut, das gesetzt oder zurückgegeben werden soll.

DM-indexcount

Mit Hilfe dieses Strukturelementes wird gesteuert, wie viele Indizes der Dialog Manager beim Setzen oder Abfragen eines Attributes verwenden soll.

DM-index

Index des Attributes, das gesetzt oder zurückgegeben werden soll.

Dieser Index ist nur gültig oder wird nur benutzt, wenn das Attribut ein indiziertes Attribut ist oder *DM-indexcount* mit einem Wert größer *0* belegt ist.

DM-second

Dieses Element stellt den zweiten Index dar. Er wird nur benutzt, wenn *DM-indexcount* größer als *1* ist.

DM-long-first

Dieses Element stellt den ersten Index dar. Er wird nur benutzt, wenn *DM-indexcount* größer als *1* ist und eine Funktion aufgerufen wird, die den langen Index auswertet. Diese erkennt man an dem Namen „DMcob_L*“. Damit können Indexwerte im Bereich von *0* bis *65535* an den Dialog Manager übergeben werden.

DM-long-second

Dieses Element stellt den zweiten Index dar. Er wird nur benutzt, wenn *DM-indexcount* größer als *1* ist und eine Funktion aufgerufen wird, die den langen Index auswertet. Diese erkennt man an dem Namen „DMcob_L*“.

DM-datatype

Dieser Datentyp beschreibt, welcher Teil der Struktur gefüllt ist und benutzt werden soll, um den Wert zu erhalten.

DM-value-integer

Wird verwendet, um ein Integer-Attribut zu setzen oder zu erhalten.

DM-value-object

Wird verwendet, um ein Attribut zu setzen oder zu erhalten, das ein IDM-Objekt wie Farbe, Zeichensatz oder Objekt ist.

DM-value-boolean

Wird verwendet, um Boolean-Attribute zu setzen oder zu erhalten.

DM-value-cardinal

Wird verwendet, um Enumeration und Cardinal Werte zu übergeben.

DM-value-classid

Wird verwendet, um die Klasse zu erhalten, zu der das Objekt gehört.

DM-value-first, DM-value-second

Diese beiden Elemente werden immer zusammen benutzt. In ihnen wird ein Indexwert an den Dialog Manager übergeben bzw. ein Indexwert vom Dialog Manager an die COBOL-Funktion zurückgegeben.

DM-value-long-first, DM-value-long-second

Diese beiden Elemente werden immer zusammen benutzt. In ihnen wird ein langer Indexwert mit dem Wertebereich *0 … 65535* an den Dialog Manager übergeben bzw. vom Dialog Manager an die COBOL-Funktion zurückgegeben.

DM-value-attribute

Dieses Element wird belegt, wenn die ID eines benutzerdefinierten Attributes erfragt werden soll.

DM-value-method

Dieses Element wird zur Handhabung von im Dialog Manager implementierten Methoden verwendet.

DM-value-pointer

Dieses Element dient zur Übergabe von Sammlungen (Wertereferenzen) und Daten mit dem Datentyp *DT-anyvalue*.

Es wird nur von der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL verwendet.

DM-value-string-putlen

Kann gesetzt werden, um dem Dialog Manager das Ende eines Strings zu zeigen, wenn der String abweichend von den getroffenen Definitionen beendet werden soll. Es wird die Anzahl der

Zeichen angegeben.

DM-value-string-getlen

Beinhaltet die Länge des vom Dialog Manager zurückgegebenen Strings. Es wird die Anzahl der Zeichen angegeben.

DM-value-string-size

Internes Feld, auf keinen Fall direkt zu ändern! Es wird die Anzahl der Zeichen angegeben.

DM-value-string

Wird verwendet, um String-Attribute zu setzen oder zu erhalten. Diese Strings dürfen nicht länger als *80* Zeichen sein.

DM-value-string-u

In diesem Element können Strings als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Strings (UTF-16) gearbeitet wird (nur MICRO FOCUS VISUAL COBOL).

Hinweis

Vor IDM-Version A.06.01.d war die Struktur *DM-Value* wie folgt definiert:

```
02 DM-Value.
   03 DM-object              pic 9(9) binary value 0.
   03 DM-attribute           pic 9(9) binary value 0.
   03 DM-indexcount          pic 9(4) binary value 0.
   03 DM-index               pic 9(4) binary value 0.
   03 DM-second              pic 9(4) binary value 0.
   03 DM-datatype            pic 9(4) binary value 0.
   03 DM-long-first          pic 9(9) binary value 0.
   03 DM-long-second         pic 9(9) binary value 0.
   03 DM-value-object        pic 9(9) binary value 0.
   03 DM-value-boolean       pic 9(4) binary value 0.
   03 DM-value-cardinal      pic 9(4) binary value 0.
   03 DM-value-classid       pic XX value "??".
   03 filler                 pic XX value low-values.
   03 DM-value-integer       pic S9(9) binary value 0.
   03 DM-value-first         pic 9(4) binary value 0.
   03 DM-value-second        pic 9(4) binary value 0.
   03 DM-value-attribute     pic 9(9) binary value 0.
   03 DM-value-method        pic 9(9) binary value 0.

   03 DM-value-string-putlen pic 9(4) binary value 0.
   03 DM-value-string-getlen pic 9(4) binary value 0.
   03 DM-value-string-size   pic 9(4) binary value 80.
   03 DM-value-string        pic X(80) value low-values.
   03 filler                 pic XX value low-values.
```

4.5.2 DM-ValueIndex

Diese Struktur ermöglicht ein vollwertiges Index-Argument. Sie wird von den **DMcob_Value*** Funktionen der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL für die Verarbeitung der **Sammlungsdatentypen** genutzt. Die Struktur steht aber für alle unterstützten COBOL-Varianten zur Verfügung und ist nicht auf diese Anwendungsbereiche beschränkt. Sie ist identisch mit der Struktur DM-Value.

4.5.3 DM-ValueArray

Für den Aufruf einiger Dialog Manager Funktionen wird ein Array von bis zu 16 Strukturelementen zum Aufruf benötigt. Dazu ist folgende Struktur definiert:

```
02 DM-ValueArray.
   03 DM-value-size              pic 9(4) binary value 16.
   03 DM-value-count             pic 9(4) binary value 0.
   03 DM-ValueRecord occurs 16 indexed by DM-valuecount.
      04 DM-va-object            pic 9(9) binary value 0.
      04 DM-va-attribute         pic 9(9) binary value 0.
      04 DM-va-indexcount        pic 9(4) binary value 0.
      04 DM-va-index             pic 9(4) binary value 0.
      04 DM-va-second            pic 9(4) binary value 0.
      04 DM-va-datatype          pic 9(4) binary value 0.
      04 DM-va-long-first        pic 9(9) binary value 0.
      04 DM-va-long-second       pic 9(9) binary value 0.
      04 DM-va-value-object      pic 9(9) binary value 0.
      04 DM-va-value-boolean     pic 9(4) binary value 0.
      04 DM-va-value-cardinal    pic 9(4) binary value 0.
      04 DM-va-value-classid     pic XX value "??".
      04 filler                  pic XX value low-values.
      04 DM-va-value-integer     pic S9(9) binary value 0.
      04 DM-va-value-index.
         05 DM-va-value-long-first pic 9(9) binary value 0.
         05 filler redefines DM-va-value-long-first.
            06 filler            pic XX.
            06 DM-va-value-first pic 9(4) binary.
         05 DM-va-value-long-second pic 9(9) binary value 0.
         05 filler redefines DM-va-value-long-second.
            06 filler            pic XX.
            06 DM-va-value-second pic 9(4) binary.
      04 DM-va-value-attribute   pic 9(9) binary value 0.
      04 DM-va-value-method      pic 9(9) binary value 0.
      04 DM-va-value-pointer     pic X(8) value low-values.
      04 DM-va-value-string-putlen pic 9(4) binary value 0.
      04 DM-va-value-string-getlen pic 9(4) binary value 0.
      04 DM-va-value-string-size   pic 9(4) binary value 80.
      04 DM-va-value-string-S.
```

```
         05 DM-va-value-string   pic X(80) value low-values.
         05 filler               pic X(80) value low-values.
      04 filler                  pic XX value low-values.
```

Definition in der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

```
02 DM-ValueArray.
   03 DM-value-size              pic 9(4) binary value 16.
   03 DM-value-count             pic 9(4) binary value 0.
   03 DM-ValueRecord occurs 16 indexed by DM-valuecount.
      04 DM-va-object            pic 9(9) binary value 0.
      04 DM-va-attribute         pic 9(9) binary value 0.
      04 DM-va-indexcount        pic 9(4) binary value 0.
      04 DM-va-index             pic 9(4) binary value 0.
      04 DM-va-second            pic 9(4) binary value 0.
      04 DM-va-datatype          pic 9(4) binary value 0.
      04 DM-va-long-first        pic 9(9) binary value 0.
      04 DM-va-long-second       pic 9(9) binary value 0.
      04 DM-va-value-object      pic 9(9) binary value 0.
      04 DM-va-value-boolean     pic 9(4) binary value 0.
      04 DM-va-value-cardinal    pic 9(4) binary value 0.
      04 DM-va-value-classid     pic XX value "??".
      04 filler                  pic XX value low-values.
      04 DM-va-value-integer     pic S9(9) binary value 0.
      04 DM-va-value-index.
         05 DM-va-value-long-first pic 9(9) binary value 0.
         05 filler redefines DM-va-value-long-first.
            06 filler            pic XX.
            06 DM-va-value-first pic 9(4) binary.
         05 DM-va-value-long-second pic 9(9) binary value 0.
         05 filler redefines DM-va-value-long-second.
            06 filler            pic XX.
            06 DM-va-value-second pic 9(4) binary.
      04 DM-va-value-attribute   pic 9(9) binary value 0.
      04 DM-va-value-method      pic 9(9) binary value 0.
$IF P64 SET
      04 DM-va-value-pointer     pointer value null.
$ELSE
      04 DM-va-value-pointer     pointer value null.
      04 filler                  pic X(4) value low-values.
$END
      04 DM-va-value-string-putlen pic 9(4) binary value 0.
      04 DM-va-value-string-getlen pic 9(4) binary value 0.
      04 DM-va-value-string-size   pic 9(4) binary value 80.
      04 DM-va-value-string-S.
         05 DM-va-value-string   pic X(80) value low-values.
         05 filler               pic X(80) value low-values.
```

```
      04 filler redefines DM-va-value-string-S.
         05 DM-va-value-string-u pic N(80) national.
      04 filler                  pic XX value low-values.
```

Bedeutung der Strukturelemente

DM-value-size

Hier wird die Größe des Feldes abgelegt. Dieser Wert ist momentan mit *16* vorbelegt und darf nicht geändert werden.

Hinweis

Der Wert kann sich in zukünftigen Versionen ändern.

DM-value-count

In diesem Feld wird die momentane Belegung des Arrays abgelegt, d.h., die Anzahl der im Moment benutzten Elemente des Feldes.

DM-va-object

Objekt, dessen Attribut verändert oder zurückgegeben werden soll.

DM-va-attribute

Attribut, das gesetzt oder zurückgegeben werden soll.

DM-va-indexcount

Mit Hilfe dieses Strukturelementes wird gesteuert, wie viele Indizes der Dialog Manager beim Setzen oder Abfragen eines Attributes verwenden soll.

DM-va-index

Index des Attributes, das gesetzt oder zurückgegeben werden soll.

Dieser Index ist nur gültig, wenn *DM-va-indexcount* mit einem Wert größer *0* belegt ist.

DM-va-second

Dieses Element stellt den zweiten Index dar. Er wird nur benutzt, wenn *DM-va-indexcount* größer als *1* ist.

DM-va-datatype

Dieser Datentyp beschreibt, welcher Teil der Struktur gefüllt ist und benutzt werden soll, um den Wert zu erhalten.

DM-va-value-integer

Wird verwendet, um ein Integer-Attribut zu setzen oder zu erhalten.

DM-va-value-object

Wird verwendet, um ein Attribut zu setzen oder zu erhalten, das ein IDM-Objekt wie Farbe, Zeichensatz oder Objekt ist.

DM-va-value-boolean

Wird verwendet, um Boolean-Attribute zu setzen oder zu erhalten.

DM-va-value-cardinal

Wird verwendet, um Enumeration und Cardinal Werte zu übergeben.

DM-va-value-classid

Wird verwendet, um die Klasse zu erhalten, zu der das Objekt gehört.

DM-va-value-first, DM-va-value-second

Diese beiden Elemente werden immer zusammen benutzt. In ihnen wird ein Indexwert an den Dialog Manager übergeben bzw. vom Dialog Manager an die COBOL-Funktion zurückgegeben.

DM-va-value-long-first, DM-va-value-long-second

Diese beiden Elemente werden immer zusammen benutzt. In ihnen wird ein langer Indexwert mit dem Wertebereich *0 … 65535* an den Dialog Manager übergeben bzw. vom Dialog Manager an die COBOL-Funktion zurückgegeben.

DM-va-value-attribute

Dieses Element wird belegt, wenn die ID eines benutzerdefinierten Attributs erfragt werden soll.

DM-va-value-method

Dieses Element wird zur Handhabung von im Dialog Manager implementierten Methoden verwendet.

DM-va-value-pointer

Dieses Element dient zur Übergabe von Sammlungen (Wertereferenzen) und Daten mit dem Datentyp *DT-anyvalue*.

Es wird nur von der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL verwendet.

DM-va-value-string-putlen

Kann gesetzt werden, um dem Dialog Manager das Ende eines Strings zu zeigen, wenn der String abweichend von den getroffenen Definitionen beendet werden soll.

DM-va-value-string-getlen

Beinhaltet die Länge des vom Dialog Manager zurückgegebenen Strings.

DM-va-value-string-size

Internes Feld, auf keinen Fall direkt zu ändern!

DM-va-value-string

Wird verwendet, um String-Attribute zu setzen oder zu erhalten. Diese Strings dürfen nicht länger als 80 Zeichen sein.

DM-va-value-string-u

In diesem Element können Strings als „National Character“ (PIC N) übergeben werden, wenn mit Unicode-Strings (UTF-16) gearbeitet wird (nur MICRO FOCUS VISUAL COBOL).

Hinweis

Vor IDM-Version A.06.01.d war die Struktur *DM-ValueArray* wie folgt definiert:

```
02 DM-ValueArray.
   03 DM-value-size          pic 9(4) binary value 16.
   03 DM-value-count         pic 9(4) binary value 0.
   03 DM-ValueRecord occurs 16 indexed by DM-valuecount.
      04 DM-va-object        pic 9(9) binary value 0.
      04 DM-va-attribute     pic 9(9) binary value 0.
      04 DM-va-indextype     pic 9(4) binary value 0.
      04 DM-va-index         pic 9(4) binary value 0.
      04 DM-va-second        pic 9(4) binary value 0.
      04 DM-va-datatype      pic 9(4) binary value 0.
      04 DM-va-long-first    pic 9(9) binary value 0.
      04 DM-va-long-scrond   pic 9(9) binary value 0.

      04 DM-va-value-object    pic 9(9) binary value 0.
      04 DM-va-value-boolean   pic 9(4) binary value 0.
      04 DM-va-value-cardinal  pic 9(4) binary value 0.
      04 DM-va-value-classid   pic XX value "??".
      04 filler                pic XX value low-values.
      04 DM-va-value-integer   pic S9(9) binary value 0.
      04 DM-va-value-first     pic 9(4) binary value 0.
      04 DM-va-value-second    pic 9(4) binary value 0.
      04 DM-va-value-attribute pic 9(9) binary value 0.
      04 DM-va-value-method    pic 9(9) binary value 0.

      04 DM-va-value-string-putlen pic 9(4) binary value 0.
      04 DM-va-value-string-getlen pic 9(4) binary value 0.
      04 DM-va-value-string-size   pic 9(4) binary value 80.
      04 DM-va-value-string  pic X(80) value low-values.
      04 filler              pic XX value low-values.
```

4.5.4 Datentypen in der DM-Value und DM-ValueArray Struktur

Werden über **DM-Value** bzw. **DM-ValueArray** beim Dialog Manager Werte erfragt, so setzt der DM entsprechend dem erfragten Attribut das Strukturelement *DM-datatype* bzw. *DM-va-datatype*.

Dabei gibt es folgende mögliche Belegungen:

Identifikator (ID)	COBOL-Daten-typ	Bedeutung/belegtes Element
DT-accel	pic 9 (9) binary	Acceleratorreferenz. DM-value-object / DM-va-value-object
DT-anyvalue	pointer	Daten mit beliebigem Datentyp. DM-value-pointer / DM-va-value-pointer
DT-application	pic 9 (9) binary	Applicationreferenz. DM-value-object / DM-va-value-object.
DT-attribute	pic 9 (9) binary	Datentyp eines Attributes im Dialog Manager.
DT-boolean	pic 9 (4) binary	Boolean-Wert. DM-value-boolean / DM-va-value-boolean
DT-class	pic XX	Dieser Datentyp beschreibt die Klassen der verschiedenen Objekte. DM-value-classid / DM-va-value-classid
DT-color	pic 9 (9) binary	Farbreferenz. DM-value-object / DM-va-value-object
DT-cursor	pic 9 (9) binary	Cursorreferenz. DM-value-object / DM-va-value-object
DT-enum	pic 9 (4) binary	Stellt den Enumerationtyp im DM dar. DM-value-cardinal / DM-va-value-cardinal.
DT-event	pic 9 (4) binary	Definiert den Namen eines Ereignisses. DM-value-cardinal / DM-va-value-cardinal.
DT-font	pic 9 (9) binary	Zeichensatzreferenz.
DT-hash	pointer	Hash-Referenz. DM-value-pointer / DM-va-value-pointer
DT-index	pic 9 (4) binary	Dieser Datentyp beschreibt einen zweidimensionalen Index. Aus diesem Grund müssen auch immer zwei Elemente in der Value Struktur ausgewertet werden. DM-value-first DM-value-second / DM-va-value-first & DM-va-value-second.
DT-instance	pic 9 (9) binary	Instanzreferenz. DM-value-object / DM-va-value-object
DT-integer	pic 9 (9) binary	Zahl-Wert. DM-value-integer / DM-va-value-integer
DT-list	pointer	Listen-Referenz. DM-value-pointer / DM-va-value-pointer
DT-matrix	pointer	Matrix-Referenz. DM-value-pointer / DM-va-value-pointer
DT-object	pic 9 (9) binary	Objektreferenz. DM-value-object / DM-va-value-object

Identifikator (ID)	COBOL-Daten-typ	Bedeutung/belegtes Element
DT-refvec	pointer	Refvec-Referenz. DM-value-pointer / DM-va-value-pointer
DT-rule	pic 9 (9) binary	Regelreferenz. DM-value-object / DM-va-value-object
DT-scope	pic 9 (4) binary	Definiert, ob es sich um ein Defaultobjekt, ein Modell oder ein Objekt handelt. DM-value-cardinal / DM-va-value-car-dinal
DT-string	pic X (??) pic N (??)	Übergabe von String. DM-value-string / DM-va-value-string Übergabe eines Unicode-String (UTF-16). DM-value-string-u / DM-va-value-string-u
DT-text	pic 9 (9) binary	Textreferenz. DM-value-object / DM-va-value-object
DT-tile	pic 9 (9) binary	Musterreferenz. DM-value-object / DM-va-value-object
DT-timer	pic 9 (9) binary	Timerreferenz. DM-value-object / DM-va-value-object
DT-undefined	keines	Datentyp ist undefiniert.
DT-var	pic 9 (9) binary	Variablenreferenz. DM-value-object / DM-va-value-object
DT-vector	pointer	Vektor-Referenz. DM-value-pointer / DM-va-value-pointer
DT-void	keines	Datentyp ist unbekannt oder nicht spezifiziert.

Da es beim Setzen von Attributwerten mehrere mögliche Datentypen geben kann, muss entsprechend dem verwendeten Datentyp das Element *DM-datatype* gesetzt werden, damit der DM den Wert aus dem entsprechenden Element herauslesen kann.

Die möglichen Werte für *DT-scope* sind:

1 Defaultobjekt

2 Modell

3 Instanz

4.6 Objektcallback

Alle COBOL-Callback-Funktionen werden mit **zwei** Parametern aufgerufen: der erste ist die allgemeine Datenstruktur, die beim Aufruf von **DMcob_Initialize** von der Anwendung an den Dialog Manager übergeben wurde; der zweite Parameter ist eine Struktur, die vom Dialog Manager definiert

wurde. In dieser zweiten Struktur, der DM-ObjectCallback-Data, werden der Anwendung die zu diesem Callback gehörenden Daten übergeben.

```
01  DM-ObjectCallback-Data
    02  DM-ocb-status       pic 9(4) binary.
    02  DM-ocb-object       pic 9(9) binary.
    02  DM-ocb-evbits       pic 9(4) binary.
    02  DM-ocb-index        pic 9(4) binary.
    02  DM-ocb-accel        pic 9(9) binary.
```

Durch das *DM-ocb-object*-Feld wird das Objekt übergeben, welches an die Funktion gebunden ist.

Wenn das Objekt mehr als einen Eintrag hat (z.B. Selektion in einer Listbox oder einem Poptext), ist der Index des selektierten Eintrags im *DM-ocb-index*-Feld enthalten.

Die Ereignismaske des auslösenden Ereignisses wird in *DM-ocb-evbits* übergeben. Da genau ein Bit zu jedem Ereignis in der Maske gesetzt wird, können diese Ereignisse auch in Kombination verwendet werden. Untenstehend finden Sie die Möglichkeiten, wie sie in **IDMcobws.cob** definiert sind. Das *DM-ocb-accel*-Feld wird nur gesetzt, wenn der Callback für ein Key-Ereignis aufgerufen wird.

Im *DM-ocb-status*-Feld kann die Applikation dem Dialog Manager mitteilen, die Regeln, die zu diesem Ereignis gehören, auszuführen oder die Regeln nicht abzuarbeiten. Wenn der Wert dieses Statusfeldes *DM-success* ist, wird der Dialog Manager alle Regeln ausführen, die zu diesem Ereignis gehören. Wenn die Applikationsfunktionen *not DM-success* zurückgeben, wird der Dialog Manager die Regeln, die zu diesem Ereignis gehören, nicht ausführen.

Eventbit	Bedeutung
EM-activate	Objekt wurde aktiviert
EM-charinput	Eingabe von Zeichen in einen editierbaren Text
EM-close	Fenster wurde geschlossen
EM-dbselect	Objekt wurde mit einem Doppelklick selektiert
EM-deactivate	Objekt wurde deaktiviert
EM-deiconify	Icon wurde zu einem Fenster gemacht
EM-deselect-enter	Editierbarer Text wurde mit `Return`-Taste (`Enter`) verlassen
EM-deselect	Objekt wurde deselektiert
EM-finish	Dialogabarbeitung wurde beendet
EM-focus	Eingabefokus wurde an Objekt gesetzt
EM-help	Für Objekt wurde Hilfe angefragt
EM-hscroll	Objekt wurde horizontal gescrollt

Eventbit	Bedeutung
EM-iconify	Fenster wurde ikonifiziert
EM-key	Eingabe von Zeichen
EM-modified	Inhalt von editierbarem Text wurde geändert.
EM-move	Objekt wurde bewegt
EM-resize	Objektgröße wurde verändert
EM-scroll	Objekt wurde gescrollt
EM-select	Objekt wurde selektiert
EM-start	Dialogabarbeitung wurde gestartet
EM-vscroll	Objekt wurde vertikal gescrollt

Anmerkung

Der erste Parameter eines Objektcallbacks ist in COBOL immer *DM-COMMON-DATA*!

Der COBOL-Objektcallback muss im Dialogskript wie folgt definiert werden:

```
callback cobol FuncName ();
```

In COBOL muss diese Funktion dann wie folgt definiert werden:

```
entry "FuncName" using DM-COMMON-DATA
        DM-ObjectCallback-Data.
```

4.7 Nachlade-Funktionalität

Alle COBOL Nachlade-Funktionen werden mit **zwei** Parametern aufgerufen: der erste ist die allgemeine Datenstruktur, die beim Aufruf von **DMcob_Initialize** von der Anwendung an den Dialog Manager übergeben wurde; der zweite Parameter ist eine Struktur, die **DM-Content-Data** Struktur, die vom Dialog Manager definiert wurde. Nachladefunktionen werden vom Dialog Manager immer dann aufgerufen, wenn in einem Tablefield durch Scrollen des Benutzers Teile sichtbar gemacht wurden, denen bisher noch kein Inhalt zugewiesen worden ist.

Diese Struktur ist wie folgt definiert:

```
01  DM-Content-Data
    02  DM-CO-OBJECT          pic 9(9) binary.
    02  DM-CO-REASON          pic 9(4) binary.
    02  DM-CO-VISFIRST        pic 9(4) binary.
    02  DM-CO-VISLAST         pic 9(4) binary.
    02  DM-CO-LOADFIRST       pic 9(4) binary.
    02  DM-CO-LOADLAST        pic 9(4) binary.
    02  DM-CO-COUNT           pic 9(4) binary.
```

```
    02  DM-CO-HEADER          pic 9(4) binary.
```

DM-CO-OBJECT

Hier wird das Objekt übergeben, an das diese Funktion gebunden ist, d.h. der Identifikator des zu editierenden Tablefields.

DM-CO-REASON

Hier wird der Grund angegeben, warum die Funktion aufgerufen worden ist.

Hier ist der Wert *CFR-LOAD* möglich. Dieser Wert soll angeben, dass der Inhalt in das Tablefield nachgeladen werden soll.

Die folgenden Parameter sind von der Bedeutung her abhängig davon, wie das Tablefield definiert wurde.

Ist *.direction* mit *1* definiert, sind alle nachfolgenden Werte Zeilen.

Ist *.direction* mit *2* definiert, sind alle nachfolgenden Werte Spalten.

DM-CO-VISFIRST

Bezeichnet die erste sichtbare Zeile/Spalte.

DM-CO-VISLAST

Bezeichnet die letzte sichtbare Zeile/Spalte.

DM-CO-FIRSTLOAD

Bezeichnet die erste zu ladende Zeile/Spalte.

DM-CO-LASTLOAD

Bezeichnet die letzte zu ladende Zeile/Spalte.

DM-COUNT

Bezeichnet die insgesamt definierte Zeilen-/Spaltenzahl.

DM-Header

Bezeichnet die als Überschrift definierte Anzahl von Zeilen/Spalten.

Die COBOL-Nachladefunktion muss im Dialogskript wie folgt definiert werden:

```
function cobol contentfunc FuncName( );
```

In COBOL wie folgt:

```
ENTRY "FUNCNAME" using DM-COMMON-DATA DM-CONTENT-DATA.
```

Um diese Struktur setzen zu können, muss die Datei

IDMcoboc.cob

über den Befehl

```
COPY
```

in die Quelle kopiert werden.

Beispiel

```
WORKING-STORAGE SECTION.

77  DM-POINTER    PIC 9(4) BINARY VALUE 0.
77  ICOUNT        PIC 9(4) BINARY VALUE 0.
77  IROW           PIC 9(4) BINARY VALUE 0.
77  ICOL           PIC 9(4) BINARY VALUE 0.
77  IDUMMY        PIC 9(4) VALUE 0.

LINKAGE SECTION.
COPY "IDMcobls.cob".
COPY "IDMcoboc.cob".

* Nachladefunktion
    ENTRY "ContFunc" USING DM-COMMON-DATA DM-Content-data.
    COMPUTE ICOUNT = DM-co-loadfirst - DM-co-loadlast.
    COMPUTE ICOUNT = ICOUNT + 1.
    COMPUTE ICOUNT = ICOUNT * ICOL.
    COMPUTE BASE = DM-co-loadfirst * ICOL.
    MOVE ICOUNT TO DLG-COUNT.
    MOVE DM-CO-OBJECT TO DM-OBJECT.
    MOVE DT-string TO DM-DATATYPE.

* Initialisierung eines Speicherplatzes im DM
    CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
        DT-STRING ICOUNT.

    PERFORM FILLDATA.

* Übergeben der gespeicherten Werte an die Anzeige
    MOVE AT-CONTENT TO DM-ATTRIBUTE.
    MOVE 2 TO DM-INDEXCOUNT.
    MOVE DM-co-loadfirst TO DM-INDEX.
    MOVE 1 TO DM-second.
    CALL "DMcob_FreeVector" USING DM-StdArgs DM-VALUE
            DM-POINTER 0
            DM-co-loadlast, ICOL.

* Freigeben des Speicherbereiches
    CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.

* Loeschen des Inhalts im nicht sichtbaren Bereich
    MOVE DT-INTEGER TO DM-VA-DATATYPE(1).
    MOVE DT-INTEGER TO DM-VA-DATATYPE(2).
    COMPUTE DM-VA-VALUE-INTEGER(1) = DM-CO-HEADER + 1.
    COMPUTE DM-VA-VALUE-INTEGER(2) = DM-CO-HEADER - 1
```

```
            - DM-CO-HEADER.
    MOVE 2 TO DM-VALUE-COUNT.

    CALL "DMcob_CallMethod" USING DM-STDARGS DM-VALUE
            DM-CO-OBJECT
            MT-CLEAR, DM-VALUEARRAY.

    GOBACK.
```

4.8 Datenfunktions-Struktur DM-Datafunc-Data

Diese Struktur wird immer als Parameter an Datenfunktionen übergeben.

Die Struktur ist hier vereinfacht wiedergegeben. Die vollständige Definition steht in der Datei **IDM-coboc.cob**.

```
01 DM-Datafunc-Data.
   02 DM-df-object                 pic 9(9) binary.
   02 DM-df-task                   pic 9(9) binary.
   02 DM-df-attribute              pic 9(9) binary.
   02 DM-df-method                 pic 9(9) binary.
   02 DM-df-identifier             pic X(80).

   02 DM-df-args.
*     Ist identisch zur Struktur DM-ValueArray.

   02 DM-df-index.
*     Ist identisch zur Struktur DM-Value.

   02 DM-df-data-pointer           pointer.
   02 DM-df-retval-pointer         pointer.
```

Bedeutung der Elemente

DM-df-object

In diesem Element wird die Objekt-ID des Datenmodells an die Datenfunktion übergeben.

DM-df-task

In diesem Element wird der Grund des Aufrufs angegeben. Hier sind zur Zeit die folgende Werte möglich: *MT-get*, *MT-set* oder *MT-call*. Bei *MT-get* sollte die Datenfunktion für ein Modell-Attribut die nötigen Daten zurückliefern. Über *MT-set* werden Änderungen von der View an die Datenfunktion weitergeleitet. *MT-call* dient dazu, Daten-Aktionen zu implementieren, also Aufrufe welche zu einer „vielfältigen“ Manipulation der Daten führen. Sinnvollerweise sollten Änderungen an Daten immer über **DMcob_DataChanged** gemeldet werden.

DM-df-attribute

Für die Aufrufgründe *MT-get* und *MT-set* steht in diesem Element das Attribut auf das die Funktion anzuwenden ist.

DM-df-method

Für den Aufrufgrund *MT-call* steht in diesem Element die Methode welche die Datenfunktion ausführen soll. Diese wird durch einen Aufruf einer Daten-Aktion über die Methode *:calldata(<Method>,<Arg1>,...<Arg15>)* ausgelöst.

DM-df-identifier
DM-df-indentifier-u

In diesen Elementen steht der Attributbezeichner (aus dem *DM-df-attribute*-Element) oder der Methodenbezeichner (aus dem *DM-df-method*-Element) in String-Form um die Implementierung bei benutzerdefinierten Attributen zu erleichtern bzw. einen modulunabhängigen Vergleich von Attributen bzw. Methoden zu ermöglichen.

In *DM-df-indentifier-u* können Bezeichner auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-df-args

In diesem Element sind für den Aufrufgrund *MT-call* (Daten-Aktion die über **:calldata()** aufgerufen wird) die Anzahl der Argumente (*DM-df-args-count*) und die Argumentliste (*DM-df-valuerec*) hinterlegt. Eine Rückgabe oder Veränderung von Argumenten ist damit nicht vorgesehen bzw. ohne Wirkung.

Hinweis

Diese Struktur ist vom Aufbau identisch zu **DM-ValueArray**.

DM-df-index

In diesem Element steht der Indexwert des Attributzugriffs auf den sich der Aufrufgrund *MT-get* bzw. *MT-set* bezieht. Hat das *DM-df-index*-Element den Datentyp *DT-void*, ist damit der Gesamtwert gemeint. Für indizierte Datenwerte sollten die Relationsarten und auch die Verarbeitung von unvollständigen Indexwerten (z.B. *[0]*, *[?,0]* oder *[0,?]*) beachtet werden.

Hinweis

Diese Struktur ist vom Aufbau identisch zu **DM-Value**.

DM-df-data-pointer

In diesem Element steht für den Aufrufgrund *MT-set* der Datenwert der von der Datenfunktion zu verarbeiten ist.

Hinweis

Die Daten werden als *DT-anyvalue*-Wert übergeben, da es sich hier meist um Sammlungen handeln wird. Der Wert bzw. die Werte können mit **DMcob_ValueGet** und den anderen **DMcob_Value***-Funktionen ausgelesen werden.

DM-df-retval-pointer

In diesem Element ist für den Aufrufgrund *MT-get* der Datenwert für das angegebene Attribut und den gesetzten Index zurückzuliefern.

Hinweis

Der Rückgabewert wird als *DT-anyvalue*-Wert übergeben, da es sich hier meist um Sammlungen handeln wird. Der Rückgabewert kann mit **DMcob_ValueChange** geändert und gefüllt werden.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

4.9 NULL-Objekt

Es gibt ein Objekt mit einem speziellen Identifikator, mit dessen Hilfe fehlende Ressource-Attribute und das Rücksetzen von Ressource-Attributen ermöglicht werden.

Dieses Objekt wird "NULL-Objekt" genannt und hat die Dialog Manager ID = 0.

Dieses NULL-Objekt kann dazu benutzt werden, alle Ressource-Attribute zurückzusetzen.

Es kann von **DMcob_GetValue** erhalten werden und kann für **DMcob_SetValue** oder für Funktionsargumente benutzt werden.

Das NULL-Objekt selbst ist Typ-unabhängig, d.h. es gibt nur ein NULL-Objekt für alle Ressourcen. Das NULL-Objekt kann auf alle Datentypen angewendet werden, die >= DT-instance sind. Bei GetValue ist der Realtyp des NULL-Objektes abhängig vom Typ des Attributes.

Auswirkungen

- Interface-Funktionen können ein NULL-Objekt als Return-Wert haben.
- Das NULL-Objekt kann auch als (Eingabe- und Ausgabe-) Parameter vorkommen.
- Wenn ein NULL-Objekt als Return-Wert von **DMcob_GetValue** erhalten wird, wird es immer den genauest möglichen Datentyp haben, z.B. *DT-text* anstatt *DT-instance*.
- Wenn ein Attribut von der Applikation auf Null gesetzt werden soll, muss der Datentyp so genau wie möglich angegeben werden, d.h. z.B. *DT-accelerator* anstatt *DT-instance*. *DT-instance* kann natürlich auch benutzt werden. Der Dialog Manager transformiert dies intern in den richtigen Datentyp; diese Methode ist aber anfälliger für Programmierfehler.

4.10 Zugriff auf Attribute

Der Zugriff auf die standardmäßig vom Dialog Manager angebotenen Attribute erfolgt über die in den Copy-Dateien **IDMcobws.cob** und **IDMcobls.cob** enthaltenen Definitionen.

Diese Attributdefinitionen können dann einfach in der DM-Value Struktur gesetzt und an den Dialog Manager übergeben werden.

Beispiel

Setzen des Attributes "visible" in der DM-Value Struktur.

```
MOVE AT-visible TO DM-ATTRIBUTE
```

Eine vollständige Auflistung der standardmäßig definierten Attribute kann der „Attributreferenz“ entnommen werden.

Anders sieht es jedoch mit den benutzerdefinierten Attributen aus. Diese werden von jedem Dialogprogrammierer definiert und können daher nicht global verfügbar gemacht werden. Wenn vom COBOL-Interface aus auf diese benutzerdefinierten Attribute zugegriffen werden soll, muss zunächst deren aktueller interner Wert ermittelt werden. Dieser aktuelle interne Wert ist dabei vom Dialog abhängig, über die Lebensdauer des Dialoges jedoch statisch.

Mit diesem aktuellen internen Wert kann nach dessen Ermittlung wie mit den von Dialog Manager statisch definierten Attributen gearbeitet werden.

Die Ermittelung ihres Wertes erfolgt über die Funktion DMcob_GetValue. Dazu muss in DM-datatype *DT-string* und in DM-attribute *AT-label* gesetzt werden. Das Objekt muss der Dialog sein, zu dem das Attribut gehört. Der Indexcount muss *2* sein, der Wert der Indizes jeweils *0*. Der Name des zu vermittelnden Attributes muss in DM-value-string enthalten sein.

Notwendige Werte:

DM-object	Dialog
DM-attribute	AT-label
DM-datatype	DT-string
DM-value-string	Name des gesuchten Attributes
DM-indexcount	2
DM-index	0
DM-second	0

Beispiel

Abfrage eines benutzerdefinierten Attributes bei einem Objekt.

```
ENTRY "GETATTR" USING DM-COMMON-DATA DIALOG-ID OBJECT-ID
    ATTR-NAME.

    MOVE DT-STRING TO DM-DATATYPE.
    MOVE AT-LABEL TO DM-ATTRIBUTE.
    MOVE 2 TO DM-INDEXCOUNT.
    MOVE 0 TO DM-INDEX.
    MOVE DIALOG-ID TO DM-OBJECT.
    MOVE ATTR-NAME TO DM-VALUE-STRING.
```

```
DISPLAY DM-VALUE-STRING.

CALL "DMcob_GetValue" USING DM-STDARGS DM-VALUE.
DISPLAY DM-DATATYPE.
DISPLAY DM-VALUE-ATTRIBUTE.
MOVE DM-VALUE-ATTRIBUTE TO DM-ATTRIBUTE.
MOVE OBJECT-ID TO DM-OBJECT.
MOVE 0 TO DM-INDEXCOUNT.
CALL "DMcob_GetValue" USING DM-STDARGS DM-VALUE.
DISPLAY DM-VALUE-INTEGER.
GOBACK.
```

4.11 Rückgabewerte der DM-Funktionen DM-success/DM-error

Die meisten DM-Funktionen liefern im Standardargument (DM-StdArgs) zurück, ob der Funktionsaufruf ohne oder mit Fehler durchgeführt wurde. Konnte die Funktion erfolgreich ausgeführt werden, wird DM-Status auf DM-success gesetzt. Trat bei dem Funktionsaufruf ein Fehler auf, wird DM-Status auf DM-error gesetzt. Dabei entspricht der Wert 0 DM-success, jeder andere Wert DM-error.

5 Anbindung des COBOL-Programms

5.1 Hauptprogramm

In diesem Kapitel wird beschrieben, wie die im Dialog Manager definierten Funktionen in COBOL realisiert und an den Dialog Manager angebunden werden müssen.

Prinzipiell hat jedes COBOL-Programm, das in der nicht verteilten Lösung den Namen "COBOLMAIN" besitzt, in der verteilten Lösung den Namen "COBOLAPPINIT".

Dieses Hauptprogramm wird vom Dialog Manager aufgerufen und kann dann die notwendigen Initialisierungsschritte durchführen.

5.1.1 Lokale COBOL-Programme

In der lokalen Lösung des Dialog Managers wird das Programm "COBOLMAIN" als das eigentliche Programm betrachtet. Es muss dann folgende Schritte durchführen:

- Initialisierung des Dialog Managers (**DMcob_Initialize**)
- Laden eines Dialoges (**DMcob_LoadDialog**)
- Initialisieren der Anwendung
- Übergeben der Funktionsadressen (BindFuncs, CobRecInit)
- Starten des Dialoges (**DMcob_StartDialog**)
- Starten der Verarbeitung (**DMcob_EventLoop**)

Nach der erfolgreichen Ausführung obiger Schritte kann der Anwender mit dem Programm arbeiten.

5.1.2 Verteilte COBOL-Programme

In der verteilten Lösung des Dialog Managers muss das COBOL-Hauptprogramm "COBOLAPPINIT" folgende Schritte durchführen:

- Initalisierung der COBOL-Schnittstelle des Dialog Managers (**DMcob_Initialize**)
- Übergabe der Funktionsadressen (BindFuncs)
- Initialisierung der Anwendung

5.2 Realisierung der COBOL-Funktionen

Bei der Realisierung der eigentlichen COBOL-Funktionen müssen die im Dialog definierten Parameterdatentypen auf die COBOL-Datenypen abgebildet werden. Die Funktionen können dabei als "ENTRY" oder als Programm realisiert werden.

Bei der Implementierung der COBOL-Funktionen muss beachtet werden, dass das erste Argument der Funktion immer die DM-COMMON-DATA ist. Dadurch ergibt sich gegenüber der Parameterdefinition im Dialogskript immer eine Parameterverschiebung von 1.

Beispiel

Dialogdatei

```
function void cobol Func1 (string[80] input output,
                           integer input);
function void cobol Func2 (boolean input output,
                           string[80] input);
```

COBOL

```
    77    DLG-string    pic X(80).
    77    DLG-int       pic 9(9) binary.
    77    DLG-bool      pic 9(4) binary.

    ENTRY "Func1" using DM-COMMON-DATA DLG-string DLG-int.
    ENTRY "Func2" using DM-COMMON-DATA DLG-bool DLG-string.
```

Werden bei dem Funktionsaufruf Records verwendet, so muss der generierte COBOL-Datentyp des Records als Parametertyp definiert werden.

Beispiel

Dialogdatei

```
function cobol void WriteAddr(record Address input);
function cobol void ReadAddr(record Address output,
                             integer Pos input output);
record Address
{
  string[25] Name       shadows W1.Lname.E.content;
  string[15] FirstName  shadows W1.Lfirstname.E.content;
  string[25] City       shadows W1.Lcity.E.content;
  string[30] Street     shadows W1.Lstreet.E.content;
}
```

COBOL-Hauptprogramm

```
LOAD-DIALOG.
    MOVE "DMcob_LoadDialog" TO FUNC-NAME.
    MOVE "./recordcob.dlg@" TO BUFFER.
    CALL "DMcob_LoadDialog" USING DM-STDARGS DIALOG-ID BUFFER.
    PERFORM ERROR-CHECK.
BIND-FUNCTIONS.
    CALL "cobrecninitrectest" USING DIALOG-ID RET-VALUE.
    PERFORM ERROR-CHECK.
START-DIALOG.
```

```
    MOVE "DMcob_StartDialog" TO FUNC-NAME.
    CALL "DMcob_StartDialog" USING DM-STDARGS DIALOG-ID.
    PERFORM ERROR-CHECK.
```

COBOL-Unterprogramme

```
IDENTIFICATION DIVISION.
PROGRAM-ID. READADDR.
AUTHOR. "TEST".

WORKING-STORAGE SECTION.
01 OLDPO        PIC 9(9) binary value 0.
01 DLG-DATA-STORAGE.
   03 DLG-DATA-DETAIL occurs 10 indexed by Addr-Index.
    04 A-Name pic X(25)         value SPACES.
    04 A-FirstName pic X(15)    value SPACES.
    04 A-City pic X(25)         value SPACES.
    04 A-Street pic X(30)       value SPACES.

LINKAGE SECTION.

COPY "IDMcobls.cob".
COPY "rectramcob.cob".
77 DLG_POS    PIC 9(9) binary.

PROCEDURE DIVISION USING DM-COMMON-DATA RECADDRESS DLG-POS.
    IF DLG-POS > 0 AND DLG-POS <= 10 THEN
        MOVW DLG-POS TO OLDPOS
    ELSE
        MOVE 1 TO DLG-POS
        GOBACK
    END-IF.

    MOVE A-NAME(DLGPOS) TO NAME.
    MOVE A-FIRSTNAME(DLGPOS) TO FIRSTNAME.
    MOVE A-CITY(DLGPOS) TO CITY.
    MOVE A-STREET(DLGPOS) TO STREET.
    GOBACK.
    ENTRY "WRITEADDR" USING DM-COMMON-DATA RECADDRESS.
    IF OLDPOS > 0 THEN
        MOVE NAME TO A-NAME(OLDPOS).
        MOVE FIRSTNAME TO A-FIRSTNAME(OLDPOS).
        MOVE CITY TO A-CITY(OLDPOS).
        MOVE STREET TO A-STREET(OLDPOS).
    ENDIF.
    GOBACK.
```

5.3 Objektcallback-Funktionen

Funktionen, die in der Dialogbeschreibung als Objektcallback-Funktionen deklariert sind, müssen in COBOL wie folgt deklariert werden :

```
entry "function1" using
        DM-COMMON-DATA
        DM-OBJECT-DATA.
```

Diese Funktion muss im Dialogskript wie folgt definiert werden:

```
callback cobol function1 ();
```

Anmerkung

Im Gegensatz zur C-Schnittstelle des Dialog Managers, hat die COBOL-Callback-Funktion zwei Parameter.

5.4 Nachlade-Funktionen

Funktionen, die in der Dialogbeschreibung als Nachlade-Funktionen deklariert sind, müssen in COBOL wie folgt deklariert werden:

```
entry "function1" using
        DM-COMMON-DATA
        DM-CONTENT-DATA.
```

Diese Funktion muss im Dialogskript wie folgt definiert werden

```
contentfunc cobol function1 ();
```

Anmerkung

Im Gegensatz zur C-Schnittstelle des Dialog Managers, hat die COBOL-Nachlade-Funktion zwei Parameter.

5.5 Datenfunktionen

Funktionen, die in der Dialogbeschreibung als Datenfunktionen deklariert sind, müssen in COBOL wie folgt deklariert werden:

```
entry "datafunction1" using
  DM-Common-Data
  DM-Datafunc-Data.
```

Die Funktion repräsentiert ein Datenmodell (Model-Komponente) welches die Präsentationsobjekte (View-Komponente) mit Datenwerten versorgt oder diese speichert und verwaltet.

Die Parameter dieser Funktion sind fest vom ISA Dialog Manager vorgegeben und können nicht verändert werden.

Diese Funktion wird aufgerufen, wenn eine Synchronisation zwischen View- und Model-Komponente erforderlich wird. Dies kann entweder automatisch geschehen, entsprechend den Steuerungsoptionen im Attribut *.dataoptions[]* der beteiligten Komponenten. Der Aufruf kann aber auch ausgelöst sein durch explizite Nutzung der Methoden **:apply()**, **:collect()**, **:propagate()** oder **:represent()**. Der Aufruf erfolgt einzeln für jedes Model-Attribut.

Anmerkung

Im Gegensatz zur C-Schnittstelle des ISA Dialog Managers, hat die COBOL-Datenfunktion zwei Parameter.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Kapitel „Datenfunktions-Struktur DM-Datafunc-Data“

5.6 Canvas-Funktionen

Funktionen, die in der Dialogbeschreibung als Canvas-Funktionen deklariert sind, müssen in "C" programmiert werden. COBOL-Unterstützung ist für diese Funktionen nicht verfügbar.

5.7 Format-Funktionen

Funktionen, die in der Dialogbeschreibung als Format-Funktionen deklariert sind, müssen in "C" programmiert werden. COBOL-Unterstützung ist für diese Funktionen nicht verfügbar.

5.8 Übergabe der Funktionsadressen

Damit der Dialog Manager die Funktionen in der Anwendung aufrufen kann, benötigt er die Adressen dieser Funktionen. Um diese Adressen an den Dialog Manager zu übergeben, ist die Struktur **DM_FuncMap** definiert. Diese Struktur kann aber in COBOL nicht verwendet werden, da COBOL nicht in der Lage ist, die Adressen von Funktionen an andere Funktionen zu übergeben. Aus diesem Grund muss die Übergabe der Funktionsadressen in C geschrieben werden. Bei dieser Übergabe der Funktionsadressen müssen zwei Arten von Funktionen unterschieden werden.

5.8.1 Funktionen ohne Records als Parameter

In diesem Fall kann die Übergabe der Funktionsadresse durch das C-Source File, das von dem Programm **gencobfx** generiert wurde, erfolgen. Die Übergabe der Funktionsadresse erfolgt dann durch den Aufruf der generierten C-Funktionen von COBOL-Hauptprogrammen aus.

Dazu sind folgende Schritte notwendig:

1. Definition der Funktionen im Dialogskript
2. Aufbau der Eingabedatei zum Generieren des C-Moduls
3. Generieren des C-Moduls durch **gencobfx**
4. Übersetzen des C-Moduls
5. Aufruf der generierten C-Funktion vom COBOL-Hauptprogramm aus.

5.8.1.1 Aufbau der Funktionsdefinitionsdatei

Die Funktionsdefinitionsdatei, die notwendig ist, um COBOL-Funktionen an den DM zu binden, kann automatisch erzeugt werden. Die Struktur der Eingabedatei ist systemunabhängig; die Art, die C-Source-Datei zu erhalten, ist systemabhängig.

Die Datei muss alle COBOL-Funktionen enthalten, die direkt vom DM aufgerufen werden sollen. Die erste Spalte muss den Namen der Funktion in der Dialogdatei enthalten, die zweite Spalte kann den Namen der Funktion im COBOL-Programm enthalten. Wenn die zweite Spalte für eine Funktion fehlt, nimmt das Skript den ersten Namen, um den Namen der COBOL-Funktion zu erzeugen.

5.8.1.2 Erzeugen der Funktionsdefinitionsdatei

5.8.1.2.1 Unix

Auf UNIX können Sie ein Programm namens **gencobfx** verwenden, um die Funktionsdefinitionsdatei zu erzeugen. Eines der Argumente muss eine Datei sein, die alle Namen der Funktionen enthält. Diese Datei kann Kommentare enthalten; die Zeile, die ein Kommentar sein soll, muss mit einem „*“ beginnen.

Die Syntax des Skripts sieht aus wie folgt:

```
gencobfx -mf|-mfvis [-o outfile] [-f func-entry] descriptionfile
```

-mf

Durch Angabe dieses Parameters wird eine Funktionsdatei erzeugt, die für den Einsatz zusammen mit dem MICRO FOCUS COBOL-Compiler geeignet ist.

-mfvis

Durch Angabe dieses Parameters wird eine Funktionsdatei erzeugt, die für den Einsatz mit MICRO FOCUS VISUAL COBOL geeignet ist.

outfile

Durch Angabe dieses Parameters kann die Funktionstabelle in jeglicher Datei gespeichert werden. Normalerweise übernimmt der DM den Namen der Beschreibungsdatei und ersetzt die Zeichen nach dem Punkt mit „c“.

func-entry

Durch Angabe dieses Parameters können Sie den Namen der Funktion spezifizieren, die von Ihrem COBOL-Hauptprogramm aufgerufen werden muss. Diese Funktion bindet durch Aufrufen von DM_BindCallBacks die Funktionen an den DM. Normalerweise nimmt der DM den Namen „BindFuncs“.

descriptionfile

Diese Datei muss die Namen aller Funktionen enthalten.

5.8.1.2.2 Microsoft Windows

Auf MICROSOFT WINDOWS können Sie ein Programm namens **gencobfx.exe** verwenden, um die Funktionsdefinitionsdatei zu erzeugen. Eines der Argumente muss eine Datei sein, die alle Namen Ihrer Funktionen enthält. Diese Datei kann Kommentare enthalten. Die Zeile, die ein Kommentar sein soll, muss mit einem "*" beginnen.

Die Syntax des Skripts sieht aus wie folgt:

```
gencobfx.exe -mf|-mfvis [-o outfile] [-f func-entry] descriptionfile
```

-mf

Durch Angabe dieses Parameters wird eine Funktionsdatei erzeugt, die für den Einsatz zusammen mit dem MICRO FOCUS COBOL-Compiler geeignet ist.

-mfvis

Durch Angabe dieses Parameters wird eine Funktionsdatei erzeugt, die für den Einsatz mit MICRO FOCUS VISUAL COBOL geeignet ist.

outfile

Durch Angabe dieses Parameters kann die Funktionstabelle in jeglicher Datei gespeichert werden. Normalerweise übernimmt der DM den Namen der Beschreibungsdatei und ersetzt die Zeichen nach dem Punkt mit „c“.

func-entry

Durch die Angabe dieses Parameters können Sie den Namen der Funktion spezifizieren, die von Ihrem COBOL-Hauptprogramm aufgerufen werden muss. Diese Funktion bindet durch Aufrufen von DM_BindCallBacks die Funktionen an den DM. Normalerweise nimmt der DM den Namen „BindFuncs“.

descriptionfile

Diese Datei muss die Namen aller Funktionen enthalten.

5.8.1.3 Beispiel

```
* Function definitions for tabdemo
TABLETEST
LOADTABLE
SAVETABLE
```

Generierte C-Datei

```
#include "IDMuser.h"

extern DML_pascal DM_ENTRY tabletest();
extern DML_pascal DM_ENTRY loadtable();
extern DML_pascal DM_ENTRY savetable();

static struct {
  char *funcname;
  void (*address)();
} bindfuncs_FuncMap[] = {
  { "TABLETEST", (DM_EntryFunc) tabletest },
  { "LOADTABLE", (DM_EntryFunc) loadtable },
  { "SAVETABLE", (DM_EntryFunc) savetable },
};

struct CobolStatus {
  unsigned short errcode;
  unsigned short options;
};

void bindfuncs (cobstatus, dialogID)
struct CobolStatus *cobstatus;
DM_ID *dialogID;
{
  cobstatus->errcode =
      !DM_BindCallBacks(
          bindfuncs_FuncMap,
          sizeof(bindfuncs_FuncMap) / sizeof(bindfuncs_FuncMap[0]),
          *dialogID,
          cobstatus->options);
  cobstatus->options = 0;
}
```

5.8.1.4 BindThruLoader-Funktionalität

Wenn **BindThruLoader** benutzt wird, dann wird **gencobfx** nicht mehr benötigt und der Aufruf von **BindFuncs** kann entfallen.

Eingeschaltet wird die Funktionalität mit:

» `CALL "DMufcob_BindThruLoader" USING DM-STDARGS DIALOG-ID.`
für MICRO FOCUS COBOL auf UNIX-Systemen

» `CALL "DMmfviscob_BindThruLoader" USING DM-STDARGS DIALOG-ID.`
für MICRO FOCUS VISUAL COBOL

Alternativ kann auch folgende Version verwendet werden, die portabler ist:

```
77  ACTION  PIC 9(4)  BINARY VALUE 0.

MOVE DMF-CobolBindForLoader TO ACTION.
CALL "DMcob_Control" USING DM-STDARGS DIALOG-ID ACTION.
```

Auch wenn der Aufruf von **DMcob_Control** für alle COBOL-Varianten möglich ist, wird es Anbindungen geben, bei denen die dynamische Bindung dennoch nicht funktioniert.

Hinweis

Wenn im Dialog ein ***Application***-Objekt verwendet wird, dann muss die Funktionalität am ***Application***-Objekt und nicht am Dialog eingeschaltet werden:

```
77  APPL-ID  PIC 9(9)  BINARY VALUE 0.

CALL "DMcob_PathToID" USING DM-STDARGS APPL-ID NULL-OBJECT "Appl@".
```

Diese APPL-ID muss an Stelle der DIALOG-ID (siehe oben) angegeben werden.

Verfügbarkeit

DMF-CobolBindForLoader ist ab IDM-Version A.06.01.d verfügbar

5.8.2 Funktionen mit Records als Parametern

In diesem Fall erfolgt die Übergabe der Funktionsadressen automatisch durch den Aufruf der Initialisierungsfunktion in dem Recordmodul.

Dazu sind folgende Schritte notwendig:

1. Definition der Records im Dialogskript
2. Definition der Funktionen im Dialogskript
3. Generieren des C-Moduls durch den Simulator mit der Option **+writetrampolin**
4. Übersetzen des C-Moduls
5. Aufruf der generierten C-Funktion vom COBOL-Hauptprogramm aus.
6. Linken des Programms

5.8.2.1 Generierung des Trampolin-Moduls

Um Funktionen für Dialoge mit Records mit einer Applikation versehen zu können, müssen vom Dialog Manager generierte C-Module übersetzt und dazu gebunden werden. Diese Generierung ist für jedes im Dialog vorhandene Modul notwendig, das Definitionen für Funktionen beinhaltet. Die Generierung dieser Module erfolgt durch Aufruf der Simulation mit der Option **+writetrampolin**:

```
idm +writetrampolin <outfile> <dialogskript>
```

Dabei kann das angegebene Dialogskript ein Modul sein.

Dieses Statement generiert aus einem Dialogskript die notwendigen Module zum Aufruf von Funktionen. Je nach Art der Funktionen, die solche Records verwenden, werden die entsprechenden Header-Dateien (C und/oder COBOL) erzeugt.

Bei der Implementierung dieser Funktionen muss in das entsprechende Modul die erzeugte Header-Datei eingebunden werden. Dies geschieht durch das Statement:

```
COPY
```

Wenn solche Funktionen und Records in einer Applikation verwendet werden, müssen diese Records durch den Aufruf der Funktion

```
cobRecMInit<Modulname>
```

initialisiert werden.

Die Initialisierung sollte üblicherweise vor dem Funktionsaufruf BindFuncs erfolgen!

Werden solche Strukturen in Dialog Manager-Applikationen verwendet, wird also der verteilte Dialog Manager eingesetzt, muss noch zusätzlich die Applikation angegeben werden, für die die Dateien generiert werden sollen.

Der Name der Initialisierungsfunktion wird dann aus dem Namen der Applikation gebildet.

Also muss

```
cobRecMInit<ApplicationName>
```

aufgerufen werden.

```
idm +writetrampolin <contfile> +application <ApplicationName> <dialogscript>
```

Soll eine Copy-Strecke generiert werden, die auch *National Character* beinhaltet (nur MICRO FOCUS VISUAL COBOL), dann muss dies durch die Option **-mfviscob-u** angefordert werden.

5.8.2.2 Übersetzung des C-Moduls

Bei der Übersetzung der generierten C-Modul:Übersetzunge muss je nach eingesetztem COBOL ein Flag gesetzt und bei der Kompilierung des C-Moduls angegeben werden.

COBOL Compiler	Flag für C Compiler
MICRO FOCUS COBOL	-DUFCOB
MICRO FOCUS VISUAL COBOL	-DMFVISCOB

Auf den PC-Plattformen unter MICROSOFT WINDOWS muss bei Verwendung des MICRO FOCUS COBOL-Compilers zusätzlich der Schalter `-DUFCOB_LINK` gesetzt sein.

5.8.2.3 Beispiel

Dialogdatei

```
dialog RecTest
{
  .reffont MyFont;
}

function cobol void WriteAddr(record Address input);
function cobol void ReadAddr(record Address output, integer Pos input
output);

font MyFont "fixed";

record Address
{
  string[25] Name       shadows W1.Lname.E.content;
  string[15] FirstName  shadows W1.Lfirstname.E.content;
  string[25] City       shadows W1.Lcity.E.content;
  string[30] Street     shadows W1.Lstreet.E.content;
}
```

Generierte COBOL Copy-Datei

```
01 RecAddress.
    02 Name pic X(25).
    02 FirstName pic X(15).
    02 City pic X(25).
    02 Street pic X(30).
```

Generiertes C-Programm

```
#include <IDMuser.h>
#include <IDMcobol.h>
#if !defined(offsetof)
#    define offsetof(TYPE,FLD)    ((int) &(((TYPE *) 0)->FLD))
#endif
```

```
#ifdef VISCOB
#    ifdef DOS
#        define WriteAddr WRITEADDR
#        define ReadAddr READADDR
#    else
#        define WriteAddr writeaddr
#        define ReadAddr readaddr
#    endif
#endif

#ifdef UFCOB
#    ifdef UFCOB_LINK
#        define WriteAddr WRITEADDR
#        define ReadAddr READADDR
#    else
#        define WriteAddr "WRITEADDR"
#        define ReadAddr "READADDR"
#    endif
#endif

typedef struct {
    char        Name[25];
    char        FirstName[15];
    char        City[25];
    char        Street[30];
} RecCobAddres;

static DM_RecordInfo RecCobInfoAddres[5] =
{
    {".Name",DT_string, offsetof(RecCobAddres,Name),25,0},
    {".FirstName",DT_string,offsetof(RecCobAddres,FirstName),15,0},
    {".City",DT_string,offsetof(RecCobAddres,City),25,0},
    {".Street",DT_string,offsetof(RecCobAddres,Street),30,0},
};

DeclCobol(WriteAddr)

static void DML_c DM_ENTRY RecFuncWriteAddr __1(
(DM_ID, arg0))
{
    RecCobAddress record0;

    DM_GetRecord(arg0,&record0,RecCobInfoAddres,
        sizeof(RecCobInfoAddres)/sizeof(DM_RecordInfo),
        DMF_Cobol);
```

```
    CallCobol1(;,WriteAddr,&record0);
}

DeclCobol(ReadAddr)

static void DML_c DM_ENTRY RecFuncReadAddr __2(
(DM_ID, arg0),
(DM_Integer*, Pos))
{
    RecCobAddress record0;

    CallCobol2(;,ReadAddr,&record0, Pos);
    DM_SetRecord(arg0,&record0,RecCobInfoAddres,
        sizeof(RecCobInfoAddres)/sizeof(DM_RecordInfo),
        DMF_Cobol);
}

#define RecMapCount (sizeof(RecMap) / sizeof(RecMap[0]))

static DM_FuncMap RecMap[] = {
    { "WriteAddr", (DM_EntryFunc) RecFuncWriteAddr },
    { "ReadAddr", (DM_EntryFunc) RecFuncReadAddr },
};

boolean RecInitMRecTest __1((DM_ID, dialog))
{
    return DM_BindFunctions(RecMap, RecMapCount, dialog, 0, DMF_Silent);
}
void cobrecminitrectest __2((DM_ID*,  dialog), (boolean *, result))
{
    *result = RecInitMRecTest((DM_ID) *dialog);
}
```

5.8.3 Dynamische Anbindung von Record-Funktionen

Verfügbarkeit

Ab IDM-Version A.06.01.d

Mit „dynamischer Anbindung“ ist die Bindungsart von Anwendungsfunktionen gemeint, welche kein explizites Setzen des Funktionspointers der Anwendungsfunktionen über die Funktionen DM_BindFunctions oder DM_BindCallBacks benötigt. Dazu gehören z.B. die Anbindung von Funktionen aus dynamischen Bibliotheken (Transport *"dynlib"*) oder der Aufruf von COBOL-Funktionen über ihren Namen durch die „BindThruLoader-Funktionalität“. Da allerdings Anwendungsfunktionen mit **Record-Parametern** eine zusätzliche Stub-Funktion für das Umwandeln (Marshalling) der Record-Struktur benötigen, wurde bisher beim Aufruf dieser Record-Funktionen keine Struktur übertragen.

Ab IDM-Version A.06.01.d wird für dynamisch angebundene Record-Funktionen die Struktur der Record-Parameter ohne Stub-Funktion übertragen. Eine Nutzung solcher Record-Funktionen ist möglich, ohne C- oder COBOL-Code zu generieren. Es ist aber sinnvoll sich die Funktionsprototypen wie auch die Strukturdefinition über **+writeheader <basefilename>** generieren zu lassen.

Beispiel

```
dialog D
record RecAdr {
  string[80] Name := "";
  string[120] Strasse := "";
  string[40] Ort := "";
  integer PLZ := 0;
}

application Appl {
  .transport "dynlib";
  .exec "libadr.so";
  .active true;
  function boolean Search(string Pattern, record RecAdr output);
}

application ApplCobol {
  .local true;
  .active true;
  function cobol boolean New(record RecAdr) alias "NEWADR";
}

on dialog start {
  if not Appl.active orelse not ApplCobol.active then
    print "Applikation(en) nicht korrekt angebunden!";
  elseif not Search("Udo", RecAdr) then
    print "Udo nicht gefunden, f\374ge ihn hinzu";
    RecAdr.Name := "Udo";
    RecAdr.Strasse := "Meisenweg 7";
    RecAdr.Ort := "Wolkenstein";
    RecAdr.PLZ := 71723;
    New(RecAdr);
  else
    print "Udo gefunden, wohnt in: "+RecAdr.Ort;
  endif
  exit(0);
}
```

Die Generierung der Prototypen kann nun anstatt mit **+writeproto**, **+writefuncmap** und **+/-writetetrampolin** einfach mit **+writeheader** erfolgen:

```
pidm adr.dlg +application Appl +writeheader adr       // erzeugt adr.h
pidm adr.dlg +application ApplCobol +writeheader adr  // erzeugt adr.cpy
```

Die eigentliche Implementierung in C und COBOL kann nun die Funktionsprototypen bzw. Record-Definitionen für „RecRecAdr“ in **adr.h** (C-Headerdatei) und **adr.cpy** (COBOL Copy-Strecke) finden.

Soll eine Copy-Strecke generiert werden, die auch National Character beinhaltet (nur MICRO FOCUS VISUAL COBOL), dann muss dies durch die Option **-mfviscob-u** angefordert werden.

6 Funktionen der COBOL-Schnittstelle des DM

In diesem Kapitel werden die Funktionen der DM-Schnittstelle beschrieben. Nach einer kurzen Zusammenfassung (Kapitel „Übersicht über die Funktionen“) werden die verschiedenen Funktionsaufgaben erläutert (Kapitel „Initialisieren und Starten des DM“ bis „Spezielle Funktionen“). Im Kapitel „Funktionen in alphabetischer Reihenfolge“ finden Sie eine vollständige Liste mit den detailliert beschriebenen Funktionen in alphabetischer Reihenfolge.

Beim Beschreiben der Parameter werden folgende Zeichen verwendet:

-> Eingabeparameter

<- Ausgabeparameter

<-> Ein- und Ausgabeparameter

6.1 Übersicht über die Funktionen

Funktionsname	Kurzbeschreibung
COBOLAPPFINISH	Endefunktion in COBOL in einer verteilten Umgebung.
COBOLAPPINIT	Startfunktion in COBOL in einer verteilten Umgebung.
COBOLMAIN	COBOL-Hauptprogramm einer lokalen Anwendung.
DMcob_CallFunction	Aufruf einer Funktion in einem beliebigen Anwendungsteil (verteilte Anwendung).
DMcob_CallMethod	Aufruf von Methoden mit Parametern von der Anwendung aus.
DMcob_CallRule	Aufruf einer Regel mit Parametern.
DMcob_Control	Aktion auslösen.
DMcob_CreateFromModel	Erzeugt ein Objekt, das von einem Modell abgeleitet ist.
DMcob_CreateObject	Erzeugt ein Objekt einer angegebenen Klasse.
DMcob_DataChanged	Signalisiert, dass sich an einem Datenmodell (Model-Komponente) der Wert des angegebenen Attributs (Model-Attribut) geändert hat.
DMcob_DeleteArgString	Löschen von Argument, das an Programm übergeben wurde.

Funktionsname	Kurzbeschreibung
DMcob_Destroy	Löschen von Objekten.
DMcob_DialogPathToID	Umwandlung von externem Objektnamen in interne DM-ID.
DMcob_ErrMsgText	Zum Errorcode gehörender Text.
DMcob_EventLoop	Starten der Dialogabarbeitung.
DMcob_ExecRule	Starten benannter Regeln.
DMcob_FatalAppError	Fehlerbehandlung.
DMcob_FreeVector	Freigeben eines temporären Speicherbereiches.
DMcob_GetArgCount	Erfragen von Anzahl der an das Programm übergebenen Argumente.
DMcob_GetArgString	Erfragen des Arguments, das an Programm übergeben wurde.
DMcob_GetValue	Erfragen von DM-Objektattributen.
DMcob_GetValueBuff	Erfragen von DM-Objektattributen mit beliebigem Übergabebereich.
DMcob_GetVector	Abfragen eines vektoriellen Attributes in einen temporären Speicherbereich.
DMcob_GetVectorValue	Setzen eines Wertes in einem temporären Speicherbereich.
DMcob_Initialize	Initialisieren von DM.
DMcob_InitVector	Anlegen eines temporären Speicherbereiches.
DMcob_LGetValueBuff	Erfragen von DM-Objektattributen mit beliebigem Übergabebereich.
DMcob_LGetValueLBuff	Abfragen von Objektattributen mit beliebigem Übergabebereich und Indexwerten bis 65535.
DMcob_LGetVector	Abfragen eines vektoriellen Attributes in einen temporären Speicherbereich.
DMcob_LGetVectorValue	Setzen eines Wertes in einem temporären Speicherbereich.
DMcob_LGetVectorValueBuff	Abfragen vektorieller Attribute mit beliebigem Übergabebereich und Indexwerten bis 65535.
DMcob_LInitVector	Anlegen eines temporären Speicherbereiches.

Funktionsname	Kurzbeschreibung
DMcob_LoadDialog	Laden von Dialog in DM.
DMcob_LoadProfile	Laden von benutzerdefinierten Werten aus dem Profile.
DMcob_LSetValueBuff	Verändern von DM-Objektattributen mit beliebigem Übergabebereich.
DMcob_LSetValueLBuff	Ändern von Objektattributen mit beliebigem Übergabebereich und Indexwerten bis 65535.
DMcob_LSetVector	Zuweisen eines temporären Speicherbereiches an ein Objekt.
DMcob_LSetVectorValue	Abfragen eines Wertes in einem temporären Speicherbereich.
DMcob_LSetVectorValueBuff	Ändern vektorieller Attribute mit beliebigem Übergabebereich und Indexwerten bis 65535.
DMcob_OpenBox	Öffnen einer Messagebox oder Dialogbox.
DMcob_OpenBoxBuff	Öffnen einer Messagebox oder Dialogbox mit einem Übergabebereich in definierbarer Größe.
DMcob_PathToID	Umwandlung von externem Objektnamen in interne DM-ID.
DMcob_PutArgString	Ersetzen von Argument, das an Programm übergeben wurde.
DMcob_QueryBox	Öffnen einer Messagebox.
DMcob_QueryError	Erfragen von Fehler.
DMcob_QueueExtEvent	Verschicken eines externen Ereignisses.
DMcob_ResetValue	Zurücksetzen von DM-Objektattributen auf Modell- oder Default-Wert.
DMcob_SetValue	Verändern von DM-Objektattributen.
DMcob_SetValueBuff	Verändern von DM-Objektattributen mit beliebigem Übergabebereich.
DMcob_SetVector	Zuweisen eines temporären Speicherbereiches an ein Objekt.
DMcob_SetVectorValue	Abfragen eines Wertes in einem temporären Speicherbereich.
DMcob_ShutDown	Kontrolliertes Beenden der Anwendung.
DMcob_StartDialog	Starten der eigentlichen Dialoganwendung.
DMcob_StopDialog	Anhalten eines Dialoges.

Funktionsname	Kurzbeschreibung
DMcob_TraceMessage	Schreiben von Protokollmeldungen der Applikation in die DM-Protokolldatei.
DMcob_ValueChange	Ersetzen des Gesamtwerts oder eines Elementwerts in einer Sammlung.
DMcob_ValueChangeBuffer	Ersetzen des Gesamtwerts oder eines Elementwerts in einer Sammlung mit einem Übergabebereich in definierbarer Größe.
DMcob_ValueCount	Liefert die Anzahl der Werte (ohne die Standardwerte), den Indextyp oder den höchsten Indexwert einer Sammlung zurück.
DMcob_ValueGet	Holt aus einer Sammlung einen einzelnen Elementwert an einem definierten Index.
DMcob_ValueGetBuffer	Holt aus einer Sammlung einen einzelnen Elementwert an einem definierten Index, mit einem Übergabebereich in definierbarer Größe.
DMcob_ValueGetType	Abfragen des Datentyps eines vom IDM verwalteten Wertes.
DMcob_ValueIndex	Ermitteln des zu einer Position gehörenden Index einer Sammlung.
DMcob_ValueInit	Umwandeln einer Wertereferenz in eine vom IDM gemanagte lokale oder globale Wertereferenz.
DMmfviscob_BindThruLoader	Markieren aller im Dialog definierten Funktionen, dass diese über das COBOL-Runtimesystem aufgerufen werden sollen und daher nicht zu der Anwendung hinzu gelinkt worden sind (MICRO FOCUS VISUAL COBOL).
DMufcob_BindThruLoader	Markieren aller im Dialog definierten Funktionen, dass diese über das COBOL-Runtimesystem aufgerufen werden sollen und daher nicht zu der Anwendung hinzu gelinkt worden sind MICRO FOCUS Server Express).

6.2 Initialisieren und Starten des DM

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

» DMcob_EventLoop
Diese Funktion startet die Abarbeitung im Dialog Manager. Ohne diesen Aufruf wäre keine Interaktion mit der Benutzeroberfläche möglich.

- DMcob_Initialize
  Diese Funktion initialisiert den DM.
- DMcob_LoadDialog
  Diese Funktion lädt einen Dialog in den DM. Diese Dialogbeschreibung kann eine ASCII- oder eine Binärdatei sein.
- DMcob_LoadProfile
  Mit Hilfe dieser Funktion können benutzerdefinierte Werte geladen und im Dialog verarbeitet werden.
- DMcob_StartDialog
  Diese Funktion startet den Dialog. Sie initialisiert das Fenstersystem und macht die Fenster sichtbar, die in der Datei als sichtbar definiert sind. Sie führt auch die Dialog-Start-Regel aus.
- DMmfviscob_BindThruLoader
  Mit dieser Funktion erfolgt die Initialisierung, damit das COBOL-Laufzeitsystem Funktionen ohne eine Funktionstabelle verwenden kann. Diese Funktion ist für MICRO FOCUS VISUAL COBOL vorgesehen.
- DMufcob_BindThruLoader
  Mit dieser Funktion erfolgt die Initialisierung, damit das COBOL-Laufzeitsystem Funktionen ohne eine Funktionstabelle verwenden kann. Diese Funktion ist für MICRO FOCUS Server Express vorgesehen.

6.3 Beenden des ISA Dialog Managers

Mit diesen Funktionen können die Ausführung eines Dialogs und des IDM kontrolliert beendet werden:

- DMcob_ShutDown
  Diese Funktion beendet den IDM vollständig und macht globale Initialisierungen rückgängig. Sie wird normalerweise nur benötigt, wenn ein anwendungsspezifisches **Main**-Programm verwendet wird.
- DMcob_StopDialog
  Mit dieser Funktion wird der aktuelle oder ein bestimmter Dialog beendet. Die `on dialog finish` Regel wird ausgeführt.

6.4 Zugriffsfunktionen

Mit Hilfe der folgenden Funktionen können Sie die Objekte des Dialog Managers und deren Attribute manipulieren.

6.4.1 Zugriff auf DM-Identifikatoren

Um auf den DM-Identifikator für ein Objekt zugreifen zu können, benötigen Sie den Namen des Objekts. Zusammen mit diesem Namen und einem Aufruf der folgenden Funktionen, erhalten Sie den

DM-internen Identifikator des Objekts.

- DMcob_DialogPathToID
  Diese Funktion gibt auch den Dialog Manager Identifikator eines Objektes zurück. Diese Funktion kann jedoch mehr als einen Dialog verarbeiten.
- DMcob_PathToID
  Diese Funktion gibt den Identifikator des Objektes zurück. Dieser Identifikator muss für alle anderen Aufrufe an den Dialog Manager benutzt werden.

6.4.2 Zugriff auf Objektattribute

Um auf irgendein Objektattribut zugreifen zu können, brauchen Sie den Identifikator, den Datentyp und den Namen der gewünschten Attribute.

- DMcob_DataChanged
  Mit dieser Funktion kann signalisiert werden, dass sich an einem bestimmten Datenmodell (Model-Komponente) der Wert des angegebenen Attributs (Model-Attribut) geändert hat.
- DMcob_GetValue
  Diese Funktion gibt den Wert des gewünschten Attributes von jedem Objekt zurück.
- DMcob_GetValueBuff
  Diese Funktion gibt den String-Wert eines Attributes in einem separaten Puffer zurück.
- DMcob_LGetValueBuff
  Diese Funktion dient zum Abfragen des Wertes eines Objektattributs. Für die Übergabe von Strings wird ein Puffer genutzt. Es können Indexwerte bis *65.535* übergeben werden.
- DMcob_LGetValueLBuff
  Diese Funktion dient zum Abfragen des Wertes eines Objektattributs. Für die Übergabe von Strings wird ein Puffer mit bis zu *65.535* Zeichen genutzt. Es können Indexwerte bis *65.535* übergeben werden.
- DMcob_ResetValue
  Diese Funktion setzt das Attribut auf den Wert des Objektmodells oder Objektdefault zurück.
- DMcob_SetValue
  Diese Funktion verändert das Attribut zu dem übergebenen Wert.
- DMcob_SetValueBuff
  Diese Funktion verändert die Attribute zu dem übergebenen Wert. Sie verlangt einen speziellen Puffer, der jede gewünschte Länge haben kann.
- DMcob_LSetValueBuff
  Diese Funktion dient zum Ändern des Wertes eines Objektattributs. Für die Übergabe von Strings wird ein Puffer genutzt. Es können Indexwerte bis *65.535* übergeben werden.
- DMcob_LSetValueLBuff
  Diese Funktion dient zum Ändern des Wertes eines Objektattributs. Für die Übergabe von Strings wird ein Puffer mit bis zu *65.535* Zeichen genutzt. Es können Indexwerte bis *65.535* übergeben werden.

6.4.3 Erzeugen und Zerstören von Objekten

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

- DMcob_CreateFromModel
  Diese Funktion erzeugt ein Objekt auf Basis eines Modells und gibt den Identifikator des erzeugten Objekts zurück.
- DMcob_CreateObject
  Diese Funktion erzeugt ein Objekt einer spezifizierten Klasse und gibt seinen Identifikator zurück. Nachdem das Objekt erfolgreich erzeugt wurde, kann von einer DM-Funktion darauf zugegriffen werden.
- DMcob_Destroy
  Diese Funktion zerstört jedes Objekt. Nach einem Aufruf dieser Funktion kann keine DM-Funktion mehr auf diese Objekte zugreifen.

6.4.4 Dienstprogramme

- DMcob_CallFunction
  Funktion um beliebige, dem Dialog Manager bekannte, Funktionen in anderen Teilen der Anwendung aufzurufen.
- DMcob_CallMethod
  Mit Hilfe dieser Funktion können Methoden mit Parametern von der Anwendung aufgerufen werden.
- DMcob_CallRule
  Mit Hilfe dieser Funktion können Regeln mit Parametern aus COBOL heraus aufgerufen werden. Im Unterschied zu **DMcob_ExecRule** werden Parameter und Rückgabewerte der Regel mit übergeben und können daher auch verarbeitet werden.
- DMcob_Control
  Durch diese Funktion kann die Anwendung den Bildschirm neu aufbauen, das gegenwärtig benutzte Codeset umschalten oder die Tastatur sperren.
- DMcob_ExecRule
  Diese Funktion startet die Abarbeitung der spezifizierten benannten Regel, als ob es von "perform" in einer anderen Regel aufgerufen worden wäre.
- DMcob_OpenBox
  Funktion zum Öffnen einer Messagebox oder Dialogbox.
- DMcob_OpenBoxBuff
  Funktion zum Öffnen einer Messagebox oder Dialogbox mit einem Übergabebereich in definierbarer Größe.
- DMcob_QueryBox
  Mit Hilfe dieser Funktion können Messageboxes geöffnet werden.
- DMcob_QueueExtEvent

Diese Funktion erlaubt es, externe Ereignisse zu erzeugen und zur Verarbeitung an den Dialog Manager zu übergeben.

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

6.4.5 Zugriff auf Listbox und Tablefield

Mit Hilfe der im folgenden beschriebenen Funktionen können Listbox- und Tablefieldinhalte effizient bearbeitet werden. Insbesondere beim Einsatz des verteilten Dialog Managers sollten diese Funktionen zum Setzen und Abfragen von Listbox- und Tablefieldinhalten benutzt werden.

» DMcob_FreeVector
Freigeben eines temporären Speicherbereiches im Dialog Manager.

» DMcob_GetVector
Mit Hilfe dieser Funktion kann ein beliebiger Vektor von Attributwerten eines Objektattributes mit einem Funktionsaufruf abgefragt werden.

» DMcob_GetVectorValue
Mit Hilfe dieser Funktion kann ein Element eines Vektors von Objekt-Attributwerten erfragt werden.

» DMcob_InitVector
Diese Funktion legt einen temporären Speicherbereich im Dialog Manager an, der dann beliebig viele Objekt-Attributwerte eines Objektes aufnehmen kann.

» DMcob_LGetVector
Diese Funktion kopiert ein vektorielles Objektattribut ganz oder teilweise in einen temporären Speicherbereich. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_LGetVectorValue
Mit dieser Funktion werden Werte aus einem temporären Speicherbereich geholt. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_LGetVectorValueBuff
Mit dieser Funktion werden Werte aus einem temporären Speicherbereich geholt. Zum Holen von Strings stehen definierbare Puffer zur Verfügung. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_LInitVector
Diese Funktion legt einen temporären Speicherbereich für Werte eines vektoriellen Objektattributs an. Der Bereich kann bis zu *65.535* Werte aufnehmen.

» DMcob_LSetVector
Mit Hilfe dieser Funktion wird ein temporärer Speicherbereich an ein Objekt übergeben. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_LSetVectorValue
Mit Hilfe dieser Funktion wird in einem temporären Bereich ein Objekt-Attributwert gespeichert. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_LSetVectorValueBuff
Mit dieser Funktion können Werte in einem temporären Speicherbereich gesetzt werden. Für Strings stehen Puffer mit bis zu *65.535* Zeichen zur Verfügung. Es können Indexwerte bis *65.535* verwendet werden.

» DMcob_SetVector
Diese Funktion weist einem vektoriellen Objektattribut einen temporären Speicherbereich zu.

» DMcob_SetVectorValue
Diese Funktion ändert einen Wert in einem temporären Speicherbereich.

6.5 Bearbeitung von Sammlungen

» DMcob_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.

» DMcob_ValueChangeBuffer
Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung. Im Unterschied zu **DMcob_ValueChange** kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.

» DMcob_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.

» DMcob_ValueGet
Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört.

» DMcob_ValueGetBuffer
Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört. Im Unterschied zu **DMcob_ValueGet** kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.

» DMcob_ValueGetType
Diese Funktion dient zur Abfrage des Datentyps eines vom IDM verwalteten Wertes.

» DMcob_ValueIndex
Mit dieser Funktion kann der zu einer Position zugehörige Index einer Sammlung ermittelt werden.

» DMcob_ValueInit
Mit dieser Funktion kann ein Wert in eine vom IDM gemanagte lokale oder globale Wertereferenz umgewandelt werden. Dadurch ist die weitere Manipulation des Wertes durch **DMcob_Value…()**-Funktionen möglich sowie die Rückgabe als Parameter bzw. Rückgabewert.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

6.6 Fehlerbehandlung

Wenn ein Funktionsaufruf an den DM fehlschlägt, kann die Applikation den Fehler mit Hilfe der folgenden Dienstprogramme erfragen.

- DMcob_ErrMsgText
  Diese Funktion gibt den Textstring an einen Fehlercode zurück.
- DMcob_FatalAppError
  Diese Funktion sollte aufgerufen werden, wenn die Anwendung einen schwerwiegenden Fehler entdeckt und nicht weiterarbeiten kann. Sie beendet jegliches Arbeiten im Dialog Manager.
- DMcob_QueryError
  Diese Funktion gibt die Anzahl von aktuellen Fehlern zusammen mit den Fehlercodes zurück.

6.6.1 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 zu extrahieren, müssen Sie das Datenfeld *DM-error-severity* der Struktur **DM-ErrorDetail** verwenden.

Die folgenden Fehlergrade sind möglich:

DM-SeveritySuccess	es liegt kein Fehler vor
DM-SeverityWarning	Warnung
DM-SeverityError	Fehler
DM-SeverityFatal	schwerwiegender Fehler, DM kann nicht weiterarbeiten
DM-SeverityProgErr	Programmfehler

Um das Modul zu extrahieren, das den Fehler verursacht hat, kann die Datenstruktur **DM-error-package** benutzt werden.

Die folgenden Module sind möglich:

DM-ModuleIDM	der Fehler ist innerhalb des DM aufgetreten
DM-ModuleUnix	der Fehler ist bei einem Funktionsaufruf an das Betriebssystem Unix aufgetreten

Der Fehlercode selbst ist in der Datenstruktur **DM-error-code**.

Die möglichen Fehlercodes sind im **IDMcobws.cob** definiert.

6.7 Spezielle Funktionen

Mit den folgenden Funktionen können Sie auf die Argumente, die an das Programm übergeben wurden, zugreifen und sie manipulieren.

- DMcob_DeleteArgString
  Diese Funktion löscht ein Argument.
- DMcob_GetArgCount
  Diese Funktion gibt die Anzahl der Argumente zurück, die an das Programm übergeben wurden.
- DMcob_GetArgString
  Diese Funktion gibt den Argumentstring eines beliebigen Arguments zurück.
- DMcob_PutArgString
  Diese Funktion ersetzt ein Argument durch einen neuen Wert.

6.8 Funktionen in alphabetischer Reihenfolge

6.8.1 COBOLMAIN

Diese Funktion ist die Hauptfunktion der Applikation. Sie wird vom DM sofort nach dem Programmstart aufgerufen. Diese Funktion sollte die notwendigen Funktionsaufrufe an die Applikation und an den DM erledigen. Sie bekommt einen Parameter, in dem sie Fehlercodes an die Umgebung zurückgeben kann.

```
COBOLMAIN using Exit-Status.
```

Beispiel

```
identification division.
program-id. COBOLMAIN.

data division.
working-storage section.
copy "IDMcobws.cob".

77  DM-dialogid    pic 9(9) binary value 0.
77  func-name      pic X(30) value spaces.

linkage section.
01  exit-status pic 9(4) binary.

procedure division using exit-status.
startup section.

initialize-IDM.
  move "@" to DM-setsep.
  call "DMcob_Initialize" using DM-StdArgs
        DM-Common-Data.
  perform error-check.

load-dialog.
  call "DMcob_LoadDialog" using DM-StdArgs DM-dialogid
         by content "cobol.dlg@".
  perform error-check.

start-dialog.
  call "DMcob_StartDialog" using DM-StdArgs DM-dialogid.
  perform error-check.

event-loop.
  call "DMcob_EventLoop" using DM-StdArgs.
  perform error-check.
```

```
dialog-done.
  display "Application finishes successfully.".
  goback.

error-check.
  if DM-ErrorOccurred
    display "Error occurred in " func-name upon sysout
    display "Application terminates due to error."
    move 1 to exit-status
    goback.
```

Wenn eine Anwendung nicht direkt mit dem Displayteil des Dialog Manager zusammen gelinkt wird, d.h., sie arbeitet in einer verteilten Umgebung, müssen anstatt der Funktion

COBOLMAIN

die Funktionen COBOLAPPINIT und COBOLAPPFINISH

benutzt werden.

Siehe auch

Objekt Application

Handbuch „Distributed Dialog Manager (DDM)“

6.8.1.1 COBOLAPPINIT

Mit Hilfe dieser Funktion wird die Anwendung gestartet. Sie muss dann die für die Initialisierung notwendigen Schritte durchführen. Diese Funktion wird nur in der verteilten Version benutzt.

```
77 EXIT-STATUS pic 9(4) binary.
77 DM-APPL-ID pic 9(9) binary.
77 DM-DIALOGID pic 9(9) binary.

COBOLAPPINIT using EXIT-STATUS DM-APPL-ID DM-DIALOGID.
```

Parameter

-> EXIT-STATUS

Rückgabewert der COBOL-Funktion. Wenn der Wert auf DM-success gesetzt wird, war die Initialisierung erfolgreich. Ein Wert ungleich DM-success bedeutet, dass die Initialisierung nicht fehlerfrei durchgeführt werden konnte.

-> DM-APPL-ID

Identifikator der Applikation, die initialisiert werden soll.

-> DM-DIALOGID

Identifikator des Dialoges, zu dem die Applikation gehört.

Diese Funktion muss DM-success zurückgeben, wenn die Initialisierung erfolgreich durchgeführt werden konnte. Wenn die Initialisierung nicht erfolgreich war, sollte ein Fehlercode zurückgegeben werden.

Beispiel

```
identification division.
program-id. COBOLAPPINIT.

data division.
working-storage section.
copy "IDMcobws.cob".

linkage section.
01  exit-status        pic 9(4) binary.
77  DM-appl-id        pic 9(9) binary.
77  DM-dialogid        pic 9(9) binary.

procedure division using exit-status DM-appl-id
    DM-dialog-id.
startup section.

initialize-IDM.
  call "DMcob_Initialize" using DM-StdArgs
    DM-Common-Data.
  PERFORM ERROR-CHECK.

BIND-FUNCTIONS.
  Call "BindFuncs" USING DM-STDARGS DM-APPL-ID
  PERFORM ERROR-CHECK.
  MOVE DM-SUCCESS TO EXIT-STATUS.
  GOBACK.
```

Achtung

Die Funktion COBOLAPPINIT muss DMcob_Initialize aufrufen, um ihr die DM-Common-Data zu übergeben!

6.8.1.2 COBOLAPPFINISH

Mit Hilfe dieser Funktion wird die Anwendung in einer verteilten Umgebung beendet.

```
77 EXIT-STATUS pic 9(4) binary.
77 DM-APPL-ID pic 9(9) binary.
77 DM-DIALOGID pic 9(9) binary.

COBOLAPPFINISH using  EXIT-STATUS  DM-APPL-ID DM-DIAOGID.
```

Parameter

<- EXIT-STATUS

Rückgabewert der Funktion. Wenn dieser Wert DM-success ist, wurde die Anwendung erfolgreich beendet. Jeder andere Wert bedeutet hier einen Fehlercode. Daher muss dieser Wert immer von der Anwendung gesetzt werden.

-> DM-APPL-ID

Identifikator der Applikation, die beendet werden soll.

-> DM-DIALOGID

Identifikator des Dialoges, zu dem die Applikation gehört.

Beispiel

```
Entry COBOLAPPFINISH using Exit-StatusDM-Appl-Id
    DM-dialogid.
finish-IDM.
  MOVE DM-SUCCESS TO EXIT-STATUS.
  GoBack.
```

6.8.2 DM_BindCallBacks

Informationen zu dieser Funktion finden Sie in der Beschreibung von DM_BindCallBacks im Handbuch „C-Schnittstelle - Funktionen“.

6.8.3 DM_BindFunctions

Informationen zu dieser Funktion finden Sie in der Beschreibung von DM_BindFunctions im Handbuch „C-Schnittstelle - Funktionen“.

6.8.4 DM_BootStrap

Informationen zu dieser Funktion finden Sie in der Beschreibung von DM_BootStrap im Handbuch „C-Schnittstelle - Funktionen“.

6.8.5 DMcob_CallFunction

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

```
77  DM-funcID pic 9(9) binary.

call "DMcob_CallFunction" using
            DM-StdArgs
            DM-Value
            DM-funcID
            DM-ValueArray.
```

Parameter

<- DM-Value

In diesem Parameter wird der Rückgabewert der Funktion zurückgegeben, falls der Funktionsaufruf durchgeführt werden konnte. Um auf diesen Wert zugreifen zu können, muss zuerst der Datentyp überprüft und entsprechend diesem Wert auf die Struktur zugegriffen werden.

-> DM-funcID

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

<-> DM-ValueArray

Dieser Parameter ist ein Array von Werten, die als Parameter für den Funktionsaufruf genommen werden sollen. Die Länge dieses Vektors kann dabei maximal 16 sein. Abhängig vom jeweiligen Datentyp des Parameters muss dann das entsprechende Strukturelement gefüllt werden. Sind bei der aufgerufenen Funktion Parameter als input/output deklariert, so kann auf diese Werte nach dem Return von DMcob_CallFunction zugegriffen werden.

-> DM-value-count of DM-ValueArray

In diesem Parameter muss die aktuelle Anzahl gesetzter Werte in dem Array angegeben werden. Diese Anzahl muss dabei identisch mit der bei der Funktionsdeklaration angegebenen Parameter im Dialogskript sein. Ist dies nicht der Fall, wird die Funktion vom Dialog Manager nicht aufgerufen.

<-> DM-va-datatype (index) of DM-ValueArray

In diesem Element wird der Datentyp des Parameters mit der Nummer "index" an den Dialog Manager übergeben. Abhängig von diesem Wert muss dann das entsprechende Strukturelement gesetzt sein.

Achtung

Der Datentyp des Parameters kann bei input/output Parameter vom Dialog Manager geändert werden! Um daher auf die Rückgabewerte zugreifen zu können, muss zuerst der Datentyp abgefragt werden.

-> DM-Options of DM-StdArgs

Wenn Strings als Parameter dienen, sollte hier DMF-GetMasterString gesetzt sein, damit Strings und keine TextIDs als Rückgabewerte kommen. Zu diesem Wert kann dann aber jede andere Option addiert werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* bzw. *DM-va-value-string* kann auch *DM-value-string-u* bzw. *DM-va-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* und *DM-va-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Funktion konnte nicht aufgerufen werden.
DM-success	Funktion wurde erfolgreich aufgerufen.

Beispiel

Um die Funktion

```
function cobol void ExampleFunction (string[80] input output, integer input output)
```

aufzurufen, muss das COBOL-Programm etwa wie folgt aussehen:

```
move DT-string to DM-va-datatype(1).
move "only for testing@" to DM-va-value-string(1).
move DT-integer to DM-va-datatype(2).
move 15 to DM-va-value-integer(2).
move 2 to DM-value-count.
move DMF-GetMasterString to DM-Options.
call "DMcob_CallFunction" using DM-StdArgs DM-Value
                        DM-Object
                        DM-ValueArray.
display DM-va-value-string(1).
display DM-va-value-integer(2).
```

Siehe auch

Objekt Application

Handbuch „Distributed Dialog Manager (DDM)“

6.8.6 DMcob_CallMethod

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

Die Belegung der Parameter ist dabei von dem Objekt und der aufgerufenen Methode abhängig.

```
77  DM-method      pic 9(9) binary.
77  DM-objectID    pic 9(9) binary.

call "DMcob_CallMethod" using
             DM-StdArgs
             DM-Value
             DM-objectID
             DM-method
             DM-ValueArray.
```

Parameter

<- DM-Value

In diesem Parameter wird der Rückgabewert der Methode zurückgegeben, falls der Methodenaufruf durchgeführt werden konnte. Um auf diesen Wert zugreifen zu können, muss zuerst der Datentyp überprüft und entsprechend diesem Wert auf die Struktur zugegriffen werden.

-> DM-objectID

Dies ist der Identifikator des Objekts, bei dem die Methode aufgerufen werden soll.

-> DM-method

Dies ist die Methode, die aufgerufen werden soll. Diese Methoden sind in der Copy-Datei **IDMcobls.cob** bzw. **IDMcobws.cob** definiert.

Dabei bedeuten die definierten Konstanten folgendes:

- *MT-insert*
  Einfügen von Zeilen/Spalten beim Tablefield.
- *MT-delete*
  Löschen von Zeilen/Spalten beim Tablefield.
- *MT-clear*
  Löschen des Inhalts in Zeilen/Spalten eines Tablefields.

<-> DM-ValueArray

Dieser Parameter ist ein Array von Werten, die als Parameter für den Methodenaufruf genommen werden sollen. Die Länge dieses Vektors kann dabei maximal 16 sein. Abhängig vom jeweiligen Datentyp des Parameters muss dann das entsprechende Strukturelement gefüllt werden.

-> DM-count of DM-ValueArray

In diesem Parameter muss die aktuelle Anzahl gesetzter Werte in dem Array angegeben werden. Diese Anzahl muss dabei identisch mit der bei der Methode erlaubten Parameter sein. Ist dies nicht der Fall, wird die Methode vom Dialog Manager nicht aufgerufen.

<-> DM-va-datatype (index) of DM-ValueArray

In diesem Element wird der Datentyp des Parameters mit der Nummer "index" an den Dialog Manager übergeben. Abhängig von diesem Wert muss dann das entsprechende Strukturelement gesetzt sein.

-> DM-Options of DM-StdArgs

Unbenutzt. Muss 0 sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* bzw. *DM-va-value-string* kann auch *DM-value-string-u* bzw. *DM-va-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* und *DM-va-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error Methode konnte nicht aufgerufen werden.

DM-success Methode konnte ausgeführt werden.

Die nachfolgende Tabelle zeigt die Belegung der Parameter bei den Methoden des Tablefields.

	MT-insert	MT-delete	MT-clear
DM-va-datatype(1)	DT-integer	DT-integer	DT-integer
DM-va-value-integer (1)	Position, an der Zeilen/Spalten eingefügt werden sollen	Position, an der Zeilen/Spalten gelöscht werden sollen	Position, an der in Zeilen/Spalten der Inhalt gelöscht werden soll.
DM-va-datatype(2)	DT-integer	DT-integer	DT-integer
DM-va-value-integer (2)	Anzahl der einzufügenden Zeilen / Spalten	Anzahl der zu löschenden Zeilen / Spalten	Anzahl der Zeile / Spalten, in denen der Inhalt gelöscht werden soll.
DM-va-datatype(3)	DT-boolean	DT-boolean	DT-boolean
	Beschreibt, ob gegen die Dynamikrichtung (TRUE) oder mit der Dynamikrichtung eingefügt werden soll	Beschreibt, ob gegen die Dynamikrichtung (TRUE) oder mit der Dynamikrichtung gelöscht werden soll	Beschreibt, ob die Inhalte gegen die Dynamikrichtung (TRUE) oder mit der Dynamikrichtung gelöscht werden sollen

Beispiel

Entladen eines Tablefields in einer Nachladefunktion.

```
LINKAGE SECTION.
COPY "IDMcobls.cob".
COPY "IDMcoboc.cob".

*Loeschen des Inhaltes im nicht sichtbaren Bereich.

    MOVE DT-INTEGER TO DM-VA-DATATYPE(1).
    MOVE DT-INTEGER TO DM-VA-DATATYPE(2).
    COMPUTE DM-VA-VALUE-INTEGER(1) = DM-CO-HEADER + 1.
    COMPUTE DM-VA-VALUE-INTEGER(2) = DM-CO-VISFIRST - 1.
        - DM-CO-HEADER.
    MOVE 2 TO DM-VALUE-COUNT.

    CALL "DMcob_CallMethod" USING DM-STDARGS DM-VALUE
       DM-CO-OBJECT MT-CLEAR, DM-VALUEARRAY.

    GOBACK.
```

6.8.7 DMcob_CallRule

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

Im Gegensatz zur Funktion DMcob_ExecRule, mit deren Hilfe nur Regeln ohne Parameter von der Anwendung aufgerufen werden können, sind bei der Funktion DMcob_CallRule Parameterübergabe und Rückgabewerte beim Aufruf von Dialogregeln erlaubt.

```
77  DM-ruleID     pic 9(9) binary.
77  DM-objectID    pic 9(9) binary.

call "DMcob_CallRule" using
            DM-StdArgs
            DM-Value
            DM-ruleID
            DM-objectID
            DM-ValueArray.
```

Parameter

<- DM-Value

In diesem Parameter wird der Rückgabewert der Regel zurückgegeben, falls der Regelaufruf durchgeführt werden konnte. Um auf diesen Wert zugreifen zu können, muss zuerst der Datentyp überprüft und entsprechend diesem Wert auf die Struktur zugegriffen werden.

-> DM-ruleID

Dies ist der Identifikator der Regel, die aufgerufen werden soll.

-> DM-objectID

Dies ist der Identifikator des Objektes, das als das "this"-Objekt in der Regel dienen soll.

<-> DM-ValueArray

Dieser Parameter ist ein Array von Werten, die als Parameter für den Regelaufruf genommen werden sollen. Die Länge dieses Vektors kann dabei maximal 16 sein. Abhängig vom jeweiligen Datentyp des Parameters muss dann das entsprechende Strukturelement gefüllt werden. Sind bei der aufgerufenen Regel Parameter als input/output deklariert, so kann auf diese Werte nach dem Return von DMcob_CallRule zugegriffen werden.

-> DM-count of DM-ValueArray

In diesem Parameter muss die aktuelle Anzahl gesetzter Werte in dem Array angegeben werden. Diese Anzahl muss dabei identisch mit der bei der Regeldeklaration angegebenen Parameter im Dialogskript sein. Ist dies nicht der Fall, wird die Regel vom Dialog Manager nicht aufgerufen.

<-> DM-va-datatype (index) of DM-ValueArray

In diesem Element wird der Datentyp des Parameters mit der Nummer "index" an den Dialog Manager übergeben. Abhängig von diesem Wert muss dann das entsprechende Strukturelement gesetzt sein.

Achtung

Der Datentyp des Parameters kann bei input/output Parameter vom Dialog Manager geändert werden! Um daher auf die Rückgabewerte zugreifen zu können, muss zuerst der Datentyp abgefragt werden.

-> DM-Options of DM-StdArgs

Wenn Strings als Parameter dienen, sollte hier DMF-GetMasterString gesetzt sein, damit Strings und keine TextIDs als Rückgabewerte kommen. Zu diesem Wert kann dann aber jede andere Option addiert werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* bzw. *DM-va-value-string* kann auch *DM-value-string-u* bzw. *DM-va-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* und *DM-va-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Regel konnte nicht aufgerufen werden.
DM-success	Regel konnte aufgerufen werden.

Beispiel

Um die Regel

```
rule boolean ExampleRule (string Arg1 input output, string Arg2 input output,
integer Arg3 input output)
{
    print this;
    print Arg1;
    print Arg2;
    print Arg3;
    Arg1 := "New valid string";
    Arg2 := "also new";
    Arg3 := Arg3 + 1;
    return (true);
}
```

aus COBOL heraus aufzurufen, muss das COBOL-Programm etwa wie folgt aussehen:

```
move DT-string to DM-va-datatype(1).
move "only for testing@" to DM-va-value-string(1).
move DT-string to DM-va-datatype(2).
move "this is the second string@" to
    DM-va-value-string(2).
move DT-integer to DM-va-datatype(3).
move 15 to DM-va-value-integer(3).
move 3 to DM-value-count.
move DMF-GetMasterString to DM-Options.
call "DMcob_CallRule" using DM-StdArgs DM-Value DM-Object
                    DialogID, DM-ValueArray.
display DM-va-value-string(1).
display DM-va-value-string(2).
display DM-va-value-integer(3).
```

6.8.8 DMcob_Control

Mit Hilfe dieser Funktion können generelle Einstellungen im ISA Dialog Manager geändert oder Aktionen ausgelöst werden.

```
77  DM-objectid pic 9(9) binary.
77  action pic 9(4) binary.

call "DMcob_Control" using
            DM-StdArgs
            DM-objectid
            action.
```

Parameter

-> DM-objectid

Objekt, zu dem die gewünschte Aktion ausgeführt werden soll.

-> action

Aktion, die der DM ausführen soll. Dazu sind verschiedene Konstanten in der Copy-Datei **IDM-cobws.cob** definiert. Diese Konstanten sind in der Tabelle unten erläutert.

-> DM-Options of DM-StdArgs

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

Rückgabewert

DM-status of DM-StdArgs

DM-error	Die Aktion konnte nicht ausgeführt werden.
DM-success	Die Aktion wurde erfolgreich durchgeführt.

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

action	DM-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	DM-options	Bedeutung
DMF-UIAutomationMode		Mit dieser Aktion kann die spezielle 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 **DMcob_Initialize** und nach dem Bootstrapping erfolgen.
	0	Deaktiviert den UI Automation Support des IDM.
	1	Aktiviert den UI Automation Support des IDM.
DMF-SignalMode	0	Die Signale werden über die Funktion signal abgefangen.
	1	Die Signale werden über die Funktion sigaction abgefangen.
DMF-BindForLoader		Schaltet die BindThruLoader-Funktionalität ein, mit der das COBOL Runtime-System Anwendungsfunktionen aufrufen kann. **Verfügbarkeit** Ab IDM-Version A.06.01.d **Siehe auch** Kapitel „BindThruLoader-Funktionalität“

action	DM-options	Bedeutung
DMF-SetCodePage		Mit Hilfe 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	DM-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. Sie ist kompatibel mit der Unicode-Kodierung von MICRO FOCUS VISUAL COBOL.
	CP-cp1252	Westeuropäische Zeichencodierung gemäß Microsoft Windows Codepage 1252.
	CP-acp	Aktuell von einer Anwendung verwendete „ANSI-Codepage“ unter Microsoft Windows. Nur unter MS 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.

Anmerkung

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 ISA Dialog Manager an die Anwendung übergibt, werden in diese Codepage übersetzt.

Beispiel

Um die Codepage auf IBM437 zu setzen, muss der Aufruf etwa wie folgt aussehen:

```
77 action        pic 9(4) binary value 0.

move CP-cp437 to DM-Options.
move DMF-SetCodePage to action.
call "DMcob_Control" using DM-StdArgs 0 action.
```

6.8.9 DMcob_CreateFromModel

Mit dieser Funktion können beliebige, auf einem Modell basierende, Objekte als Instanz oder Modell erzeugt werden.

Die Parameter definieren das Modell, von dem das neu erzeugte Objekt abgeleitet wird, sowie den Vater und den Objekttyp (Instanz oder Modell) des neu erzeugten Objekts.

```
77  DM-objectid  pic 9(9)  binary value 0.
77  DM-parentid  pic 9(9)  binary value 0.
77  DM-modelid   pic 9(9)  binary value 0.

call "DMcob_CreateFromModel" using
            DM-StdArgs
            DM-objectid
            DM-parentid
            DM-modelid.
```

Parameter

-> DM-Options of DM-StdArgs

Folgende Optionen können angegeben werden, wobei mehrere Optionen mit „oder“ verknüpft werden können:

Option	Bedeutung
0	Es wird eine Instanz erzeugt.
DMF-CreateModel	Es wird ein Modell erzeugt.
DMF-CreateInvisible	Das neu generierte Objekt wird, unabhängig von der Definition im Modell oder Default, unsichtbar erzeugt.

<- DM-objectid

In diesem Parameter wird der Identifikator des neu erzeugten Objekts zurückgegeben.

-> DM-parentid

Dieser Parameter definiert den Vater des neu erzeugten Objekts.

-> DM-modelid

Dieser Parameter definiert das Modell, von dem das neu erzeugte Objekt abgeleitet wird. Wenn es sich um ein hierarchisches Modell handelt, werden dessen Kinder auch im neu erzeugten Objekt angelegt.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Objekt konnte nicht generiert werden.
DM-success	Objekt konnte generiert werden.

Verfügbarkeit

Die Funktion **DMcob_CreateFromModel** ist ab IDM-Version A.06.01.g verfügbar.

Siehe auch

Funktion DMcob_CreateObject

Eingebaute Funktion create() im Handbuch „Regelsprache“

Methode :create()

6.8.10 DMcob_CreateObject

Mit dieser Funktion können Objekte einer beliebigen Objektklasse als Instanz oder Modell erzeugt werden.

Die Parameter definieren die Objektklasse (Pushbutton, Fenster usw.), den Vater und den Objekttyp (Instanz oder Modell) des neu erzeugten Objekts.

```
77  DM-objectid  pic 9(9)  binary value 0.
77  DM-parentid  pic 9(9)  binary value 0.
77  DM-classid   pic X(2)  value "??".

call "DMcob_CreateObject" using
            DM-StdArgs
            DM-objectid
            DM-parentid
            DM-classid.
```

Parameter

-> DM-Options of DM-StdArgs

Folgende Optionen können angegeben werden, wobei mehrere Optionen mit „oder“ verknüpft werden können:

Option	Bedeutung
0	Es wird eine Instanz erzeugt.
DMF-CreateModel	Es wird ein Modell erzeugt.
DMF-CreateInvisible	Das neu generierte Objekt wird, unabhängig von der Definition im Default, unsichtbar erzeugt.
DMF-InheritFromModel	Diese Option war vorgesehen, um ein Objekt auf Basis eines im *classid*-Parameter angegebenen Modells zu erzeugen. *DMF_InheritFromModel* sollte allerdings **nicht** verwendet werden. Stattdessen sollten Objekte, die von einem Modell abgeleitet sind, mit der Funktion DMcob_CreateFromModel erzeugt werden.

<- DM-objectid

In diesem Parameter wird der Identifikator des neu erzeugten Objekts zurückgegeben.

-> DM-parentid

Dieser Parameter definiert den Vater des neu erzeugten Objekts.

-> DM-classid

Dieser Parameter legt die Klasse des neuen Objekts fest. Alle Konstanten für die Klassendefinitionen sind in der Datei **IDMcobws.cob**, die mit dem IDM ausgeliefert wird, enthalten.

Wertebereich

- DM-Class-Application
- DM-Class-Canvas
- DM-Class-Check (Checkbox)
- DM-Class-Doccursor
- DM-Class-Document
- DM-Class-Editext (Edittext)
- DM-Class-Filereq
- DM-Class-Groupbox
- DM-Class-Image
- DM-Class-Layoutbox
- DM-Class-Listbox
- DM-Class-Mapping
- DM-Class-Menubox
- DM-Class-Menuitem
- DM-Class-Menusep
- DM-Class-Messagebox
- DM-Class-Notebook
- DM-Class-Notepage
- DM-Class-Poptext
- DM-Class-Progressbar
- DM-Class-Push (Pushbutton)
- DM-Class-Radio (Radiobutton)
- DM-Class-Record
- DM-Class-Rect (Rectangle)
- DM-Class-Scroll (Scrollbar)
- DM-Class-Spinbox
- DM-Class-Splitbox
- DM-Class-Statext (Statictext)
- DM-Class-Statusbar
- DM-Class-Tablefield

- DM-Class-Timer
- DM-Class-Toolbar
- DM-Class-Transformer
- DM-Class-Treeview
- DM-Class-Window

Rückgabewert

DM-status of DM-StdArgs

DM-error	Objekt konnte nicht generiert werden.
DM-success	Objekt konnte generiert werden.

Beispiel

Es soll ein neues Fenster generiert werden.

```
Call "DMcob_CreateObject" using DM-StdArgs
    DM-newobject DM-dialogID DM-CLass-Window.
```

Siehe auch

Funktion DMcob_CreateFromModel

Eingebaute Funktion create() im Handbuch „Regelsprache“

Methode :create()

6.8.11 DMcob_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.

```
call "DMcob_DataChanged" using
            DM-StdArgs
            DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

Option	Bedeutung
0	Es findet kein Tracing dieser Funktion statt.
DMF-Verbose	Aktiviert das Tracing dieser Funktion.

-> DM-object of DM-Value

Dieses ist die Objekt-ID des Datenmodell-Objekts an dem sich ein Datenwert geändert hat. Die Objekt-ID kann man als Rückgabewert der Funktion DMcob_PathToID erhalten.

-> DM-attribute of DM-Value

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

-> DM-index of DM-Value
-> DM-value-index of DM-Value

Dieser Parameter definiert den Index des geänderten Attributs.

Hinweis

Wenn kein Index angegeben werden soll, muss *DM-datatype* of **DM-Value** auf den Wert *DT-void* gesetzt werden.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

6.8.12 DMcob_DeleteArgString

Diese Funktion löscht ein Argument in der internen Liste. Das Argument wurde vom Benutzer an das Programm übergeben.

Alle Argumente der Kommandozeile, die von der Applikation bearbeitet werden, sollten gelöscht werden, ansonsten überprüft der DM die Argumente und bearbeitet sie eventuell.

```
77  DM-Index  pic 9(4)  binary.

call "DMcob_DeleteArgString" using
            DM-StdArgs
            DM-Index.
```

Parameter

-> DM-Index

Nummer des Argumentes, das gelöscht werden soll. Der Wert dieses Parameter muss größer gleich Null und kleiner gleich der Anzahl der Argumente sein.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Argument konnte nicht gelöscht werden.
DM-success	Argument wurde erfolgreich gelöscht.

Beispiel

Siehe Funktion **DMcob_GetArgString**.

6.8.13 DMcob_Destroy

Mit Hilfe dieser Funktion können beliebige Objekte oder Modelle des Dialogs gelöscht werden. Dabei werden alle Kinder des Objektes mit gelöscht.

```
77  DM-object  pic 9(9)  binary.

call "DMcob_Destroy" using
            DM-StdArgs
            DM-object.
```

Parameter

-> DM-Object

Identifikator des Objekts, welches Sie löschen wollen.

-> DM-Options of DM-StdArgs

Kontrolliert das Verhalten der Funktion beim Löschen.

DMF-ForceDestroy

Wenn zu dieser Option *DMF-ForceDestroy* angegeben wird, wird das Objekt gelöscht und alle Regelteile, die dieses Objekt benutzen, abgeändert, so dass 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 hierbei aber *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

DM-status of DM-StdArgs

DM-error	Objekt konnte nicht gelöscht werden.
DM-success	Objekt wurde erfolgreich gelöscht.

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

Beispiel

Es soll das Fenster „MyWindow“ gelöscht werden.

```
move DMF-ForceDestroy to DM-Options.
Call "DMcob_Destroy" using DM-StdArgs DM-object.
```

Siehe auch

Eingebaute Funktion destroy() im Handbuch „Regelsprache“

Methode :destroy()

6.8.14 DMcob_DialogPathToID

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

```
77  DM-dialogid  pic 9(9) binary value 0.
77  DM-rootid    pic 9(9) binary value 0.
77  DM-objectid  pic 9(9) binary value 0.
77  DM-path      pic X(256).

call "DMcob_DialogPathToID" using
            DM-StdArgs
            DM-objectid
            DM-dialogid
            DM-rootid
            DM-path.
```

Die Bedeutung der Parameter ist dabei identisch wie bei DMcob_PathToID.

Parameter

<- DM-objectid

Identifikator des spezifizierten Objekts, falls das Objekt in dem spezifizierten Dialog gefunden werden konnte.

-> DM-dialogid

Dies ist der Identifikator des Dialogs, in dem nach dem Objekt gesucht werden soll. Diesen Wert haben Sie von der Funktion DMcob_LoadDialog als Rückgabewert erhalten.

-> DM-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 Identifikatoren von Regeln, Funktionen, Variablen und Ressourcen erfragt werden.
- *rootid != 0*
  Der Dialog Manager soll ab dem angegebenen Objekt auf der nächsttieferen Hierarchiestufe suchen.
  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-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 Namen, 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.

Der Pfad sollte mit einem Low-Value oder dem Separator-Zeichen beendet werden, ansonsten werden maximal 256 Zeichen gelesen.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das gesuchte Objekt wurde nicht gefunden oder der Name ist nicht eindeutig.
DM-success	Identifikator des gesuchten Objektes.

Beispiel

Es soll die ID des Objektes "MyWindow" im Dialog "SecondDialog" erfragt werden.

```
Call "DMcob_DialogPathToID" using DM-StdArgs DM-object
                SecondDialog DM-null-object
                by content "MyWindow@".
```

6.8.15 DMcob_ErrMsgText

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

```
01  DM-string pic X(80).

call "DMcob_ErrMsgText" using
            DM-StdArgs
            DM-ErrorDetail ( DM-error-depth )
            DM-string.
```

Parameter

-> DM-options of DM-StdArgs

- *DMF-IncludeModule*
  Wenn diese Option gesetzt ist, wird das Modul, in dem der Fehler aufgetreten ist, zu dem String addiert.
- *DMF-IncludeSeverity*
  Wenn diese Option gesetzt ist, wird der Fehlergrad zu dem String kopiert.
- *DMF-IncludeText*
  Wenn diese Option gesetzt wird, wird die Fehlermeldung selbst zu dem String kopiert.

-> DM-ErrorDetail

Dies ist der Fehlercode, zu dem die Fehlermeldung zurückgegeben werden soll.

<-DM-string

Die Fehlermeldung wird in diesen Parameter kopiert. Der Parameter muss auf 80 Zeichen definiert werden.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Beispiel

```
Report-Errors:
  Call "DMcob_QueryError" using DM-StdArgs DM-ErrorData.
  Perform Report-Error-Detail Varying DM-Error-Depth
  from 1 by 1 until DM-Error-Depth > DM-Error-Count.
Report-Error-Detail.
    Add DMF-IncludeText to DM-Options.
    Add DMF-IncludeModule to DM-Options.
    Call "DMcob_ErrMsgText" using DM-Status
                   DM-ErrorDetail (DM-Error-Depth)
                   Buffer.
    Display Buffer.
```

6.8.16 DMcob_EventLoop

Diese Funktion startet die Verarbeitung des Dialoges. 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.

```
call "DMcob_EventLoop" using
            DM-StdArgs.
```

Parameter

-> DM-Options of DM-StdArgs

Dieser Parameter bestimmt, ob die Ereignisverarbeitung abgebrochen werden soll, wenn kein Ereignis mehr vorliegt oder ob die Bearbeitung bis zum Dialogende durchgeführt werden soll. Wenn der DM nicht warten soll, muss die Konstante

DMF-DontWait

angegeben werden; ansonsten muss *0* angegeben werden.

In dem Fall, dass hier *DMF-DontWait* angegeben wird, muss das aufrufende Programm durch eine Programmschleife sicherstellen, dass die Ereignisverarbeitung des Dialog Managers (DMcob_EventLoop) immer wieder aufgerufen wird.

Rückgabewert

DM-ResCode of DM-StdArgs

DM-error	Die Ereignisverarbeitung wurde beendet, da das Dialogende erreicht worden ist.
DM-success	Die Ereignisverarbeitung wurde beendet, da im Moment kein Ereignis mehr zur Verarbeitung anstand.

Beispiel

```
Move 0 to DM-Options.
Call "DMcob_EventLoop" using DM-StdArgs.
perform ErrorCheck.
```

6.8.17 DMcob_ExecRule

Mit Hilfe dieser Funktion können benannte Regeln vom COBOL-Interface aus zur Verarbeitung gebracht werden. Diese Regeln werden dann so ausgeführt, als ob sie in der Regelsprache aufgerufen worden wären.

```
77  DM-objectid  pic 9(9) binary value 0.
77  DM-ruleid    pic 9(9) binary value 0.

call "DMcob_ExecRule" using
            DM-StdArgs
            DM-ruleid
            DM-objectid.
```

Parameter

-> DM-ruleid

Dieser Parameter bezeichnet die Regel, die ausgeführt werden soll.

-> DM-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-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Regel konnte nicht ausgeführt werden.
DM-success	Regel konnte ausgeführt werden.

Beispiel

Um die Regel

```
rule ExampleRule ()
{
    print this;
    print "Rule called from programming interface";
}
```

aufzurufen, muss das COBOL-Programm etwa wie folgt aussehen:

```
    call "DMcob_PathToID" using DM-StdArgs DM-ruleID
                        Null-Object,
                        By content "ExampleRule@".
    call "DMcob_ExecRule" using DM-StdArgs DM-ruleID
         DialogID.
```

Siehe auch

Funktion DMcob_CallRule

6.8.18 DMcob_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 - wenn nötig - die Kontrolle an die Anwendung zurück.

```
77  DM-string    pic X(80).
77  DM-reaction  pic 9(4) binary value 0.

call "DMcob_FatalAppError" using
            DM-StdArgs
            DM-reaction
            DM-string.
```

Parameter

-> DM-reaction

In diesem Parameter wird angegeben, wie die Funktion **DMcob_FatalAppError** sich verhalten soll. Dabei gibt es folgende Möglichkeiten:

-1 Das Programm wird durch einen Core-Dump abgebrochen, falls es das zugrundeliegende Betriebssystem erlaubt.

0 Dem Benutzer wird eine Meldung ausgegeben und danach wird das Programm regulär beendet.

> 0 Dem Benutzer wird eine Meldung ausgegeben und danach das Programm mit dem angegebenen Wert als Exit-Code verlassen.

-> DM-string

In diesem Parameter wir der String übergeben, der als Fehlermeldung für den Benutzer erscheinen soll.Die Fehlermeldung muss mit einem Low-Value oder dem Separator-Zeichen beendet werden.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

-> DM-Opions of DM-StdArgs

Unbenutzt. Muss 0 sein.

Verfügbarkeit

Die Funktion ist unter UNIX und ab IDM-Version A.06.01.d auch unter MICROSOFT WINDOWS verfügbar.

6.8.19 DMcob_FreeVector

Mit Hilfe dieser Funktion wird ein temporär angelegter Speicherbereich wieder freigegeben.

```
77  DM-POINTER    pic 9(4) binary.

call "DMcob_FreeVector" using
            DM-StdArgs
            DM-POINTER.
```

Parameter

-> DM-POINTER

Identifikator des freizugebenden Speicherbereiches. Dieser Identifikator wurde entweder von DMcob_InitVector oder DMcob_GetVector erhalten.

-> DM-Options of DM-StdArgs

Unbenutzt. Muss 0 sein.

Rückgabewert

DM-Status od DM-StdArgs

DM-error	Angegebener Identifikator war kein Speicherbereich.
DM-success	Speicherbereich wurde gelöscht.

6.8.20 DMcob_GetArgCount

Mit Hilfe dieser Funktion können Sie die Anzahl der Argumente erfragen, die an das Programm übergeben worden sind.

```
77  DM-ArgCount  pic 9(4)  binary value 0.

call "DMcob_GetArgCount" using
            DM-StdArgs
            DM-ArgCount.
```

Parameter

<- DM-ArgCount

Dieser Parameter gibt die Anzahl der Argumente zurück, die an das Programm übergeben worden sind.

Bitte beachten Sie, dass das erste Argument immer der Programmname selbst ist, sodass jedes Programm mindestens ein Argument hat.

Rückgabewert

DM-status of DM-StdArgs

DM-success	Funktionsaufruf war erfolgreich.

6.8.21 DMcob_GetArgString

Mit Hilfe dieser Funktion können Sie auf ein Argument zugreifen, das an das Programm übergeben worden ist.

```
77  DM-ArgNumber    pic 9(4) binary value 0.
77  DM-Buffer    pic X(80) .
77  DM-BufferSize    pic 9(4) binary value 80.

call "DMcob_GetArgString" using
            DM-StdArgs
            DM-ArgNumber
            DM-Buffer
            DM-BufferSize.
```

Parameter

-> DM-ArgNumber

Dieser Parameter ist die Nummer des Arguments, das von dieser Funktion zurückgegeben werden soll.

Hinweis

Das Executable erhält hier die Nummer 0, das erste Argument (beispielsweise die Dialogdatei) die Nummer 1 usw.

Der Parameter *DM-ArgCount* in der Funktion DMcob_GetArgCount hingegen zählt das Executable mit (bzw. beginnt mit 1)!

Beispiel

```
MeinProgramm file.dlg
```

`MeinProgramm` -> *DM-ArgNumber* ist *0*

`file.dlg` -> *DM-ArgNumber* ist *1*

DM-ArgCount ist aber *2*

<-> DM-Buffer

In diesem Parameter sollte die Funktion den Wert des Arguments zurückgeben. Daher muss der Puffer groß genug sein, um das Argument darin speichern zu können.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-BufferSize* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

-> DM-BufferSize

Dieser Parameter ist die Größe des Puffers, den Sie an den DM übergeben haben. Die Größe muss genau die Puffergröße haben, da der DM keine Möglichkeit hat, dies zu überprüfen.

Rückgabewert

DM-status of DM-StdArgs

DM-error Das Argument konnte nicht erfolgreich erfragt werden. Mögliche Fehler: Puffer ist nicht groß genug oder das erfragte Argument existiert nicht.

DM-success Das Argument konnte erfolgreich erfragt werden.

Beispiel

Um aus der Kommandozeile den zu ladenden Dialog zu erhalten, muss das COBOL-Programm etwa wie folgt aussehen:

```
77  ArgName         pic X(80) value SPACES.
77  buffer         pic X(80) value SPACES.
77  DialogID        pic 9(9) binary value 0.
77  DM-buffersize pic 9(4) binary value 80.
77  DM-ArgCount   pic 9(4) binary value 0.
77  ArgNumber       pic 9(4) binary value 0.
77  ArgCount       pic 9(4) binary value 0.
77  MyCount         pic 9(4) binary value 0.
77  HCount         pic 9(4) binary value 0.

CheckCommand.
    MOVE ZERO TO ArgCount.
    CALL "DMcob_GetArgCount" USING DM-StdArgs DM-ArgCount.
    MOVE DM-ArgCount TO ArgCount.
    IF ArgCount >= 1
        MOVE 0 TO ArgNumber
        MOVE ZERO TO MyCount
        MOVE SPACES TO DialogName
        PERFORM GET-DIALOGNAME UNTIL ArgNumber >= ArgCount.
    END-IF.
CheckArgCount.
    EVALUATE MyCount
    WHEN 1
        PERFORM Load-Dialog
    WHEN 0
        DISPLAY "Please give a dialog script name"
        GOBACK
    WHEN OTHER
        DISPLAY "To many arguments"
```

```
    END-EVALUATE.
    GOBACK.

Get-DialogName.
    MOVE SPACES TO ArgName.
    MOVE ZERO TO buffer.
    CALL "DMcob_GetArgString" USING DM-StdArgs ArgNumber
                            buffer DM-buffersize.
    MOVE buffer to ArgName.
    SET HCount TO 0
    INSPECT ArgName TALLYING HCount FOR ALL '.dlg'.
    IF HCount = 1
        MOVE ArgName TO DialogName.
        ADD 1 TO MyCount
        PERFORM DeleteArg
    END-IF.
    ADD 1 TO ArgNumber.

DeleteArg.
    CALL "DMcob_DeleteArgString" USING DM-StdArgs ArgNummer.
```

6.8.22 DMcob_GetValue

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

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
call "DMcob_GetValue" using
            DM-StdArgs
            DM-Value.
```

Parameter

-> DM-Object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie erfragen möchten. Diesen Identifikator haben Sie als Rückgabewert der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

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

-> DM-Index of DM-Value

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

<- DM-Datatype of DM-Value

In diesem Parameter erhalten Sie den Datentyp des gesuchten Attributs. Dabei müssen Sie beachten, dass Sie das unbedingt lesen, bevor Sie auf die Daten zugreifen. Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.For the data type of each attribute please refer to the chapter „Attributes and Definitions“.

-> DM-Indexcount of DM-Value

Hier wird angegeben, wie viele Indexwerte beim Aufruf der Funktion beachtet werden sollen:

- 2-dimensionale Attribute -> Wert muss auf 2 gesetzt werden
- 1-dimensionale Attribute -> Wert muss auf 1 gesetzt werden
- nicht-indizierte Attribute -> Wert muss auf 0 gesetzt werden

-> DM-value-string-putlen of DM-Value

Hier wird angegeben, wie lang der zu erfragende String sein darf.

<- DM-value-object of DM-Value
<- DM-value-boolean of DM-Value
<- DM-value-classid of DM-Value
<- DM-value-integer of DM-Value
<- DM-value-string of DM-Value

In einem dieser Parameter erhalten Sie den Wert des gefragten Attributs. Der Wert, der gesetzt wird, ist abhängig vom Datentyp des Attributs. Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.For the data type of each attribute please refer to the chapter „Attributes and Definitions“.

-> DM-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.

- *DMF-GetMasterString*
  Wird hier *DMF-GetMasterString* angegeben, wird der Originalstring zurückgeliefert.
- *DMF-GetLocalString*
  Ist hier *DMF-GetLocalString* angegeben, wird der String in der aktuell eingestellten Sprache zurückgeliefert.
- *DMF-GetTextID*
  Wird hier *DMF-GetTextID* angegeben, liefert der Dialog Manager nur den Identifikator des Textes. Dieser kann dann direkt zum Setzen über die Funktion DMcob_SetValue verwendet werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut ist für das Objekt nicht zulässig.
DM-success	Das Attribut konnte erfolgreich erfragt werden.

Beispiel

Um auf den Titel des Fensters "TestWindow" zuzugreifen muss das COBOL Programm etwa wie folgt aussehen:

```
move 0 to DM-indexcount.
move AT-title to DM-Attribute.
move DMF-GetMasterString to DM-Options.
```

```
Call "DMcob_GetValue" using DM-StdArgs DM-Value.
if DM-Datatype is equal to DT-string
    display DM-value-string.
```

6.8.23 DMcob_GetValueBuff

Mit Hilfe dieser Funktion können Sie Attribute von DM-Objekten erfragen. Der Unterschied zu DMcob_GetValue ist, dass Sie zu dieser Funktion Ihren eigenen Puffer für String-Rückgabewerte übergeben können. Dieser Puffer kann größer sein als der Standardpuffer.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
77  DM-stringpic x(800).
77  DM-stringlen pic 9(4) binary value 800.

call "DMcob_GetValueBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-Object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribute Sie erfragen wollen. Sie haben diesen Identifikator als Rückgabewert von der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, welches Sie erfragen wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-Index of DM-Value

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

<- DM-Datatype of DM-Value

In diesem Parameter erhalten Sie den Datentyp des gesuchten Attributes. Dabei müssen Sie den vom DM erhaltenen Datentyp beachten, bevor Sie auf die Daten zugreifen.

<- DM-value-object of DM-Value
<- DM-value-boolean of DM-Value
<- DM-value-classid of DM-Value
<- DM-value-integer of DM-Value

In einem dieser Parameter erhalten Sie den Wert des gefragten Attributs. Der Wert, der gesetzt wird, ist abhängig vom Datentyp des Attributs. Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.For the data type of each attribute please refer to the chapter „Attributes and Definitions“.

<- DM-string

In diesem Parameter erhalten Sie den String des erfragten Attributs. Sie sollten dies unbedingt beachten, da es nur gültig ist, wenn der zurückgegebene Datentyp *DT-string* ist.

-> DM-stringlen

In diesem Parameter übergeben Sie die Länge des übergebenen Strings. Diese Länge darf nicht größer als die wirkliche Stringlänge des übergebenen Strings sein.

-> DM-Options of DM-StdArgs

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

- *DMF-GetMasterString*
  Wird hier *DMF-GetMasterString* angegeben, wird der Originalstring zurückgeliefert.
- *DMF-GetLocalString*
  Ist hier *DMF-GetLocalString* angegeben, wird der String in der aktuell eingestellten Sprache zurückgeliefert.
- *DMF-GetTextID*
  Wird hier *DMF-GetTextID* angegeben, liefert der Dialog Manager nur die ID des Textes. Diese kann dann direkt zum Setzen über die Funktion DMcob_SetValue verwendet werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut ist für Objekt nicht zulässig.
DM-success	Das Attribut konnte erfolgreich erfragt werden.

Beispiel

Um auf den Inhalt der Listbox "TestListbox" mit sehr langen Einträgen zugreifen zu können, muss das COBOL-Programm etwa wie folgt aussehen:

```
move 1 to DM-indexcount.
move 1 to DM-index.
```

```
move AT-content to DM-Attribute.
move DMF-GetMasterString to DM-Options.
Call "DMcob_GetValueBuff" using     DM-StdArgs DM-Value
                        DM-String Dm-Stringlen.
if DM-Datatype is equal to DT-string
    display DM-String.
```

6.8.24 DMcob_GetVector

Mit Hilfe dieser Funktion wird von einem Objekt ein vektorielles Attribut in einen temporären Speicherbereich kopiert.

```
77  DM-POINTER  pic 9(4) binary.
77  DM-ENDCOL   pic 9(4) binary.
77  DM-ENDROW   pic 9(4) binary.
77  COUNT       pic 9(4) binary.

call "DMcob_GetVector" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            COUNT
            DM-ENDCOL
            DM-ENDROW.
```

Parameter

-> DM-object of DM-Value

Identifikator des Objekts, dessen Attribute in den temporären Speicherbereich kopiert werden sollen

-> DM-attribute of DM-Value

Attribut des Objekts, das in den temporären Speicherbereich kopiert werden soll.

Dabei sind alle vektoriellen Attribute eines Objekts zugelassen.

-> DM-indexcount of DM-Value

Anzahl der Indizes, die bei dem Funktionsaufruf ausgewertet werden sollen. Handelt es sich bei der Abfrage um einen normalen Attributvektor, muss hier *1* angegeben werden. Handelt es sich jedoch um ein zweidimensionales Feld von Attributen bei einem Tablefield, muss hier *2* angegeben werden.

-> DM-index of DM-Value

Startindex, ab dem das Attribut erfragt werden soll. Sind zwei Indices zur Bezeichnung notwendig, wird hier die Zeile angegeben.

-> DM-second of DM-Value

Zweiter Startindex, ab dem das Attribut erfragt werden soll. Dieser Parameter wird nur dann ausgewertet, wenn es sich um ein Attribut mit zwei Indizes handelt und *DM-indexcount* auf *2* gesetzt wurde. Er beschreibt dann die Spalte, ab der die Werte erfragt werden sollen.

<- DM-POINTER

Identifikator des temporären Speicherbereichs, der für die Attribute des Objektes angelegt wurde.

-> DM-ENDCOL

Ende-Index, bis zu dem die Attribute dem Objekt erfragt werden sollen. Ist dieser Wert = 0, soll ab dem Startindex alles erfragt werden. Hat dieser Parameter einen Wert != 0, werden die entsprechenden Attributwerte des Objekts erfragt.

-> DM-ENDROW

Zweiter Ende-Index, falls es sich um ein zweidimensionales Attribut handelt. Ist dieser Wert = 0, wird alles ab dem Startindex erfragt. Hat dieser Parameter einen Wert != 0, werden die entsprechenden Attributwerte des Objekts erfragt.

<- COUNT

In diesem Parameter wird die Anzahl der erfragten Werte zurückgeliefert.

-> DM-options of DM-Value

Optionen, die bei dem Aufruf verwendet werden sollen.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Werte konnten nicht erfragt werden.
DM-success	Werte konnten erfolgreich erfragt werden.

Beispiel

Setzen eines Listboxinhalts mit einer unbekannten Anzahl von Strings und Abfragen des Inhalts:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FILLLISTBOX.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01    STR-TAB.
    05 STRFIELd PIC X OCCURS 80.
01    INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.
77  COUNTER     PIC 9(4) VALUE 0.
77  DLG-ID    PIC 9(9) BINARY VALUE 0.
77  ICOUNT    PIC 9(4) BINARY VALUE 0.
77  I        PIC 99 VALUE ZERO.
77  J        PIC 99 VALUE ZERO.
77  NULLVAL    PIC 9(4) BINARY VALUE 0.
LINKAGE SECTION.
COPY "IDMcobls.cob".
```

```
       77 DLG-COUNT PIC 9(9) binary.
       77 DLG-OBJECT PIC 9(9) binary.
       77 DLG-STRING PIC X(80).
       77 DLG-STR-LEN PIC 9(9) binary.

       PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT
                                       DLG-COUNT
           DLG-STRING DLG-STR-LEN.
       ORGANIZE-IN SECTION.
           MOVE DLG-COUNT TO ICOUNT.
      *Initialisierung eines Speicherplatzes im DM
           CALL "DMcob_InitVector" USING DM-StdArgs DLG-ID
                   DT-String ICOUNT.
      *Initialisierung der DM-Value Struktur
           MOVE DT-STRING TO DM DATATYPE.
           MOVE DLG-STRING TO DM-VALUE-STRING.
      *Setzen der einzelnen Inhalte
           PERFORM VARYING COUNTER FROM 1 BY 1
           UNTIL COUNTER = DLG-COUNT
               MOVE COUNTER TO DM-INDEX
               MOVE DLG-STRING TO STR-TAB
.      *Aufbereiten eines veraenderten Strings fuer die Anzeige
               MOVE COUNTER TO INT-TAB
               MOVE DLG-STR-LEN TO J
               MOVE SPACE TO STRFIELD (J)
               ADD 1 TO J
               PERFORM VARYING I FROM 1 BY 1 UNTIL 1 > 5
                   MOVE INTFIELD (I) TO STRFIELD (J)
                   ADD 1 TO J
               END-PERFORM
               MOVE STR-TAB TO DM-VALUE-STRING
               CALL "DMcob_SetVectorValue" USING DM-STDARGS
                   DM-VALUE DLG-ID
           END-PERFORM.
      *Uebergeben der gespeicherten Werte an die Anzeige
           MOVE DLG-OBJECT TO DM-OBJECT.
           MOVE AT-CONTENT TO DM-ATTRIBUTE.
           MOVE 1 TO DM-INDEXCOUNT.
           MOVE 1 TO DM-index.
           CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
               DLG-ID NULLVAL NULLVAL NULLVAL.
      *Freigeben des Speicherbereiches
           CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.

         ENTRY "GETLISTBOX" USING DM-COMMON-DATA DLG-OBJECT.
         MOVE DLG-OBJECT TO DM-OBJECT.
```

```
   MOVE AT-CONTENT TO DM-ATTRIBUTE.
   MOVE 1 TO DM-INDEXCOUNT.
   MOVE 1 TO DM-INDEX.
   CALL "DMcob_GetVector" USING DM-StdArgs DM-VALUE
     DM-POINTER ICOUNT 0 0.

*Erfragen der einzelnen Inhalte
     PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL COUNTER = ICOUNT
         MOVE COUNTER TO DM-INDEX
         CALL "DMcob_GetVectorValue" USING DM-StdArgs
             DM-VALUE DM-POINTER
         END-PERFORM.
*FREIGEBEN DES SPEICHERBEREICHES
         CALL "DMcob_FreeVector" USING DM-StdArgs
             DM-POINTER.
     GOBACK.
```

6.8.25 DMcob_GetVectorValue

Mit Hilfe dieser Funktion können Werte aus einem temporären Speicherbereich in das COBOL-Programm geholt werden.

```
77  DM-POINTER    pic 9(4) binary.

call "DMcob_GetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameter

-> DM-Index of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert aus dem temporären Speicherbereich erfragt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index 1 hat. Zusätzlich darf der Wert nicht größer als der temporäre Speicherbereich sein.

-> DM-Attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* angelegt wurde.

Es sind dann folgende Attribute zulässig: *AT-content*, *AT-userdata*, *AT-sensitive* und *AT-active*.

<- DM-Datatype of DM-Value

In diesem Element wird der Datentyp des erfragten Attributes gesetzt.

<- DM-Value… of DM-Value

In dieser Struktur wird der eigentliche Wert des Attributes gespeichert. Dabei muss beachtet werden, dass nur auf das dem Datentyp entsprechende Feld zugegriffen werden darf.

-> DM-Options of DM-StdArgs

Dieser Parameter ist zur Zeit noch unbenutzt.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Attribut konnte nicht erfragt werden.
DM-success	Attribut konnte erfolgreich erfragt werden.

Beispiel

Abfrage des 5. Strings in einem temporären Speicherbereich.

```
77  DM-POINTER        pic 9(9) binary.

MOVE 5 TO DM-INDEX.
CALL     "DMcob_GetVectorValue" USING
    DM-StdArgs DM-Value DM-Pointer.
```

6.8.26 DMcob_Initialize

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

```
call "DMcob_Initialize" using
            DM-StdArgs
            DM-Common-Data.
```

Parameter

-> DM-Common-Data

Dieser Parameter enthält die allgemeine Datenstruktur des COBOL-Programms. Dieser Wert wird an eine beliebige Callback-Funktion übergeben, um globale Daten zu behandeln. Die Datenstruktur kann vom Benutzer verändert werden, um anwendungsspezifische Daten hinzuzufügen.

-> DM-setsep of DM-StdArgs

In diesem Parameter wird dem Dialog Manager mitgeteilt, welches Zeichen das Ende der vom COBOL-Programm übergebenen Strings darstellen soll.

-> DM-getsep of DM-StdArgs

In diesem Parameter wird dem Dialog Manager mitgeteilt, mit welchem Zeichen die Strings, die an das COBOL-Programm übergeben werden sollen, aufgefüllt werden sollen.

-> DM-truncspaces of DM-StdArgs

In diesem Parameter wird dem Dialog Manager mitgeteilt, ob er generell Blanks am Ende von dem vom COBOL-Programm übergebenen Strings ignorieren soll.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-setsep* bzw. *DM-getsep* kann auch *DM-setsep-u* bzw. *DM-getsep-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Rückgabewert

DM-status of DM-StdArgs

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

Beispiel

```
MOVE "@" TO DM-SetSep.
MOVE " " TO DM-GetSep.
MOVE 1 TO DM-TRUNCSPACES.
```

```
call "DMcob_Initialize" using DM-StdArgs DM-Common-Data.
perform ErrorCheck.
```

6.8.27 DMcob_InitVector

Mit Hilfe dieser Funktion kann im Dialog Manager ein temporärer Speicherbereich generiert werden, um dort eine größere Anzahl von Attributwerten eines Objektes zu speichern. Dieser Speicherbereich sollte vor allem bei Objekten wie Tablefield und Listbox zum Setzen von Inhalten benutzt werden.

```
77  DM-POINTER    PIC 9(4) binary.
77  DM-VALUETYPE    PIC 9(4) binary.
77  COUNT    PIC 9(4) binary.

call "DMcob_InitVector"using
            DM-StdArgs
            DM-POINTER
            DM-VALUETYPE
            COUNT.
```

Parameter

<- DM-POINTER

In diesem Parameter wird der neu generierte Speicherbereich zurückgegeben. Dieser Wert muss bei allen Zugriffen auf diesen Bereich mitangegeben werden.

-> DM-VALUETYPE

Über diesen Parameter wird gesteuert, welchen Datentyp die abzuspeichernden Attributwerte haben. Die dafür notwendigen Definitionen befinden sich in den Copy-Dateien **IDMcobws.cob** und **IDMcobls.cob**. Mögliche Werte sind z.B. *DT-string*, *DT-boolean*, *DT-instance*, *DT-integer*.

Eine Ausnahme stellt *DT-void* dar. Wird dieser Wert angegeben, so wird der Platz für vier Attribute (*.content*, *.userdata*, *.sensitive* und *.active*) geschaffen.

-> DM-COUNT

In diesem Parameter wird übergeben, welche Anzahl von Attributen intern gespeichert werden soll. Diese Zahl sollte möglichst genau dem entsprechen, was wirklich benötigt wird. Wird hier *0* angegeben, so wird ein Bereich mit einer Defaultgröße angelegt.

Reicht später der generierte Bereich nicht aus, wird er automatisch vom Dialog Manager vergrößert.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Bereich konnte nicht angelegt werden.
DM-success	Bereich konnte erfolgreich angelegt werden.

Anmerkung

Speicherbereiche die mit **DMcob_InitVector** angelegt wurden, müssen immer durch DMcob_ FreeVector freigegeben werden!

Beispiel

Es soll ein Bereich für 100 Stringwerte angelegt werden.

```
77  DM-POINTER     pic 9(4) binary value 0.
77  COUNT         pic 9(4) binary value 0.

MOVE 100 COUNT.
CALL     "DMcob_InitVector"    USING
            DM-StdArgs DM-POINTER
            DT-String 100.
```

6.8.28 DMcob_LGetValueBuff

Mit Hilfe dieser Funktion können Sie Attribute von DM-Objekten erfragen. Der Unterschied zu DMcob_GetValue ist, dass Sie zu dieser Funktion Indexwerte bis zu 65535 übergeben können.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
77  DM-string    pic x(800).
77  DM-stringlen pic 9(4) binary value 800.

call "DMcob_LGetValueBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-Object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribute Sie erfragen wollen. Sie haben diesen Identifikator als Rückgabewert von der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, welches Sie erfragen wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-LONG-FIRST of DM-Value

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

-> DM-LONG-SECOND of DM-Value

Dieser Parameter wird nur bei zweidimensionalen Attributen ausgewertet.

<- DM-Datatype of DM-Value

In diesem Parameter erhalten Sie den Datentyp des gesuchten Attributes. Dabei müssen Sie den vom DM erhaltenen Datentyp beachten, bevor Sie auf die Daten zugreifen.

<- DM-value-object of DM-Value
<- DM-value-boolean of DM-Value
<- DM-value-classid of DM-Value
<- DM-value-integer of DM-Value

In einem dieser Parameter erhalten Sie den Wert des gefragten Attributs. Der Wert, der gesetzt wird, ist abhängig vom Datentyp des Attributs.

Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

<- DM-string

In diesem Parameter erhalten Sie den String des erfragten Attributs. Sie sollten dies unbedingt beachten, da es nur gültig ist, wenn der zurückgegebene Datentyp *DT-string* ist.

-> DM-stringlen

In diesem Parameter übergeben Sie die Länge des übergebenen Strings. Diese Länge darf nicht größer als die wirkliche Stringlänge des übergebenen Strings sein.

-> DM-Options of DM-StdArgs

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

- *DMF-GetMasterString*
  Wird hier *DMF-GetMasterString* angegeben, wird der Originalstring zurückgeliefert.
- *DMF-GetLocalString*
  Ist hier *DMF-GetLocalString* angegeben, wird der String in der aktuell eingestellten Sprache zurückgeliefert.
- *DMF-GetTextID*
  Wird hier *DMF-GetTextID* angegeben, liefert der Dialog Manager nur die ID des Textes. Diese kann dann direkt zum Setzen über die Funktion DMcob_SetValue verwendet werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut ist für das Objekt nicht zulässig.
DM-success	Das Attribut konnte erfolgreich erfragt werden.

Beispiel

Um auf den Inhalt der Listbox "TestListbox" mit sehr langen Einträgen zugreifen zu können, muss das COBOL-Programm etwa wie folgt aussehen:

```
move 1 to DM-indexcount.
move 15000 to DM-long-first.
```

```
move AT-content to DM-Attribute.
move DMF-GetMasterString to DM-Options.
Call "DMcob_LGetValueBuff" using     DM-StdArgs DM-Value
                        DM-String Dm-Stringlen.
if DM-Datatype is equal to DT-string
    display DM-String.
```

6.8.29 DMcob_LGetValueLBuff

Mit Hilfe dieser Funktion können Sie Attribute von DM-Objekten erfragen. Der Unterschied zu DMcob_GetValue ist, dass Sie zu dieser Funktion Indexwerte bis zu 65535 übergeben können und der Puffer eine Größe von bis zu 65535 Zeichen haben kann.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
77  DM-string    pic x(15000).
77  DM-stringlen pic 9(9) binary value 15000.

call "DMcob_LGetValueLBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-Object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribute Sie erfragen wollen. Sie haben diesen Identifikator als Rückgabewert von der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, welches Sie erfragen wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-LONG-FIRST of DM-Value

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

-> DM-LONG-SECOND of DM-Value

Dieser Parameter wird nur bei zweidimensionalen Attributen ausgewertet.

<- DM-Datatype of DM-Value

In diesem Parameter erhalten Sie den Datentyp des gesuchten Attributes. Dabei müssen Sie den vom DM erhaltenen Datentyp beachten, bevor Sie auf die Daten zugreifen.

<- DM-value-object of DM-Value
<- DM-value-boolean of DM-Value
<- DM-value-classid of DM-Value
<- DM-value-integer of DM-Value

In einem dieser Parameter erhalten Sie den Wert des gefragten Attributs. Der Wert, der gesetzt wird, ist abhängig vom Datentyp des Attributs.

Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

<- DM-string

In diesem Parameter erhalten Sie den String des erfragten Attributs. Sie sollten dies unbedingt beachten, da es nur gültig ist, wenn der zurückgegebene Datentyp *DT-string* ist.

-> DM-stringlen

In diesem Parameter übergeben Sie die Länge des übergebenen Strings. Diese Länge darf nicht größer als die wirkliche Stringlänge des übergebenen Strings sein.

-> DM-Options of DM-StdArgs

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

- *DMF-GetMasterString*
  Wird hier *DMF-GetMasterString* angegeben, wird der Originalstring zurückgeliefert.
- *DMF-GetLocalString*
  Ist hier *DMF-GetLocalString* angegeben, wird der String in der aktuell eingestellten Sprache zurückgeliefert.
- *DMF-GetTextID*
  Wird hier *DMF-GetTextID* angegeben, liefert der Dialog Manager nur die ID des Textes. Diese kann dann direkt zum Setzen über die Funktion DMcob_SetValue verwendet werden.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut ist für das Objekt nicht zulässig.
DM-success	Das Attribut konnte erfolgreich erfragt werden.

Beispiel

Um auf den Inhalt der Listbox "TestListbox" mit sehr langen Einträgen zugreifen zu können, muss das COBOL-Programm etwa wie folgt aussehen:

```
move 1 to DM-indexcount.
move 5 to DM-long-first.
```

```
move AT-content to DM-Attribute.
move DMF-GetMasterString to DM-Options.
Call "DMcob_LGetValueLBuff" using     DM-StdArgs DM-Value
                        DM-String Dm-Stringlen.
if DM-Datatype is equal to DT-string
    display DM-String.
```

6.8.30 DMcob_LGetVector

Mit Hilfe dieser Funktion wird von einem Objekt ein vektorielles Attribut in einen temporären Speicherbereich kopiert. Diese Funktion ist in der Lage, mit Indexwerten bis 65535 zu arbeiten.

```
77  DM-POINTER  pic 9(4) binary.
77  DM-ENDCOL   pic 9(9) binary.
77  DM-ENDROW   pic 9(9) binary.
77  COUNT       pic 9(9) binary.

call "DMcob_LGetVector" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            COUNT
            DM-ENDCOL
            DM-ENDROW.
```

Parameter

-> DM-object of DM-Value

Identifikator des Objekts, dessen Attribute in den temporären Speicherbereich kopiert werden sollen.

-> DM-attribute of DM-Value

Attribut des Objekts, das in den temporären Speicherbereich kopiert werden soll.

Dabei sind alle vektoriellen Attribute eines Objekts zugelassen.

-> DM-indexcount of DM-Value

Anzahl der Indizes, die bei dem Funktionsaufruf ausgewertet werden sollen. Handelt es sich bei der Abfrage um einen normalen Attributvektor, muss hier *1* angegeben werden. Handelt es sich jedoch um ein zweidimensionales Feld von Attributen bei einem Tablefield, muss hier *2* angegeben werden.

-> DM-long-first of DM-Value

Startindex, ab dem das Attribut erfragt werden soll. Sind zwei Indizes zur Bezeichnung notwendig, wird hier die Zeile angegeben.

-> DM-long-second of DM-Value

Zweiter Startindex, ab dem das Attribut erfragt werden soll. Dieser Parameter wird nur dann ausgewertet, wenn es sich um ein Attribut mit zwei Indizes handelt und *DM-indexcount* auf *2* gesetzt wurde. Er beschreibt dann die Spalte, ab der die Werte erfragt werden sollen.

<- DM-POINTER

Identifikator des temporären Speicherbereichs, der für die Attribute des Objektes angelegt wurde.

-> DM-ENDCOL

Ende-Index, bis zu dem die Attribute dem Objekt erfragt werden sollen. Ist dieser Wert = *0*, soll ab dem Startindex alles erfragt werden. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte des Objekts erfragt.

-> DM-ENDROW

Zweiter Ende-Index, falls es sich um ein zweidimensionales Attribut handelt. Ist dieser Wert = *0*, wird alles ab dem Startindex erfragt. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte des Objekts erfragt.

<- COUNT

In diesem Parameter wird die Anzahl der erfragten Werte zurückgeliefert.

-> DM-options of DM-Value

Dieser Parameter wird zur Zeit noch nicht benutzt.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Werte konnten nicht erfragt werden.
DM-success	Werte konnten erfolgreich erfragt werden.

Beispiel

Setzen eines Listboxinhalts mit einer unbekannten Anzahl von Strings und Abfragen des Inhalts:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FILLLISTBOX.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01    STR-TAB.
    05 STRFIELd PIC X OCCURS 80.
01    INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.
77  COUNTER     PIC 9(9) VALUE 0.
77  DLG-ID    PIC 9(9) BINARY VALUE 0.
77  ICOUNT    PIC 9(9) BINARY VALUE 0.
77  I        PIC 99 VALUE ZERO.
77  J        PIC 99 VALUE ZERO.
77  NULLVAL    PIC 9(4) BINARY VALUE 0.
LINKAGE SECTION.
```

```
 COPY "IDMcobls.cob".
 77 DLG-COUNT PIC 9(9) binary.
 77 DLG-OBJECT PIC 9(9) binary.
 77 DLG-STRING PIC X(80).
 77 DLG-STR-LEN PIC 9(9) binary.

 PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT
                                  DLG-COUNT
     DLG-STRING DLG-STR-LEN.
 ORGANIZE-IN SECTION.
     MOVE DLG-COUNT TO ICOUNT.
*Initialisierung eines Speicherplatzes im DM
     CALL "DMcob_LInitVector" USING DM-StdArgs DLG-ID
             DT-String ICOUNT.
*Initialisierung der DM-Value Struktur
     MOVE DT-STRING TO DM DATATYPE.
     MOVE DLG-STRING TO DM-VALUE-STRING.
*Setzen der einzelnen Inhalte
     PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL COUNTER = DLG-COUNT
         MOVE COUNTER TO DM-LONG-FIRST
         MOVE DLG-STRING TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
         MOVE COUNTER TO INT-TAB
         MOVE DLG-STR-LEN TO J
         MOVE SPACE TO STRFIELD (J)
         ADD 1 TO J
         PERFORM VARYING I FROM 1 BY 1 UNTIL 1 > 5
             MOVE INTFIELD (I) TO STRFIELD (J)
             ADD 1 TO J
         END-PERFORM
         MOVE STR-TAB TO DM-VALUE-STRING
         CALL "DMcob_LSetVectorValue" USING DM-STDARGS
             DM-VALUE DLG-ID
     END-PERFORM.
*Uebergeben der gespeicherten Werte an die Anzeige
     MOVE DLG-OBJECT TO DM-OBJECT.
     MOVE AT-CONTENT TO DM-ATTRIBUTE.
     MOVE 1 TO DM-INDEXCOUNT.
     MOVE 1 TO DM-long-first.
     CALL "DMcob_LSetVector" USING DM-StdArgs DM-Value
         DLG-ID NULLVAL NULLVAL NULLVAL.
*Freigeben des Speicherbereiches
     CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.

   ENTRY "GETLISTBOX" USING DM-COMMON-DATA DLG-OBJECT.
```

```
   MOVE DLG-OBJECT TO DM-OBJECT.
   MOVE AT-CONTENT TO DM-ATTRIBUTE.
   MOVE 1 TO DM-INDEXCOUNT.
   MOVE 1 TO DM-LONG-FIRST.
   CALL "DMcob_LGetVector" USING DM-StdArgs DM-VALUE
     DM-POINTER ICOUNT 0 0.

*Erfragen der einzelnen Inhalte
     PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL COUNTER = ICOUNT
         MOVE COUNTER TO DM-INDEX
         CALL "DMcob_LGetVectorValue" USING DM-StdArgs
             DM-VALUE DM-POINTER
         END-PERFORM.
*FREIGEBEN DES SPEICHERBEREICHES
         CALL "DMcob_FreeVector" USING DM-StdArgs
             DM-POINTER.
     GOBACK.
```

6.8.31 DMcob_LGetVectorValue

Mit Hilfe dieser Funktion können Werte aus einem temporären Speicherbereich in das COBOL-Programm geholt werden.

```
77  DM-POINTER    pic 9(4) binary.

call "DMcob_LGetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameter

-> DM-long-first of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert aus dem temporären Speicherbereich erfragt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index *1* hat. Zusätzlich darf der Wert nicht größer als der temporäre Speicherbereich sein.

-> DM-Attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* angelegt wurde.

Es sind dann folgende Attribute zulässig: *AT-content*, *AT-userdata*, *AT-sensitive* und *AT-active*.

<- DM-Datatype of DM-Value

In diesem Element wird der Datentyp des erfragten Attributes gesetzt.

<- DM-Value… of DM-Value

In dieser Struktur wird der eigentliche Wert des Attributes gespeichert. Dabei muss beachtet werden, dass nur auf das dem Datentyp entsprechende Feld zugegriffen werden darf.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Attribut konnte nicht erfragt werden.

DM-success Attribut konnte erfolgreich erfragt werden.

Beispiel

Abfrage des 5. Strings in einem temporären Speicherbereich.

```
77  DM-POINTER        pic 9(9) binary.

MOVE 5 TO DM-INDEX.
CALL     "DMcob_LGetVectorValue" USING
    DM-StdArgs DM-Value DM-Pointer.
```

6.8.32 DMcob_LGetVectorValueBuff

Mit Hilfe dieser Funktion können Werte aus einem temporären Speicherbereich in das COBOL-Programm geholt werden. Im Gegensatz zu DMcob_GetVectorValue können bei dieser Funktion Indexwerte bis 65535 und ein Datenbereich mit bis zu 65535 Zeichen angegeben werden.

```
01  DM-POINTER    pic 9(4) binary.
01  DM-string     pic X(15000).
01  DM-stringlen  pic 9(9) binary value 15000.

call "DMcob_LGetVectorValueBuff" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            DM-string
            DM-stringlen.
```

Parameter

-> DM-Long-First of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert aus dem temporären Speicherbereich erfragt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index *1* hat. Zusätzlich darf der Wert nicht größer als der temporäre Speicherbereich sein.

-> DM-Attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* angelegt wurde.

Es sind dann folgende Attribute zulässig: *AT-content*, *AT-userdata*, *AT-sensitive* und *AT-active*.

<- DM-Datatype of DM-Value

In diesem Element wird der Datentyp des erfragten Attributes gesetzt.

<- DM-Value… of DM-Value

In dieser Struktur wird der eigentliche Wert des Attributes gespeichert. Dabei muss beachtet werden, dass nur auf das dem Datentyp entsprechende Feld zugegriffen werden darf.

-> DM-string

In diesem Parameter übergeben Sie den Bereich für einen String. In diesen Bereich wird dann der aktuelle Wert kopiert. Dieser String kann eine beliebige Länge haben.

-> DM-stringlen

In diesem Parameter übergeben Sie die Stringlänge des übergebenen Strings.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Attribut konnte nicht erfragt werden.
DM-success	Attribut konnte erfolgreich erfragt werden.

Beispiele

Abfrage des 5. Strings in einem temporären Speicherbereich.

```
77  DM-POINTER        pic 9(9) binary.
77  DM-string         pic X(28000).
77  DM-stringlen      pic 9(9) binary value 28000.

MOVE 5 TO DM-Long-First.
CALL     "DMcob_LGetVectorValueBuff" USING
    DM-StdArgs DM-Value DM-Pointer
    DM-string DM-stringlen.
```

6.8.33 DMcob_LInitVector

Mit Hilfe dieser Funktion kann im Dialog Manager ein temporärer Speicherbereich generiert werden, um dort eine größere Anzahl von Attributwerten eines Objektes zu speichern. Dieser Speicherbereich sollte vor allem bei Objekten wie Tablefield und Listbox zum Setzen von Inhalten benutzt werden. Im Gegensatz zu der Funktion DMcob_InitVector können mit Hilfe dieser Funktion Bereiche bis 65535 angegeben werden.

```
77  DM-POINTER    PIC 9(4) binary.
77  DM-VALUETYPE    PIC 9(4) binary.
77  COUNT    PIC 9(9) binary.

call "DMcob_LInitVector"using
            DM-StdArgs
            DM-POINTER
            DM-VALUETYPE
            COUNT.
```

Parameter

<- DM-POINTER

In diesem Parameter wird der neu generierte Speicherbereich zurückgegeben. Dieser Wert muss bei allen Zugriffen auf diesen Bereich mitangegeben werden.

-> DM-VALUETYPE

Über diesen Parameter wird gesteuert, welchen Datentyp die abzuspeichernden Attributwerte haben. Die dafür notwendigen Definitionen befinden sich in den Copy-Dateien **IDMcobws.cob** und **IDMcobls.cob**. Mögliche Werte sind z.B. *DT-string*, *DT-boolean*, *DT-instance*, *DT-integer*.

Eine Ausnahme stellt *DT-void* dar. Wird dieser Wert angegeben, so wird der Platz für vier Attribute (*.content*, *.userdata*, *.sensitive* und *.active*) geschaffen.

-> COUNT

In diesem Parameter wird übergeben, welche Anzahl von Attributen intern gespeichert werden soll. Diese Zahl sollte möglichst genau dem entsprechen, was wirklich benötigt wird. Wird hier *0* angegeben, so wird ein Bereich mit einer Defaultgröße angelegt.

Reicht später der generierte Bereich nicht aus, wird er automatisch vom Dialog Manager vergrößert.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Bereich konnte nicht angelegt werden.

DM-success Bereich konnte erfolgreich angelegt werden.

Anmerkung

Speicherbereiche die mit **DMcob_LInitVector** angelegt wurden, müssen immer durch DMcob_FreeVector freigegeben werden!

Beispiel

Es soll ein Bereich für 15000 Stringwerte angelegt werden.

```
77  DM-POINTER    pic 9(4) binary value 0.
77  COUNT         pic 9(9) binary value 15000.

MOVE 100 COUNT.
CALL     "DMcob_LInitVector"    USING
            DM-StdArgs DM-POINTER
            DT-String count.
```

6.8.34 DMcob_LoadDialog

Mit Hilfe dieser Funktion können die Dialoge in den DM geladen werden.

```
77  DM-dialogid  pic 9(9) binary value 0.
01  DM-path      pic X(256).

call "DMcob_LoadDialog" using
            DM-StdArgs
            DM-dialogid
            DM-path.
```

Parameter

<- DM-dialogid

Dieser Parameter enthält den Identifikator des geladenen Dialogs. Die folgenden Werte sind zulässig:

0	Die angezeigte Datei ist keine fehlerfreie DM-Datei oder die Datei wurde vom DM nicht gefunden.
!= 0	Identifikator des Dialogs, der vom DM geladen wurde.

<- DM-path

Die zu ladende Datei wird in diesem Parameter spezifiziert. Es kann ein Pfad zu der zu ladenden Datei sein.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

-> DM-Options of DM-StdArgs

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

Rückgabewert

DM-status of DM-StdArgs

DM-error	Die angegebene Datei ist keine fehlerfreie DM-Datei oder die Datei wurde vom DM nicht gefunden.
DM-success	Identifikator des vom DM geladenen Dialogs.

Der Dialog Manager ist in der Lage, die Dialogdaten sowohl aus einer ASCII-Datei als auch einer Binärdatei zu laden. Der Vorteil der Binärdatei gegenüber der ASCII-Datei 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 <binary file name> <ASCII file name>
```

Beispiel

```
77  buffer     pic x(256) value spaces.
77  DialogID     pic 9(9) binary value 0.

move "./cobol.dlg" to buffer.
call "DMcob_LoadDialog" using DM-StdArgs DialogID buffer.
perform ErrorCheck.
```

6.8.35 DMcob_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.

```
77  DM-dialogid  pic 9(9) binary value 0.
01  DM-path      pic X(256).

call "DMcob_LoadProfile" using
            DM-StdArgs
            DM-dialogid
            DM-path.
```

Parameter

<- DM-dialogid

Dies ist der Identifikator des zu konfigurierenden Dialoges.

-> DM-path

Pfad der Datei, die eingelesen werden soll.Der Pfad sollte mit einem Low-Value oder dem Separator-Zeichen beendet werden, ansonsten werden maximal 256 Zeichen gelesen.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

-> DM-Options of DM-StdArgs

Unbenutzt. Muss *0* sein.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Datei konnte nicht eingelesen werden.
DM-success	Datei konnte eingelesen werden.

Beispiel

```
77  DM-Profile     pic x(256) value spaces.

move "./myModifications" to DM-profile.
call "DMcob_LoadProfile" using DM-StdArgs DialogID
    DM-profile.
perform ErrorCheck.
```

6.8.36 DMcob_LSetValueBuff

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten zu verändern. Der Unterschied zu DMcob_SetValue besteht darin, dass Indexwerte bis 65535 angegeben werden können.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
01  DM-string     pic X(800)
01  DM-stringlen  pic 9(4) binary value 800.

call "DMcob_LSetValueBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-object of DM-Value

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

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, das Sie ändern wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-LONG-FIRST of DM-Value

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

-> DM-LONG-SECOND of DM-Value

Dieser Parameter wird nur bei zweidimensionalen Vektorattributen von Objekten ausgewertet. Er beschreibt den zweiten Index des gesuchten Unterobjekts (z.B. Text in Tablefield).

-> DM-value-object of DM-Value
-> DM-value-boolean of DM-Value
-> DM-value-classid of DM-Value
-> DM-value-integer of DM-Value

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 dem Kapitel „Attribute und Definitionen“.

-> DM-string

In diesem Parameter übergeben Sie den String, den Sie setzen wollen. Dieser String kann eine beliebige Länge haben.

-> DM-stringlen

In diesem Parameter übergeben Sie die Stringlänge des übergebenen Strings.

-> DM-Options of DM-StdArgs

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

- *DMF-Inhibit*
  Der Parameter muss mit *DMF-Inhibit* belegt sein, wenn keine Ereignisse verschickt werden sollen.
- *DMF-ShipEvent*
  Wenn Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-ShipEvent* belegt sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut konnte nicht gesetzt werden. The attribute could not be set.
DM-success	Das Attribut konnte erfolgreich gesetzt werden.

Beispiel

Um das erste Feld im Tablefield "MyTable" zu ändern, muss das COBOL-Programm etwa wie folgt aussehen:

```
01 DM-string     PIC X(300) value Spaces.
01 DM-stringlen    PIC 9(4) binary value 0.

move 2 to DM-indexcount.
move 15000 to DM-long-first.
move 1 to DM-long-second.
move AT-content to DM-Attribute.
move DT-string to DM-Datatype.
```

```
move DMF-Inhibit to DM-Options.
move "This is a long new content for the tablefield"
    to DM-string.
Call "DMcob_LSetValueBuff" using     DM-StdArgs DM-Value
                         DM-String Dm-Stringlen.
```

6.8.37 DMcob_LSetValueLBuff

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten zu verändern. Der Unterschied zu DMcob_SetValue besteht darin, dass Indexwerte bis 65535 angegeben werden können und dass die Datengröße bis zu 65535 Zeichen betragen kann.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
01  DM-string     pic X(15000)
01  DM-stringlen    pic 9(9) binary value 15000.

call "DMcob_LSetValueLBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-object of DM-Value

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

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, das Sie ändern wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-LONG-FIRST of DM-Value

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

-> DM-LONG-SECOND of DM-Value

Dieser Parameter wird nur bei zweidimensionalen Vektorattributen von Objekten ausgewertet. Er beschreibt den zweiten Index des gesuchten Unterobjekts (z.B. Text in Tablefield).

-> DM-value-object of DM-VALUE
-> DM-value-boolean of DM-VALUE
-> DM-value-classid of DM-VALUE
-> DM-value-integer of DM-VALUE

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 dem Kapitel „Attribute und Definitionen“.

-> DM-string

In diesem Parameter übergeben Sie den String, den Sie setzen wollen. Dieser String kann eine beliebige Länge haben.

-> DM-stringlen

In diesem Parameter übergeben Sie die Stringlänge des übergebenen Strings.

-> DM-Options of DM-StdArgs

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

- *DMF-Inhibit*
  Der Parameter muss mit *DMF-Inhibit* belegt sein, wenn keine Ereignisse verschickt werden sollen.
- *DMF-ShipEvent*
  Wenn Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-ShipEvent* belegt sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut konnte nicht gesetzt werden. The attribute could not be set.
DM-success	Das Attribut konnte erfolgreich gesetzt werden.

Beispiel

Um das erste Feld im Tablefield "MyTable" zu ändern, muss das COBOL-Programm etwa wie folgt aussehen:

```
01 DM-string     PIC X(30000) value Spaces.
01 DM-stringlen    PIC 9(9) binary value 30000.

move 2 to DM-indexcount.
move 5 to DM-long-first.
move 1 to DM-long-second.
move AT-content to DM-Attribute.
move DT-string to DM-Datatype.
```

```
move DMF-Inhibit to DM-Options.
move "This is a long new content for the tablefield"
    to DM-string.
Call "DMcob_LSetValueLBuff" using
           DM-StdArgs DM-Value
           DM-String Dm-Stringlen.
```

6.8.38 DMcob_LSetVector

Mit Hilfe dieser Funktion wird einem Objekt und einem Attribut ein temporärer Speicherbereich zugeordnet. Dabei können Bereiche bis zu einem Indexwert von 65535 angegeben werden.

```
77  DM-POINTER  pic 9(4) binary.
77  DM-ENDCOL   pic 9(9) binary.
77  DM-ENDROW   pic 9(9) binary.
77  COUNT       pic 9(9) binary.

call "DMcob_LSetVector" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            COUNT
            DM-ENDCOL
            DM-ENDROW.
```

Parameter

-> DM-object of DM-Value

Identifikator des Objekts, dem der temporäre Speicherbereich zugeordnet werden soll.

-> DM-attribute of DM-Value

Attribut des Objekts, auf den der temporäre Speicherbereich zugewiesen werden soll. Dabei sind alle vektoriellen Attribute eines Objekts zugelassen.

-> DM-indexcount of DM-Value

Anzahl der Indizes, die bei dem Funktionsaufruf ausgewertet werden sollen. Handelt es sich bei der Zuweisung um einen normalen Attributvektor, muss hier *1* angegeben werden. Handelt es sich jedoch um ein zweidimensionales Feld von Attributen bei einem Tablefield, muss hier *2* angegeben werden.

-> DM-long-first of DM-Value

Startindex, ab dem das Attribut zugewiesen werden soll. Sind zwei Indizes zur Bezeichnung notwendig, wird hier die Zeile angegeben.

-> DM-long-second of DM-Value

Zweiter Startindex, ab dem das Attribut zugewiesen werden soll. Dieser Parameter wird nur dann ausgewertet, wenn es sich um ein Attribut mit zwei Indizes handelt und *DM-indexcount* auf *2* gesetzt wurde. Er beschreibt dann die Spalte, ab der die Werte gesetzt werden sollen.

-> DM-POINTER

Identifikator des temporären Speicherbereichs, der dem Objekt zugewiesen werden soll.

-> DM-ENDCOL

Ende-Index, bis zu dem die Attribute dem Objekt zugewiesen werden soll. Ist dieser Wert = *0*, soll ab dem Startindex alles ersetzt werden. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte bei dem Objekt ersetzt. Die Größe des angelegten Speicherbereichs muss dann allerdings mit der Anzahl der zu setzenden Attribute übereinstimmen.

-> DM-ENDROW

Zweiter Ende-Index, falls es sich um ein zweidimensionales Attribut handelt. Ist dieser Wert = *0*, wird alles ab dem Startindex ersetzt. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte bei dem Objekt gesetzt.

-> COUNT

Über diesen Parameter kann die Anzahl der zu setzenden Werte gesteuert werden. Ist der Wert = *0*, wird der Speicherbereich, wie er angelegt wurde, an den Dialog Manager übergeben. Ist der Wert *!= 0* und kleiner als der Speicherbereich, wird die angegebene Anzahl von Werten an den Dialog Manager übergeben.

-> DM-options of DM-Value

Optionen, die bei dem Aufruf verwendet werden sollen. Um die Userdata in einer Listbox oder einem Tablefield zu setzen, muss die Option *DMF-UseUserdata* angegeben werden.

- *DMF-OmitActive*
  Mit dieser Option wird kein "ActiveVector" zugewiesen, d.h., das Attribut *.active* wird in diesem Vektor ignoriert.
- *DMF-OmitSensitive*
  Mit dieser Option wird kein "SensitiveVector" zugewiesen, d.h., das Attribut *.sensitive* wird in diesem Vektor ignoriert.
- *DMF-OmitStrings*
  Mit dieser Option werden keine Strings zugewiesen, d.h., das Attribut *.content* wird in diesem Vektor ignoriert.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Werte konnten nicht gesetzt werden.
DM-success	Werte konnten erfolgreich gesetzt werden.

Beispiel

Setzen eines Listboxinhalts mit einer unbekannten Anzahl von Strings:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FILLLISTBOX.

ENVIRONMENT DIVISION.
```

```
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01    STR-TAB.
    05 STRFIELD PIC X OCCURS 80.
01    INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.
77  COUNTER     PIC 9(9) VALUE 0.
77  DLG-ID    PIC 9(9) BINARY VALUE 0.
77  ICOUNT    PIC 9(9) BINARY VALUE 0.
77  I        PIC 99 VALUE ZERO.
77  J        PIC 99 VALUE ZERO.
LINKAGE SECTION.
COPY "IDMcobls.cob".
77 DLG-COUNT PIC 9(9) binary.
77 DLG-OBJECT PIC 9(9) binary.
77 DLG-STRING PIC X(80).
77 DLG-STR-LEN PIC 9(9) binary.

PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT
                         DLG-COUNT
    DLG-STRING DLG-STR-LEN.
ORGANIZE-IN SECTION.
    MOVE DLG-COUNT TO ICOUNT.
*Initialisierung eines Speicherplatzes im DM
    CALL "DMcob_LInitVector" USNG DM-StdArgs DLG-ID
        DT-String ICOUNT.
*Initialisierung der DM-Value Struktur
    MOVE DT-STRING TO DM DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.

*Setzen der einzelnen Inhalte
    PERFORM VARYING COUNTER FROM 1 BY 1
            UNTIL COUNTER = DLG-COUNT
        MOVE COUNTER TO DM-LONG-FIRST
        MOVE DLG-STRING TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
        MOVE COUNTER TO INT-TAB
        MOVE DLG-STR-LEN TO J
        MOVE SPACE TO STRFIELD (J)
        ADD 1 TO J
        PERFORM VARYING I FROM 1 BY 1 UNTIL 1 > 5
            MOVE INTFIELD (I) TO STRFIELD (J)
            ADD 1 TO J
```

```
        END-PERFORM

        MOVE STR-TAB TO DM-VALUE-STRING
        CALL "DMcob_LSetVectorValue" USING DM-STDARGS
            DM-VALUE DLG-ID
    END-PERFORM.
*Uebergeben der gespeicherten Werte an die Anzeige
    MOVE DLG-OBJECT TO DM-OBJECT.
    MOVE AT-CONTENT TO DM-ATTRIBUTE.
    MOVE 1 TO DM-INDEXCOUNT.
    MOVE 1 TO DM-LONG-FIRST.
    CALL "DMcob_LSetVector" USING DM-StdArgs DM-Value
        DLG-ID NULLVAL NULLVAL NULLVAL.
*Freigeben des Speicherbereiches
    CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.
```

6.8.39 DMcob_LSetVectorValue

Mit Hilfe dieser Funktion können in einem temporären Speicherbereich einzelne Attributwerte gesetzt werden. Dafür muss der Datentyp der zu setzenden Werte immer mit dem Datentyp übereinstimmen, mit dem der temporäre Speicherbereich generiert wurde.

```
77  DM-POINTER     pic 9(4) binary.

call "DMcob_LSetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameter

-> DM-Datatype of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, welcher Wert in der Value- Struktur (DM-Value-…) zum Setzen des Attributes ausgewertet werden soll. Dieser Datentyp muss unbedingt mit dem des temporären Speicherbereiches übereinstimmen. Falls der temporäre Speicherbereich mit dem Datentyp *DT-void* angelegt wurde, darf der *DM-Datatype* jeden beliebigen Datentyp annehmen.

-> DM-long-first of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert im Speicherbereich gesetzt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index *1* hat. Wird hier ein Wert angegeben, der größer als der Initialwert ist, wird der Speicherbereich automatisch vergrößert.

-> DM-attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* generiert wurde. Er kann dann folgende vier Werte annehmen:

- *AT-active*
  Datentyp muss dann *DT-boolean* sein.
- *AT-sensitive*
  Datentyp muss dann *DT-boolean* sein.
- *AT-content*
  Datentyp muss dann *DT-string* sein.
- *AT-userdata*
  Datentyp beliebig, entsprechend dem Wert, der in der Userdata hinterlegt werden soll.

-> DM-Value-… of DM-Value

In diesem Parameter wird der Wert übergeben, der gespeichert werden soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Struktur belegen.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Attributwert konnte nicht gespeichert werden.
DM-success	Attributwert wurde erfolgreich gespeichert.

Beispiel

Um einen Speicherbereich mit 100 Strings zu füllen, muss das COBOL-Programm etwa wie folgt aussehen:

```
77  DM-POINTER        pic 9(4) binary.
77  COUNTER        pic 9(4) binary.

CALL "DMcob_LInitVector" using     DM-StdArgs DM-POINTER
                        DM-String
                        by reference 15000.
MOVE DT-String TO DM-Datatype.
PERFORM VARYING COUNTER FROM
    1 BY 1 UNTIL COUNTER=100
    MOVE COUNTER TO DM-LONG-FIRST
    CALL     "DMcob_LSetVectorValue" USING
        DM-StdArgs DM-Value DM-Pointer
END-PERFORM.
```

6.8.40 DMcob_LSetVectorValueBuff

Mit Hilfe dieser Funktion können in einem temporären Speicherbereich einzelne Attributwerte gesetzt werden. Dafür muss der Datentyp der zu setzenden Werte immer mit dem Datentyp übereinstimmen, mit dem der temporäre Speicherbereich generiert wurde. Der Unterschied zu der Funktion DMcob_SetVectorValue besteht darin, dass hier Indexwerte bis 65535 angesprochen und Datenbereiche mit bis zu 65530 Zeichen übergeben werden können.

```
77  DM-POINTER     pic 9(4) binary.
77  DM-string      pic X(30000).
77  DM-stringlen   pic 9(9) binary value 30000.

call "DMcob_LSetVectorValueBuff" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            DM-string
            DM-stringlen.
```

Parameter

-> DM-Datatype of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, welcher Wert in der Value- Struktur (DM-Value-…) zum Setzen des Attributes ausgewertet werden soll. Dieser Datentyp muss unbedingt mit dem des temporären Speicherbereiches übereinstimmen. Falls der temporäre Speicherbereich mit dem Datentyp *DT-void* angelegt wurde, darf der *DM-Datatype* jeden beliebigen Datentyp annehmen.

-> DM-long-first of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert im Speicherbereich gesetzt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index *1* hat. Wird hier ein Wert angegeben, der größer als der Initialwert ist, wird der Speicherbereich automatisch vergrößert.

-> DM-attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* generiert wurde. Er kann dann folgende vier Werte annehmen:

- *AT-active*
  Datentyp muss dann *DT-boolean* sein.
- *AT-sensitive*
  Datentyp muss dann *DT-boolean* sein.
- *AT-content*
  Datentyp muss dann *DT-string* sein.
- *AT-userdata*
  Datentyp beliebig, entsprechend dem Wert, der in der Userdata hinterlegt werden soll.

-> DM-Value-... of DM-Value

In diesem Parameter wird der Wert übergeben, der gespeichert werden soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Struktur belegen.

-> DM-Options of DM-StdArgs

Dieser Parameter wird zur Zeit noch nicht benutzt.

COBOL-Schnittstelle für Micro Focus Visual COBOL

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Attributwert konnte nicht gespeichert werden.
DM-success	Attributwert wurde erfolgreich gespeichert.

Beispiel

Um einen Speicherbereich mit 100 Strings zu füllen, muss das COBOL-Programm etwa wie folgt aussehen:

```
77  DM-POINTER        pic 9(4) binary.
77  COUNTER        pic 9(4) binary.
77  DM-string      pic X(30000)
77  DM-stringlen      pic 9(9) binary value 30000.

CALL "DMcob_LInitVector" using     DM-StdArgs DM-POINTER
                        DM-String
                        by reference 15000.
MOVE DT-String TO DM-Datatype.
MOVE "Hallo" to DM-string.
PERFORM VARYING COUNTER FROM
    1 BY 1 UNTIL COUNTER=100
    MOVE COUNTER TO DM-LONG-FIRST
    CALL     "DMcob_LSetVectorValueBuff" USING
        DM-StdArgs DM-Value DM-Pointer DM-string
        DM-stringlen
END-PERFORM.
```

6.8.41 DMcob_OpenBox

Mit Hilfe dieser Funktion kann eine entsprechend angegebene Messagebox oder eine 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“.

```
77 DM-box     pic 9(9)  binary.
77 DM-parent  pic 9(9)  binary.

call "DMcob_OpenBox" using
             DM-StdArgs
             DM-box
             DM-parent
             DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

Dieser Parameter ist unbenutzt und daher mit *0* zu belegen.

-> DM-box

Dieser Parameter spezifiziert die Messagebox bzw. Dialogbox, die geöffnet werden soll. Den Identifikator erhält man als Rückgabewert der Funktion DMcob_PathToID.

-> DM-parent

Dieser Parameter spezifiziert das Fenster, in dem die Messagebox erscheinen soll. Falls dieser Parameter angegeben ist, muss ein Fenster oder NULL angegeben werden. Den Identifikator des Fensters erhält man als Rückgabewert der Funktion DMcob_PathToID.

Der Parameter kann ignoriert 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-Value

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.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Die Messagebox bzw. Dialogbox konnte nicht geöffnet werden.
DM-success	Die Messagebox bzw. Dialogbox wurde geöffnet.

Siehe auch

Funktionen DMcob_OpenBoxBuff, DMcob_QueryBox

Objekt Messagebox

Eingebaute Funktion querybox im Handbuch „Regelsprache“

6.8.42 DMcob_OpenBoxBuff

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

Der Unterschied zu DMcob_OpenBox ist, dass Sie zu dieser Funktion Ihren eigenen Puffer für String-werte übergeben können. Dieser Puffer kann größer sein als der Standardpuffer.

Hinweis

Bei Verwendung in Funktionen, welche Records als Parameter enthalten, beachten Sie bitte die Hinweise im Kapitel „Behandlung von String-Parametern“.

```
77  DM-box        pic 9(9) binary.
77  DM-parent     pic 9(9) binary.
77  DM-string     pic x(800).
77  DM-stringlen  pic 9(4) binary value 800.

call "DMcob_OpenBoxBuff" using
```

```
            DM-StdArgs
            DM-box
            DM-parent
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

<-> DM-options of DM-StdArgs

Dieser Parameter ist unbenutzt und daher mit *0* zu belegen.

-> DM-box

Dieser Parameter spezifiziert die Messagebox bzw. Dialogbox, die geöffnet werden soll. Den Identifikator erhält man als Rückgabewert der Funktion DMcob_PathToID.

-> DM-parent

Dieser Parameter spezifiziert das Fenster, in dem die Messagebox erscheinen soll. Falls dieser Parameter angegeben ist, muss ein Fenster oder NULL angegeben werden. Den Identifikator des Fensters erhält man als Rückgabewert der Funktion DMcob_PathToID.

Der Parameter kann ignoriert 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-Value

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-string

In diesem Parameter erhalten Sie den String-Rückgabewert. Sie sollten dies unbedingt beachten, da es nur gültig ist, wenn der zurückgegebene Datentyp *DT-string* ist.

-> DM-stringlen

In diesem Parameter übergeben Sie die Länge des übergebenen Strings. Diese Länge darf nicht größer als die wirkliche Stringlänge des übergebenen Strings sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Die Messagebox bzw. Dialogbox konnte nicht geöffnet werden.
DM-success	Die Messagebox bzw. Dialogbox wurde geöffnet.

Siehe auch

Funktionen DMcob_OpenBox, DMcob_QueryBox

Objekt Messagebox

Eingebaute Funktion querybox im Handbuch „Regelsprache“

6.8.43 DMcob_PathToID

Mit Hilfe dieser Funktion wird der Ihnen bekannte externe Name eines Objektes in die interne Bezeichnung umgewandelt. Diese interne Bezeichnung eines Objektes ist über einen Programmlauf hinweg konstant, sodass Sie den Identifikator eines oft benötigten Objektes nicht bei jedem Zugriff erfragen müssen.

```
01  DM-objectid  pic 9(9) binary value 0.
01  DM-rootid    pic 9(9) binary value 0.
01  DM-path      pic X(256).

call "DMcob_PathToID" using
            DM-StdArg
            DM-objectid
            DM-rootid
            DM-path.
```

Parameter

<- DM-objectid

0 Das Objekt wurde nicht gefunden oder sein Identifikator ist nicht eindeutig.

!= 0 Identifikator des gesuchten Objekts.

Mit dem so erhaltenen Identifikator können Sie nun auf die Attribute des benannten Objekts zugreifen.

-> DM-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 Identifikatoren von Regeln, Funktionen, Variablen und Ressourcen erfragt werden.
- *rootid != 0*
  Der Dialog Manager soll ab dem angegebenen Objekt auf der nächsttieferen Hierarchiestufe suchen.
  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-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.

Der Pfad sollte mit einem Low-Value oder dem Separator-Zeichen beendet werden, ansonsten werden maximal 256 Zeichen gelesen.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das gesuchte Objekt wurde nicht gefunden oder der Name ist nicht eindeutig.
DM-success	Identifikator des gesuchten Objektes wird zurückgegeben.

Anmerkung

Mit Hilfe dieser Funktion können Sie den internen Namen **aller** Objekte, Ressourcen, Variablen, Funktionen und Regeln erfragen!

Beispiel

```
Call "DMcob_PathToID" using DM-StdArgs DM-object
        DM-rootID By content "ObjectToAccess".
```

6.8.44 DMcob_PutArgString

Mit Hilfe dieser Funktion können Sie ein Argument ersetzen, das an das Programm übergeben wurde.

```
77  DM-ArgNumber  pic 9(4) binary value 0.
77  DM-Buffer     pic X(80).

call "DMcob_PutArgString" using
            DM-StdArgs
            DM-ArgNumber
            DM-Buffer.
```

Parameter

-> DM-ArgNumber

Dieser Parameter ist die Nummer des Arguments, das durch diese Funktion ersetzt werden sollte.

-> DM-Buffer

In diesem Parameter wird der neue Wert des Arguments gespeichert.

Anmerkung

Der Buffer muss mit einem **Low-Value** oder dem **Separator-Zeichen** beendet werden!

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Rückgabewert

DM-status of DM-StdArgs

DM-success	Das Argument konnte erfolgreich ersetzt werden.
DM-error	Das Argument konnte nicht erfolgreich ersetzt werden. Mögliche Fehler: gefragtes Argument existiert nicht oder der DM erhält nicht genug Speicher, um den String abzuspeichern.

6.8.45 DMcob_QueryBox

Mit Hilfe dieser Funktion kann eine entsprechend angegebene Messagebox geöffnet werden. Das Programm wartet, bis der Benutzer diese Messagebox wieder geschlossen hat.

```
77  DM-boxid     pic 9(9) binary.
77  DM-parentID  pic 9(9) binary.

call "DMcob_QueryBox" using
            DM-StdArgs
            DM-boxID
            DM-parentID.
```

Parameter

-> DM-boxID

Dieser Parameter spezifiziert die Messagebox, die geöffnet werden soll.

-> DM-parentID

Dieser Parameter spezifiziert das Fenster, in dem die Messagebox erscheinen soll. Der Parameter kann ignoriert werden.

-> DM-Options of DM-StdArgs

Unbenutzt, muss *0* sein.

Rückgabewerte

DM-status of DM-StdArgs

DM-error	Die Messagebox konnte nicht geöffnet werden.
DM-success	Die Messagebox wurde geöffnet.

DM-rescode of DM-StdArgs

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

Um die Messagebox

```
messagebox MyMessage
{
  .button[1]  button_ok;
  .button[2]  button_cancel;
  .defbutton  1;
  .icon       icon_hand;
  .text       "Would you really like to quit?";
  .title      "Question";
}
```

von COBOL aus zu öffnen, muss das Programm etwa wie folgt aussehen:

```
    77  DM-Message-Object    pic 9(9) binary value 0.

    call "DMcob_PathToID" using DM-StdArgs DM-object
                        DM-Message-Object,
                        By content "MyMessage".
    call "DMcob_QueryBox" using DM-StdArgs DM-Object
                        DM-Message-Object.
    if DM-Rescode is equal to MB-ok then
        GoBack.
```

Siehe auch

Funktionen DMcob_OpenBox, DMcob_OpenBoxBuff

Objekt Messagebox

Eingebaute Funktion querybox im Handbuch „Regelsprache“

6.8.46 DMcob_QueryError

Mit Hilfe dieser Funktion kann die Applikation den Grund abfragen, warum der letzte DM-Aufruf fehlschlug. Der DM gibt die Anzahl der Fehler und die Fehler zurück.

```
call "DMcob_QueryError" using
            DM-StdArgs
            DM-ErrorData.
```

Parameter

<- DM-ErrorData

Dies ist ein Bereich von Fehlercodes. Er wird vom DM gefüllt. Wenn der Bereich nicht groß genug ist, werden die letzten Fehler ignoriert. Um sicherzugehen, dass der ganze Fehlercode, der im DM gespeichert ist, an die Anwendung übergeben werden kann, sollte dieser Bereich 32 Einträge groß sein.

-> DM-error-size of DM-ErrorData

Dies ist die Anzahl der Fehlercodes, die von dieser Funktion zurückgegeben werden sollten.

<-> DM-error-count of DM-ErrorData

Dies ist die tatsächliche Anzahl der aufgetretenen Fehler, die in den Fehlerbuffer gefüllt werden.

<-> DM-Error-Detail of DM-ErrorData

Dies beinhaltet den Bereich, in den die Fehlercodes, der Fehlergrad und das Modul gefüllt werden sollten.

Rückgabewert

DM-status of DM-StdArgs

DM-success Die Funktion gibt die Fehlercodes zurück.

Beispiel

```
Report-Errors:
    Call "DMcob_QueryError" using DM-StdArgs
        DM-ErrorData.
    Perform Report-Error-Detail
        Varying DM-Error-Depth from 1 by 1
        until DM-Error-Depth > DM-Error-Count.
Report-Error-Detail.
    Add DMF-IncludeText to DM-Options.
    Add DMF-IncludeModule to DM-Options.
    Call "DMcob_ErrMsgText" using DM-Status
                    DM-ErrorDetail (DM-Error-Depth)
                    Buffer.
    Display Buffer.
```

6.8.47 DMcob_QueueExtEvent

Die Ausführung einer dem externen Ereignis zugeordneten Regel wird durch die Anwendung über die Schnittstellenfunktion **DMcob_QueueExtEvent**; vorgemerkt. "Vorgemerkt" bedeutet, dass die Regel nicht sofort ausgeführt wird, sondern dass das externe Ereignis in eine Queue eingetragen wird und entsprechend den Mechanismen für Dialogereignisse weiterverarbeitet wird.

Dieses Ereignis wird dann vom Dialog Manager über die ganz normale Ereignisverarbeitungslogik abgearbeitet und führt zur Bearbeitung einer Regel im Schema "on <object> extevent <no.>".

```
77  DM-eventno     pic 9(9) binary.
77  DM-objectID    pic 9(9) binary.

call "DMcob_QueueExtEvent" using
            DM-StdArgs
            DM-objectID
            DM-eventno
            DM-ValueArray.
```

Parameter

-> DM-objectID

Identifikator des Objekts, an das dieses externe Ereignis geschickt werden soll.

-> DM-eventno

Nummer des externen Ereignisses, das ausgelöst werden soll.

<-> DM-ValueArray

Dieser Parameter ist ein Array von Werten, die als Parameter für den Regelaufruf genommen werden sollen. Die Länge dieses Vektors kann dabei maximal *16* sein. Abhängig vom jeweiligen Datentyp des Parameters muss dann das entsprechende Strukturelement gefüllt werden.

-> DM-count of DM-ValueArray

In diesem Parameter muss die aktuelle Anzahl gesetzter Werte in dem Array angegeben werden. Diese Anzahl muss dabei identisch mit der bei der Regeldeklaration angegebenen Parameter im Dialogskript sein. Ist dies nicht der Fall, wird die Regel vom Dialog Manager nicht aufgerufen.

<-> DM-va-datatype (index) of DM-ValueArray

In diesem Element wird der Datentyp des Parameters mit der Nummer "index" an den Dialog Manager übergeben. Abhängig von diesem Wert muss dann das entsprechende Strukturelement gesetzt sein.

-> DM-Options of DM-StdArgs

Unbenutzt, muss *0* sein.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

An Stelle von *DM-va-value-string* kann auch *DM-va-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-va-value-string-getlen und *DM-va-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Event konnte nicht vorgemerkt werden.
DM-success	Event konnte vorgemerkt werden.

Beispiel

Um die Regel

```
on MyObj extevent 4711 (string Arg1 input output, integer Arg2 input)
{
    print this;
    print Arg1;
    print Arg2;
    Arg1 := "New valid string";
}
```

von COBOL aus auszulösen, muss das COBOL-Programm etwa wie folgt aussehen:

```
    call "DMcob_PathToID" using DM-StdArgs DM-object
                        Null-Object,
                        By content "MyObj ".
    move DT-string to DM-va-datatype(1).
    move "only for testing " to DM-va-value-string(1).
    move DT-integer to DM-va-datatype(2).
    move 15 to DM-va-value-integer(2).
    move 2 to DM-value-count.
    move 0 to DM-Options.
    move 4711 to DM-eventno.
    call "DMcob_QueueExtEvent" using DM-StdArgs DM-Object
                        DM-eventno, DM-ValueArray.
```

6.8.48 DMcob_ResetValue

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten auf den Wert des zugehörigen Modells oder des Defaults zurückzusetzen.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
call  "DMcob_ResetValue" using
            DM-StdArgs
            DM-Value.
```

Parameter

-> DM-object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie zurücksetzen möchten. Diesen Identifikator haben Sie als Rückgabewert der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Attribut, das Sie von dem Objekt zurücksetzen möchten. Alle zugelassenen Attribute sind in der Datei **IDMcobws.cob** definiert.

->DM-Index of DM-Value

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

-> DM-Options of DM-StdArgs

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

- *DMF-Inhibit*
  Wenn keine Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-Inhibit* belegt sein.
- *DMF-ShipEvent*
  Wenn Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-ShipEvent* belegt sein.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut konnte nicht zurückgesetzt werden.
DM-success	Das Attribut konnte erfolgreich zurückgesetzt werden.

6.8.49 DMcob_SetValue

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

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
call "DMcob_SetValue" using
            DM-StdArgs
            DM-Value.
```

Parameter

-> DM-object of DM-Value

Dieser Parameter beschreibt das Objekt, dessen Attribut Sie ändern möchten. Diesen Identifikator haben Sie als Rückgabewert der Funktion DMcob_PathToID erhalten.

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Attribut, das Sie von dem Objekt ändern möchten. Alle zugelassenen Attribute sind in der Datei I**DMcobws.cob** definiert.

-> DM-Index of DM-Value

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

-> DM-Indexcount of DM-Value

Hier wird angegeben, wie viele Indexwerte beim Aufruf der Funktion beachtet werden sollen:

- 2-dimensionale Attribute -> Wert muss auf *2* gesetzt werden
- 1-dimensionale Attribute -> Wert muss auf *1* gesetzt werden
- nicht-indizierte Attribute -> Wert muss auf *0* gesetzt werden

-> DM-value-string-putlen of DM-Value

Hier wird angegeben, wie lang der zu setzende String ist (0 = Dialog Manager soll nach dem Endekennzeichen suchen).

-> DM-value-object of DM-Value
-> DM-value-boolean of DM-Value
-> DM-value-classid of DM-Value
-> DM-value-integer of DM-Value
-> DM-value-string of DM-Value

In diesem Parameter wird der Wert übergeben, den das Attribut annehmen soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Struktur belegen. Den Datentyp eines jeden Attributes entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.For the data type of each attribute please refer to the chapter „Attributes and Definitions“.

-> DM-Options of DM-StdArgs

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

» *DMF-Inhibit*
Der Parameter muss mit *DMF-Inhibit* belegt sein, wenn keine Ereignisse verschickt werden sollen.

» *DMF-ShipEvent*
Wenn Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-ShipEvent* belegt sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut konnte nicht gesetzt werden.
DM-success	Das Attribut konnte erfolgreich gesetzt werden.

Beispiel

Um den Titel des Fensters "TestWindow" zu ändern, sieht das COBOL-Programm etwa wie folgt aus:

```
move MeinObj to DM-object.
move 0 to DM-indexcount.
move AT-title to DM-Attribute.
move DT-string to DM-Datatype.
move “Neuer Titel” to DM-value-string.
move DMF-Inhibit to DM-Options.
Call "DMcob_SetValue" using DM-StdArgs DM-Value.
```

6.8.50 DMcob_SetValueBuff

Mit Hilfe dieser Funktion werden Sie in die Lage versetzt, Attribute von DM-Objekten zu verändern. Der Unterschied zu DMcob_SetValue besteht darin, dass Sie Ihren eigenen Puffer an diese Funktion übergeben können. Dies ist notwendig, wenn Sie einen String setzen wollen, der länger als 80 Zeichen ist.

Die zulässigen Attribute für den jeweiligen Objekttyp entnehmen Sie bitte dem Kapitel „Attribute und Definitionen“.

```
01  DM-string     pic X(800)
01  DM-stringlen  pic 9(4) binary value 800.

call "DMcob_SetValueBuff" using
            DM-StdArgs
            DM-Value
            DM-string
            DM-stringlen.
```

Parameter

-> DM-object of DM-Value

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

-> DM-Attribute of DM-Value

Dieser Parameter beschreibt das Objektattribut, das Sie ändern wollen. Alle zulässigen Attribute sind in der Datei **IDMcobws.cob** definiert.

-> DM-Index of DM-Value

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

-> DM-value-object of DM-Value
-> DM-value-boolean of DM-Value
-> DM-value-classid of DM-Value
-> DM-value-integer of DM-Value

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 dem Kapitel „Attribute und Definitionen“.For the data type of each attribute please refer to the chapter „Attributes and Definitions“.

-> DM-string

In diesem Parameter übergeben Sie den String, den Sie setzen wollen. Dieser String kann eine beliebige Länge haben.

-> DM-stringlen

In diesem Parameter übergeben Sie die Stringlänge des übergebenen Strings.

-> DM-Options of DM-StdArgs

Über diesen Parameter wird gesteuert, ob der Dialog Manager durch das erfolgreiche Setzen des Attributes die Regelbearbeitung auslösen soll oder nicht.

» *DMF-Inhibit*
Der Parameter muss mit *DMF-Inhibit* belegt sein, wenn keine Ereignisse verschickt werden sollen.

» *DMF-ShipEvent*
Wenn Ereignisse verschickt werden sollen, muss dieser Parameter mit *DMF-ShipEvent* belegt sein.

COBOL-Schnittstelle für Micro Focus Visual COBOL

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Der Parameter *DM-string* kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

Im Parameter *DM-stringlen* muss weiterhin die Definitionsgröße angegeben werden, zum Beispiel *33* bei *PIC N(33)*.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Das Attribut konnte nicht gesetzt werden.
DM-success	Das Attribut konnte erfolgreich gesetzt werden.

Beispiel

Um das erste Feld im Tablefield "MyTable" zu ändern, muss das COBOL-Programm etwa wie folgt aussehen:

```
01 DM-string     PIC X(300) value Spaces.
01 DM-stringlen    PIC 9(4) binary value 0.

move 2 to DM-indexcount.
move 1 to DM-index.
move 1 to DM-second.
move AT-content to DM-Attribute.
move DT-string to DM-Datatype.
move DMF-Inhibit to DM-Options.
move "This is a long new content for the tablefield"
    to DM-string.
Call "DMcob_SetValueBuff" using     DM-StdArgs DM-Value
```

```
DM-String DM-Stringlen.
```

6.8.51 DMcob_SetVector

Mit Hilfe dieser Funktion wird einem Objekt und einem Attribut ein temporärer Speicherbereich zugeordnet.

```
77  DM-POINTER  pic 9(4) binary.
77  DM-ENDCOL   pic 9(4) binary.
77  DM-ENDROW   pic 9(4) binary.
77  COUNT       pic 9(4) binary.

call "DMcob_SetVector" using
            DM-StdArgs
            DM-Value
            DM-POINTER
            COUNT
            DM-ENDCOL
            DM-ENDROW.
```

Parameter

-> DM-object of DM-Value

Identifikator des Objekts, dem der temporäre Speicherbereich zugeordnet werden soll.

-> DM-attribute of DM-Value

Attribut des Objekts, auf den der temporäre Speicherbereich zugewiesen werden soll.

Dabei sind alle vektoriellen Attribute eines Objekts zugelassen.

-> DM-indexcount of DM-Value

Anzahl der Indizes, die bei dem Funktionsaufruf ausgewertet werden sollen. Handelt es sich bei der Zuweisung um einen normalen Attributvektor, muss hier *1* angegeben werden. Handelt es sich jedoch um ein zweidimensionales Feld von Attributen bei einem Tablefield, muss hier *2* angegeben werden.

-> DM-index of DM-Value

Startindex, ab dem das Attribut zugewiesen werden soll. Sind zwei Indizes zur Bezeichnung notwendig, wird hier die Zeile angegeben.

-> DM-second of DM-Value

Zweiter Startindex, ab dem das Attribut zugewiesen werden soll. Dieser Parameter wird nur dann ausgewertet, wenn es sich um ein Attribut mit zwei Indizes handelt und *DM-indexcount* auf *2* gesetzt wurde. Er beschreibt dann die Spalte, ab der die Werte gesetzt werden sollen.

-> DM-POINTER

Identifikator des temporären Speicherbereichs, der dem Objekt zugewiesen werden soll.

-> DM-ENDCOL

Ende-Index, bis zu dem die Attribute dem Objekt zugewiesen werden soll. Ist dieser Wert = *0*, soll ab dem Startindex alles ersetzt werden. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte bei dem Objekt ersetzt. Die Größe des angelegten Speicherbereichs muss dann allerdings mit der Anzahl der zu setzenden Attribute übereinstimmen.

-> DM-ENDROW

Zweiter Ende-Index, falls es sich um ein zweidimensionales Attribut handelt. Ist dieser Wert = *0*, wird alles ab dem Startindex ersetzt. Hat dieser Parameter einen Wert *!= 0*, werden die entsprechenden Attributwerte bei dem Objekt gesetzt.

-> COUNT

Über diesen Parameter kann die Anzahl der zu setzenden Werte gesteuert werden. Ist der Wert = *0*, wird der Speicherbereich, wie er angelegt wurde, an den Dialog Manager übergeben. Ist der Wert *!= 0* und kleiner als der Speicherbereich, wird die angegebene Anzahl von Werten an den Dialog Manager übergeben.

-> DM-options of DM-Value

Optionen, die bei dem Aufruf verwendet werden sollen. Um die Userdata in einer Listbox oder einem Tablefield zu setzen, muss die Option *DMF-UseUserdata* angegeben werden.

- *DMF-OmitActive*
  Mit dieser Option wird kein "ActiveVector" zugewiesen, d.h., das Attribut *.active* wird in diesem Vektor ignoriert.
- *DMF-OmitSensitive*
  Mit dieser Option wird kein "SensitiveVector" zugewiesen, d.h., das Attribut *.sensitive* wird in diesem Vektor ignoriert.
- *DMF-OmitStrings*
  Mit dieser Option werden keine Strings zugewiesen, d.h., das Attribut *.content* wird in diesem Vektor ignoriert.

Rückgabewert

DM-Status of DM-StdArgs

DM-error	Werte konnten nicht gesetzt werden.
DM-success	Werte konnten erfolgreich gesetzt werden.

Beispiel

Setzen eines Listboxinhalts mit einer unbekannten Anzahl von Strings:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FILLLISTBOX.

ENVIRONMENT DIVISION.
```

```
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01    STR-TAB.
    05 STRFIELD PIC X OCCURS 80.
01    INT-TAB.
    05 INTFIELD PIC 9 OCCURS 5.
77  COUNTER     PIC 9(4) VALUE 0.
77  DLG-ID    PIC 9(9) BINARY VALUE 0.
77  ICOUNT    PIC 9(4) BINARY VALUE 0.
77  I        PIC 99 VALUE ZERO.
77  J        PIC 99 VALUE ZERO.
LINKAGE SECTION.
COPY "IDMcobls.cob".
77 DLG-COUNT PIC 9(9) binary.
77 DLG-OBJECT PIC 9(9) binary.
77 DLG-STRING PIC X(80).
77 DLG-STR-LEN PIC 9(9) binary.

PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT
                         DLG-COUNT
    DLG-STRING DLG-STR-LEN.
ORGANIZE-IN SECTION.
    MOVE DLG-COUNT TO ICOUNT.
*Initialisierung eines Speicherplatzes im DM
    CALL "DMcob_InitVector" USNG DM-StdArgs DLG-ID
        DT-String ICOUNT.
*Initialisierung der DM-Value Struktur
    MOVE DT-STRING TO DM DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.

*Setzen der einzelnen Inhalte
    PERFORM VARYING COUNTER FROM 1 BY 1
            UNTIL COUNTER = DLG-COUNT
        MOVE COUNTER TO DM-INDEX
        MOVE DLG-STRING TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
        MOVE COUNTER TO INT-TAB
        MOVE DLG-STR-LEN TO J
        MOVE SPACE TO STRFIELD (J)
        ADD 1 TO J
        PERFORM VARYING I FROM 1 BY 1 UNTIL 1 > 5
            MOVE INTFIELD (I) TO STRFIELD (J)
            ADD 1 TO J
```

```
        END-PERFORM

        MOVE STR-TAB TO DM-VALUE-STRING
        CALL "DMcob_SetVectorValue" USING DM-STDARGS
            DM-VALUE DLG-ID
    END-PERFORM.
*Uebergeben der gespeicherten Werte an die Anzeige
    MOVE DLG-OBJECT TO DM-OBJECT.
    MOVE AT-CONTENT TO DM-ATTRIBUTE.
    MOVE 1 TO DM-INDEXCOUNT.
    MOVE 1 TO DM-index.
    CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
        DLG-ID NULLVAL NULLVAL NULLVAL.
*Freigeben des Speicherbereiches
    CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.
```

6.8.52 DMcob_SetVectorValue

Mit Hilfe dieser Funktion können in einem temporären Speicherbereich einzelne Attributwerte gesetzt werden. Dafür muss der Datentyp der zu setzenden Werte immer mit dem Datentyp übereinstimmen, mit dem der temporäre Speicherbereich generiert wurde.

```
77  DM-POINTER  pic 9(4) binary.

call "DMcob_SetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameter

-> DM-Datatype of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, welcher Wert in der Value- Struktur (DM-Value-…) zum Setzen des Attributes ausgewertet werden soll. Dieser Datentyp muss unbedingt mit dem des temporären Speicherbereiches übereinstimmen. Falls der temporäre Speicherbereich mit dem Datentyp *DT-void* angelegt wurde, darf der *DM-Datatype* jeden beliebigen Datentyp annehmen.

-> DM-index of DM-Value

In diesem Parameter wird der Funktion mitgeteilt, der wievielte Wert im Speicherbereich gesetzt werden soll. Dabei muss beachtet werden, dass der erste gültige Wert den Index *1* hat. Wird hier ein Wert angegeben, der größer als der Initialwert ist, wird der Speicherbereich automatisch vergrößert.

-> DM-attribute of DM-Value

Dieser Parameter wird nur beachtet, wenn der temporäre Speicherbereich mit *DT-void* generiert wurde. Er kann dann folgende vier Werte annehmen:

- *AT-active*
  Datentyp muss dann *DT-boolean* sein.
- *AT-sensitive*
  Datentyp muss dann *DT-boolean* sein.
- *AT-content*
  Datentyp muss dann *DT-string* sein.
- *AT-userdata*
  Datentyp beliebig, entsprechend dem Wert, der in der Userdata hinterlegt werden soll.

-> DM-Value-… of DM-Value

In diesem Parameter wird der Wert übergeben, der gespeichert werden soll. Dabei müssen Sie unbedingt beachten, dass Sie das richtige Element in dieser Struktur belegen.

-> DM-Options of DM-StdArgs

Dieser Parameter ist zur Zeit noch unbenutzt.

An Stelle von *DM-value-string* kann auch *DM-value-string-u* verwendet werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

DM-value-string-getlen und *DM-value-string-putlen* geben weiterhin die Anzahl der Zeichen und nicht der Bytes an.

Rückgabewert

DM-status of DM-StdArgs

DM-error Attributwert konnte nicht gespeichert werden.

DM-success Attributwert wurde erfolgreich gespeichert.

Beispiel

Um einen Speicherbereich mit 100 Strings zu füllen, muss das COBOL-Programm etwa wie folgt aussehen:

```
77  DM-POINTER        pic 9(4) binary.
77  COUNTER        pic 9(4) binary.

CALL "DMcob_InitVector" using     DM-StdArgs DM-POINTER
                        DM-String
                        by reference 100.
MOVE DT-String TO DM-Datatype.
PERFORM VARYING COUNTER FROM
    1 BY 1 UNTIL COUNTER=100
    MOVE COUNTER TO DM-INDEX
    CALL     "DMcob_SetVectorValue" USING
        DM-StdArgs DM-Value DM-Pointer
END-PERFORM.
```

6.8.53 DMcob_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 COBOLMAIN in der Anwendung aufruft. Daher darf diese Funktion nur dann aufgerufen werden, wenn das Programm außerplanmäßig beendet werden soll.

```
call "DMcob_ShutDown" using DM-StdArgs.
```

Parameter

-> DM-Options of DM-StdArgs

Unbenutzt. Muss *0* sein.

Rückgabewert

Keiner.

6.8.54 DMcob_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 im Dialog als sichtbar definierten Toplevel-Objekte auf den Bildschirm und führt die Startregel aus.

```
77  DM-dialogid    pic 9(9) binary value 0.

call "DMcob_StartDialog" using
            DM-StdArgs
            DM-dialogid.
```

Parameter

-> DM-dialogid

Dies ist der Identifikator des zu startenden Dialogs. Diesen Identifikator haben Sie von der Funktion DMcob_LoadDialog als Rückgabewert erhalten.

-> DM-Options of DM-StdArgs

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

Beispiel

```
call "DMcob_StartDialog" using DM-StdArgs DialogID.
perform ErrorCheck.
```

Siehe auch

Eingebaute Funktion run im Handbuch „Regelsprache“

6.8.55 DMcob_StopDialog

Mit Hilfe der Funktion **DMcob_StopDialog** wird ein Dialog beendet. War dies der letzte laufende Dialog, wird die Ereignis-Schleife verlassen.

Hat der Parameter *DM-Options* den Wert *DMF-ForceDestroy*, wird der Dialog nach Ausführung seiner "finish"-Regeln gelöscht.

```
77  DM-dialogid    pic 9(9) binary value 0.

call "DMcob_StopDialog" using
            DM-StdArgs
            DM-dialogid.
```

Parameter

-> DM-dialogid

Dies ist der Identifikator des zu stoppenden Dialogs. Diesen Identifikator haben Sie von der Funktion DMcob_LoadDialog als Rückgabewert erhalten.

-> DM-Options

Mit Hilfe der Optionen wird gesteuert, ob der gestoppte Dialog gelöscht werden soll oder nicht. Soll er erhalten bleiben, muss hier *0* angegeben werden. Wird hier *DMF-ForceDestroy* angegeben, wird der Dialog aus dem Speicher entfernt.

Beispiel

```
move DMF-ForceDestroy to DM-Options.
call "DMcob_StopDialog" using DM-StdArgs DialogID.
perform ErrorCheck.
```

Siehe auch

Eingebaute Funktion stop im Handbuch „Regelsprache“

6.8.56 DMcob_TraceMessage

Mit Hilfe dieser Funktion können Trace-Meldungen von der Anwendung in die Trace-Datei des Dialog Managers geschrieben werden.

```
01  DM-string  pic X(99).

call "DMcob_TraceMessage" using
            DM-StdArgs
            DM-string.
```

Parameter

-> DM-string

Dieser String wird in die Trace-Datei geschrieben. Die Fehlermeldung muss mit einem Low-Value oder dem Separator-Zeichen beendet werden.

COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Der Parameter kann auch als *National Character* (*PIC N*) übergeben werden, wenn mit Unicode-Texten (UTF-16) gearbeitet wird.

-> DM-Options of DM-StdArgs

Mit diesem Parameter 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.

Anmerkung

Die Traces werden nur ausgeführt, wenn der Dialog Manager mit der Trace-Option gestartet wird.

Beispiel

```
move "Now we have reached the cobol function " to
       DM-string.
call "DMcob_TraceMessage" using DM-StdArgs DM-string.
```

6.8.57 DMcob_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.

Handelt es sich beim *AnyValue*-Parameter um eine Sammlung, z.B. vom Typ *DT-vector*, *DT-list*, *DT-hash*, *DT-matrix* oder *DT-refvec*, so kann durch Angabe des *DM-ValueIndex*-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 *DM-Value*-Parameter dem Ziel komplett (also mit *NULL* als *DM-ValueIndex*-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.

```
77 AnyValue pointer value null.

call "DMcob_ValueChange" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF-AppendValue	Bei Sammlungen wird der Datenwert aus *DM-Value* 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 eingebaute Funktion sort) erfolgt. Kombinierbar mit *DMF-SortReverse*.
DMF-SortReverse	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung in umgekehrter Reihenfolge.

-> AnyValue

Handle des verwalteten Wertes, welcher verändert werden soll.

-> DM-ValueIndex

Dieser Parameter kann für die Änderung von Elementwerten in Sammlungen verwendet werden und dient zur Angabe des Index. Ansonsten sollte *DM-idx-datatype* mit *DT-void* belegt werden. Dieser Parameter muss nicht eine gemanagte Wertereferenz sein.

-> DM-Value

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Wertesetzung konnte nicht durchgeführt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.
DM-success	Funktion erfolgreich ausgeführt, die Wertesetzung war erfolgreich oder der Wert war schon gesetzt.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChangeBuffer

Kapitel „Verwendung des Datentyps anyvalue“

6.8.58 DMcob_ValueChangeBuffer

Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung. Im Unterschied zu DMcob_ValueChange kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.

Handelt es sich beim *AnyValue*-Parameter um eine Sammlung, z.B. vom Typ *DT-vector*, *DT-list*, *DT-hash*, *DT-matrix* oder *DT-refvec*, so kann durch Angabe des *DM-ValueIndex*-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 *DM-Value*-Parameter dem Ziel komplett (also mit *NULL* als *DM-ValueIndex*-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.

```
77 AnyValue pointer value  null.
77 StringBuffer pic X(100) value  spaces.
77 StringLength pic 9(9)   binary value 100.

call "DMcob_ValueChangeBuffer" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value
            StringBuffer
            StringLength.
```

Parameter

<-> DM-options of DM-StdArgs

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF-AppendValue	Bei Sammlungen wird der Datenwert aus *DM-Value* 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*.

Option	Bedeutung
DMF-SortLinguistic	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung, welche für Strings nach linguistischen Regeln (siehe hierzu die eingebaute Funktion sort) erfolgt. Kombinierbar mit *DMF-SortReverse*.
DMF-SortReverse	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung in umgekehrter Reihenfolge.

-> AnyValue

Handle des verwalteten Wertes, welcher verändert werden soll.

-> DM-ValueIndex

Dieser Parameter kann für die Änderung von Elementwerten in Sammlungen verwendet werden und dient zur Angabe des Index. Ansonsten sollte *DM-idx-datatype* mit *DT-void* belegt werden. Dieser Parameter muss nicht eine gemanagte Wertereferenz sein.

-> DM-Value

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

<- StringBuffer

Puffer für den String, um Strings zu ändern, die länger als *80* Zeichen sind.

-> StringLength

Länge des Puffers (hier *100*), muss der Definition entsprechen.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Wertesetzung konnte nicht durchgeführt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.
DM-success	Funktion erfolgreich ausgeführt, die Wertesetzung war erfolgreich oder der Wert war schon gesetzt.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChange

Kapitel „Verwendung des Datentyps anyvalue“

6.8.59 DMcob_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 DMcob_ValueIndex lassen sich so einfach Schleifen über alle indizierten Werte bzw. Elemente realisieren.

```
77 AnyValue pointer  value  null.
77 Count    pic 9(9) binary value 0.

call "DMcob_ValueCount" using
            DM-StdArgs
            AnyValue
            DM-Value
            Count.
```

Parameter

<-> DM-options of DM-StdArgs

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.
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.

-> AnyValue

Handle des verwalteten Wertes, von welchem die Werteanzahl geholt wird. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

<- DM-Value

Hierin wird der Count-Wert zurückgeliefert. Es kann sich hierbei um eine gemanagte Wertereferenz handeln, muss aber nicht.

Folgende Rückgaben sind je nach *AnyValue*-Parameter zu erwarten:

AnyValue-Typ	Count-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

<- Count

0 … INT_MAX Anzahl von Werten (ohne die Standardwerte bei Index *[0]*, *[0,*]* oder *[*,0]*).

DM-Value Höchster Indexwert, kann entweder *void* (skalarer Wert), ein *integer*-Wert (eindimensionales Feld), ein *index*-Wert (zweidimensionales Feld) oder ein Datentyp (assoziatives Feld) sein.

Rückgabewert

DM-status of DM-StdArgs

DM-error Der Anzahl der Werte konnte nicht ermittelt werden. Das kann auf einen fehlerhaften Aufruf oder eine nicht gemanagte oder ungültige Wertereferenz hinweisen.

DM-success Das Abfragen der Werteanzahl war erfolgreich.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChange, DMcob_ValueGet, DMcob_ValueIndex

Kapitel „Verwendung des Datentyps anyvalue“

Eingebaute Funktionen countof, itemcount

6.8.60 DMcob_ValueGet

Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört.

Ist der definierte Index vom Typ *DT-void*, so wird der Gesamtwert zurückgegeben, was normalerweise einem Kopieren des Wertes entspricht.

Handelt es sich beim *DM-Value*-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.

```
77 AnyValue pointer value null.

call "DMcob_ValueGet" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

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.
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.

-> AnyValue

Handle des verwalteten Wertes. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM-ValueIndex

Dieser Parameter definiert den Index von dem der Elementwert geholt wird. Dieser Parameter muss kein gemanagter Wert sein. Falls *DM-idx-datatype* auf den Wert *DT-void* gesetzt ist, wird der *DM-Value*-Parameter zurückgegeben.

<- DM-Value

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Der Wert konnte nicht geholt werden. Das kann auf fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.
DM-success	Das Holen des Wertes war erfolgreich.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGetBuffer, DMcob_ValueIndex

Kapitel „Verwendung des Datentyps anyvalue“

6.8.61 DMcob_ValueGetBuffer

Diese Funktion dient dazu, aus Sammlungen einen einzelnen Elementwert zu holen, der zu einem definierten Index gehört. Im Unterschied zu DMcob_ValueGet kann bei dieser Funktion ein Puffer für String-Rückgabewerte angegeben werden, der größer als der Standardpuffer ist.

Ist der definierte Index vom Typ *DT-void*, so wird der Gesamtwert zurückgegeben, was normalerweise einem Kopieren des Wertes entspricht.

Handelt es sich beim *DM-Value*-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.

```
77 AnyValue pointer value  null.
77 StringBuffer pic X(100) value  spaces.
77 StringLength pic 9(9)   binary value 100.

call "DMcob_ValueGetBuffer" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value
            StringBuffer
            StringLength.
```

Parameter

<-> DM-options of DM-StdArgs

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.
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.

Option	Bedeutung
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.

-> AnyValue

Handle des verwalteten Wertes. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM-ValueIndex

Dieser Parameter definiert den Index von dem der Elementwert geholt wird. Dieser Parameter muss kein gemanagter Wert sein. Falls *DM-idx-datatype* auf den Wert *DT-void* gesetzt ist, wird der *DM-Value*-Parameter zurückgegeben.

<- DM-Value

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

<- StringBuffer

Puffer für den String, um Strings zu erhalten, die länger als *80* Zeichen sind.

-> StringLength

Länge des Puffers (hier *100*), muss der Definition entsprechen.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Der Wert konnte nicht geholt werden. Das kann auf fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.
DM-success	Das Holen des Wertes war erfolgreich.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGet, DMcob_ValueIndex

6.8.62 DMcob_ValueGetType

Diese Funktion dient zur Abfrage des Datentyps eines vom IDM verwalteten Wertes.

```
77 AnyValue pointer  value  null.
77 Type     pic 9(4) binary value 0.

call "DMcob_ValueType" using
            DM-StdArgs
            AnyValue
            Type.
```

Parameter

<-> DM-options of DM-StdArgs

Unbenutzt, muss *0* sein.

-> AnyValue

Handle des verwalteten Wertes.

<- Type

Datentyp des verwalteten Wertes.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Der Datentyp konnte nicht ermittelt werden. Das kann auf fehlerhaften Aufruf oder eine nicht gemanagte oder ungültige Wertereferenz hinweisen.
DM-success	Das Abfragen des Datentyps war erfolgreich.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Kapitel „Verwendung des Datentyps anyvalue“

6.8.63 DMcob_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 *DM-Value*-Parameterwert abgelegt werden oder in einem ungemanagten. Bei letzterem wird wenn nötig eine Allokierung des String-Wertes im temporären Puffer ausgeführt.

```
77 AnyValue pointer  value  null.
77 IndexPos pic 9(9) binary value 0.

call “DMcob_ValueIndex” using
            DM-StdArgs
            AnyValue
            IndexPos
            DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

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.
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.

-> AnyValue

Handle des verwalteten Wertes, von welcher der Index an der Position geholt wird.

-> IndexPos

Dieser Parameter definiert die Position des Index und sollte zwischen *0* < *IndexPos* <= *DMcob_ValueCount()* liegen.

<- DM-Value

Hier wird der entsprechende Index abgelegt. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Der Index konnte nicht ermittelt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Position hinweisen.
DM-success	Der Wert hat einen Index an dieser Position, der geholte Index steht danach in *DM-Value* falls *DM-Value != NULL* ist.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Funktion DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGet

Kapitel „Verwendung des Datentyps anyvalue“

Methode index

Eingebaute Funktionen countof, itemcount

6.8.64 DMcob_ValueInit

Mit dieser Funktion kann ein Wert in eine vom IDM gemanagte lokale oder globale Wertereferenz umgewandelt werden. Dadurch ist die weitere Manipulation des Wertes durch **DMcob_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.

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 DMcob_ValueChange geschehen.

```
77 AnyValue pointer  value  null.
77 Type     pic 9(4) binary value 0.

call "DMcob_ValueInit" using
            DM-StdArgs
            AnyValue
            Type
            DM-Value.
```

Parameter

<-> DM-options of DM-StdArgs

Option	Bedeutung
0	Wertereferenz wird als lokaler Wert initialisiert.
DMF-StaticValue	Wertereferenz wird als globaler, statischer Wert initialisiert.

<- AnyValue

Handle des neu angelegten verwalteten Wertes.

-> Type

Dieser Parameter bezeichnet den geforderten initialen Typ.

-> DM-indexcount of DM-Value

In diesem Parameter kann die initiale Größe von Sammlungen wie *list* oder *matrix* angegeben werden oder der zugehörige Wertetyp bei *vector*-Werten.

Rückgabewert

DM-status of DM-StdArgs

DM-error	Wertereferenz konnte nicht initialisiert werden.
DM-success	Funktion konnte erfolgreich ausgeführt werden, die Wertereferenz ist damit initialisiert.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Siehe auch

Kapitel „Verwendung des Datentyps anyvalue“

6.8.65 DMmfviscob_BindThruLoader

Mit Hilfe dieser Funktion werden Funktionen über das COBOL Runtime-System angesprochen. Sie brauchen daher nicht in der Funktionstabelle eingetragen werden, sondern müssen als separate Dateien im aktuellen Verzeichnis oder einem Verzeichis, das über die Umgebungsvariable `COBLIB` angesprochen werden kann, gehalten werden. Wenn der Name der Funktion und der Datei übereinstimmt, kann das COBOL Runtime-System diese Funktion finden und aufrufen. Um diesen gesamten Vorgang zu initialisieren, muss **DMmfviscob_BindThruLoader** aufgerufen werden.

```
call "DMmfviscob_BindThruLoader" using
        DM-StdArgs
        DM-dialogid.
```

Parameter

-> DM-Options of DM-StdArgs

- *DMF-Silent*
  Keine Fehlermeldungen sollen an den Benutzer ausgegeben werden.
  Ansonsten wird bei Fehlermeldungen zwischen „überflüssigen Funktionen“ und „fehlenden Funktionen“ unterschieden.
- *DMF-Verbose*
  Normale Fehlermeldungen an den Benutzer sollen ausgegeben werden.
  Ansonsten wird bei Fehlermeldungen zwischen „überflüssigen Funktionen“ und „fehlenden Funktionen“ unterschieden.

-> DM-dialogid

Dies ist der Identifikator des Dialogs bzw. der Applikation.

Den Dialog-Identifikator haben Sie von der Funktion DMcob_LoadDialog als Rückgabewert erhalten.

Den Identifikator der Applikation erhalten Sie von dem Parameter *DM-Appl-Id* der Funktion COBOLAPPINIT.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

Diese Funktion steht unter UNIX und MICROSOFT WINDOWS zur Verfügung.

6.8.66 DMufcob_BindThruLoader

Mit Hilfe dieser Funktion werden Funktionen über das COBOL Runtime-System angesprochen. Sie brauchen daher nicht in der Funktionstabelle eingetragen werden, sondern müssen als separate Dateien im aktuellen Verzeichnis oder einem Verzeichis, das über die Umgebungsvariable `COBLIB` angesprochen werden kann, gehalten werden. Wenn der Name der Funktion und der Datei übereinstimmt, kann das COBOL Runtime-System diese Funktion finden und aufrufen. Um diesen gesamten Vorgang zu initialisieren, muss **DMufcob_BindThruLoader** aufgerufen werden.

```
call "DMufcob_BindThruLoader" using
        DM-StdArgs
        DM-dialogid.
```

Parameter

-> DM-Options of DM-StdArgs

- *DMF-Silent*
  Keine Fehlermeldungen sollen an den Benutzer ausgegeben werden.
  Ansonsten wird bei Fehlermeldungen zwischen „überflüssigen Funktionen“ und „fehlenden Funktionen“ unterschieden.
- *DMF-Verbose*
  Normale Fehlermeldungen an den Benutzer sollen ausgegeben werden.
  Ansonsten wird bei Fehlermeldungen zwischen „überflüssigen Funktionen“ und „fehlenden Funktionen“ unterschieden.

-> DM-dialogid

Dies ist der Identifikator des Dialogs bzw. der Applikation.

Den Dialog-Identifikator haben Sie von der Funktion DMcob_LoadDialog als Rückgabewert erhalten.

Den Identifikator der Applikation erhalten Sie von dem Parameter *DM-Appl-Id* der Funktion COBOLAPPINIT.

Verfügbarkeit

Nur MICRO FOCUS COBOL auf UNIX

6.9 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	DMcob_GetValue DMcob_SetValue	Ist gültig bei Attribut *AT-options* der Canvas unter Motif. Damit wird eine Canvas bezeichnet, die Kinder haben kann.
DMF-AppendValue	DMcob_ValueChange DMcob_ValueChan-geBuffer	Anhängen eines Datenwerts an eine Sammlung, wenn *Index* = *NULL*.
DMF-BindForLoader	DMcob_Control	Schaltet die BindThruLoader-Funk-tionalität ein, mit der das COBOL Run-time-System Anwendungsfunktionen aufrufen kann.
DMF-CreateInvisible	DMcob_CreateFromModel DMcob_CreateObject	Das neu generierte Objekt soll unsicht-bar generiert werden; unabhängig davon, wie das Attribut *AT-visible* in dem Kopiermodell gesetzt ist.
DMF-CreateModel	DMcob_CreateFromModel DMcob_CreateObject	Das neu zu generierende Objekt soll als Modell generiert werden.
DMF-DontFreeLastStrings	DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	Temporärer Puffer für Strings bleibt über den nächsten Aufruf des IDM hin-aus erhalten.
DMF-DontTrace	DMcob_QueueExtEvent	Funktionsaufruf soll nicht mit-protokolliert werden.
DMF-DontWait	DMcob_EventLoop	Die Ereignisverarbeitung soll nicht „blockierend“ durchgeführt werden, d.h. ist ein Ereignis da, soll es ver-arbeitet werden; ist kein Ereignis da, soll sofort in die aufrufende Funktion zurückgesprungen werden.
DMF-ForceDestroy	DMcob_Destroy DMcob_StopDialog	Diese Option zeigt an, dass das ange-gebene Objekt gelöscht werden soll. Bei DMcob_Destroy muss diese Option gesetzt sein, wenn das Objekt wirklich gelöscht werden soll.

Option	Funktion	Bedeutung
DMF-GetLocalString	DMcob_GetValue DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	Text soll als String in der aktuell eingestellten Sprache zurückgeliefert werden.
DMF-GetMasterString	DMcob_GetValue DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	Text soll als String in der Originalsprache zurückgeliefert werden.
DMF-GetTextID	DMcob_GetValue	Text soll als Textid zurück-geliefert werden.
DMF-IncludeIdent	DMcob_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	DMcob_ErrMsgText	Der Name des Moduls, in dem der Fehler aufgetreten ist, soll in der Fehlermeldung enthalten sein.
DMF-IncludeSeverity	DMcob_ErrMsgText	Der Grad des Fehlers (Warnung, Fehler, fataler Fehler) soll in dem Fehlertext enthalten sein.
DMF-IncludeText	DMcob_ErrMsgText	Der eigentliche Fehlertext soll in der Fehlermeldung enthalten sein.
DMF-InheritFromModel	DMcob_CreateObject	Das Objekt soll einschließlich der bei dem Modell angegebenen Kinder generiert werden.
DMF-Inhibit	DMcob_SetValue DMcob_SetVector	Für das Setzen des Attributwerts soll kein Ereignis erzeugt werden und damit keine Regel: *on Objekt.Attribut changed*"ausgelöst werden.
DMF-InhibitTag	DMcob_TraceMessage	Beim Tracing soll der Zeilenanfang („header“) *„[UM]“* nicht gedruckt werden.

Option	Funktion	Bedeutung
DMF-LogFile	DMcob_TraceMessage	Ausgabe erfolgt nicht in das Tracefile, sondern in das Logfile.
DMF-OmitActive	DMcob_GetVector DMcob_SetVector	Das Attribut *AT-active* soll nicht mit übergeben werden.
DMF-OmitSensitive	DMcob_GetVector DMcob_SetVector	Das Attribut *AT-sensitive* soll nicht mit übergeben werden.
DMF-OmitStrings	DMcob_GetVector DMcob_SetVector	Die Strings sollen nicht mit übergeben werden.
DMF-OmitUserData	DMcob_GetVector	Das Attribut *AT-userdata* soll nicht zurückgegeben werden.
DMF-SetCodePage	DMcob_Control	Die angegebene Codepage soll in der Anwendung verwendet werden und alle Texte in diese Codepage umgewandelt werden.
DMF-ShipEvent	DMcob_SetValue DMcob_SetVectorValue	Für das Setzen des Attributs soll ein Ereignis er-zeugt werden und damit die möglicherweise vorhandene Regel *on .attribute changed* ausgelöst werden.
DMF-SortBinary	DMcob_ValueChange DMcob_ValueChangeBuffer	Binäres Sortieren einer Sammlung.
DMF-SortLinguistic	DMcob_ValueChange DMcob_ValueChangeBuffer	Sortieren einer Sammlung nach linguistischen Regeln.
DMF-SortReverse	DMcob_ValueChange DMcob_ValueChangeBuffer	Sortieren einer Sammlung in umgekehrter Reihenfolge.
DMF-StaticValue	DMcob_ValueInit	Anlegen einer statischen bzw. globalen, gemanagten Wertereferenz.

Option	Funktion	Bedeutung
DMF-Synchronous	DMcob_QueueExtEvent	Kann gesetzt werden, um interne Prozesse zu optimieren, wenn DMcob_QueueExtEvent nicht aus einer „Signal Handler“-Funktion gerufen wird.
DMF-UpdateScreen	DMcob_Control	Alle intern abgelaufenen Aktionen sollen auf dem Bildschirm sichtbar gemacht werden.
DMF-UseUserData	DMcob_SetVector	In dem DM-Content-Vektor soll die Userdata beachtet und dem Objekt zugewiesen werden.
DMF-Verbose	DMcob_DataChanged	Aktiviert das Tracing der Funktion.
DMF-XlateString	DMcob_SetValue	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.

7 Attribute und Definitionen

Um Zugriffe von der Anwendung auf den DM zu ermöglichen, stehen in den Include-Dateien **IDMcobws.cob** und **IDMcobls.cob** eine Reihe von symbolischen Namen zur Verfügung, die hier kurz erläutert werden sollen.

7.1 Attributdefinitionen und ihre Datentypen

Für die Zugriffsfunktionen DMcob_GetValue, DMcob_SetValue und DMcob_ResetValue stehen der Anwendung eine Reihe von Definitionen zur Verfügung. Diese Definitionen finden Sie in der "Attributreferenz".

7.2 Klassendefinition

Um die Klasse eines Objekts zu erfragen, steht im DM das Attribut *AT-class* zur Verfügung. Als Ergebnis eines Aufrufs von DMcob_GetValue gibt es folgende in **IDMcbws.cob** definierte Möglichkeiten:

Klassenidentifikator	Bedeutung
DM-Class-Application	Application
DM-Class-Canvas	Canvas
DM-Class-Check	Checkbox
DM-Class-Color	Farbe
DM-Class-Cursor	Cursor
DM-Class-Dialog	Dialog
DM-Class-Editext	editierbarer Text
DM-Class-Font	Zeichensatz
DM-Class-Func	Funktion
DM-Class-Groupbox	Groupbox
DM-Class-Image	Bild
DM-Class-Import	Importobjekt
DM-Class-Listbox	Listbox
DM-Class-Menubox	Menü

Klassenidentifikator	Bedeutung
DM-Class-Menuitem	Menüeintrag
DM-Class-Menusep	Menüseparator
DM-Class-Messagebox	Messagebox
DM-Class-Module	Modul
DM-Class-Poptext	Poptext (Combobox)
DM-Class-Push	Pushbutton
DM-Class-Radio	Radiobutton
DM-Class-Rect	Rechteck
DM-Class-Rule	Regel
DM-Class-Scroll	Scrollbar
DM-Class-Statext	statischer Text
DM-Class-Tablefield	Tablefield
DM-Class-Text	Text
DM-Class-Tile	Muster
DM-Class-Timer	Timer
DM-Class-Var	Variable
DM-Class-Window	Fenster

8 COBOL in verteilter Umgebung

Wenn COBOL zusammen mit dem verteilten Dialog Manager eingesetzt werden soll, sind hierbei folgende Randbedingungen zu beachten:

- In der Regel ist der Funktionsaufruf über das Netz teuer, nur unwesentlich entscheidend ist der dabei übermittelte Inhalt. Aus diesem Grund sollten die COBOL-Funktionen soweit wie möglich bei ihrem Aufruf über die Parameterschnittstelle mit den für sie notwendigen Informationen versorgt werden. Dieses Vorgehen ist wesentlich performanter, als die fehlenden Informationen einzeln per DMcob_GetValue anzufordern. Daher eignen sich die Records besonders gut als Parameter, da sie vom Dialog Designer selber konfiguriert werden können.
- Wenn die Inhalte von Listboxes oder Tablefields bearbeitet werden sollen, sollte dieser Inhalt zunächst in einen lokalen temporären Speicherbereich kopiert bzw. angelegt werden. Das Kopieren erfolgt mit Hilfe der Funktion DMcob_GetVector, das Anlegen über die Funktion DMcob_InitVector. Danach kann der Inhalt über die Funktion DMcob_GetVectorValue einzeln, aber lokal, abgefragt bzw. über die Funktion DMcob_SetVector Value einzeln gesetzt werden. Die Zuweisung über das Netz erfolgt dann mit einem Funktionsaufruf DMcob_SetVector. Für den Einsatz in einer Netzwerkumgebung ist diese Lösung um ein Vielfaches effizienter als jeden Listbox- oder Tablefieldinhalt einzeln über das Netzwerk zu setzen.

Beispiel

Im nachfolgenden Beispiel wird eine Listbox mit einer beliebigen, vom Benutzer definierten Anzahl von Einträgen gefüllt. Das Füllen wird zunächst in einem temporären Speicherbereich vorgenommen, der anschließend dem Objekt zugewiesen wird. In einer zweiten Funktion wird der Inhalt einzeln abgefragt, nachdem er über das Netzwerk kopiert wurde.

Dialogdatei

```
dialog DEMO
{
    .xraster   8;
    .yraster   20;
}

function void InitTestAppl(object);

application TestAppl
{
    .active false;
    .connect "lovelace:4711";

    /* F U N C T I O N   D E F I N I T I O N S */
    function cobol void FillListbox  (object, integer,
             string[80], integer);
    function cobol void GetListbox  (object);
```

```
}

on TestAppl start
{
    InitTestAppl(this);
}

/* V A R I A B L E   D E F I N I T I O N S */
variable boolean Retvalue := true;
variable boolean Changed := false;
variable integer Number := 0;
variable integer CurrentIndex := 0;

/* C O L O R   D E F I N I T I O N S */
color RED "red", grey(75), white;

/* D E F I N I T I O N S   O F   A C C E L E R A T O R S */
accelerator  AP1
{
    0: F5;
}
accelerator  AP2
{
    0: F6;
}

accelerator AP3
{
    0: cntrl + F4;
}

accelerator AP4
{
    0: cntrl + F9;
}

accelerator AP5
{
    0: F7;
}
accelerator AP6
{
    0: F8;
}

accelerator EXIT
```

```
{
    0: cntrl + 'e';
}

/* D E F A U L T    D E F I N I T I O N S */

default window
{
    .sizeraster    true;
    .posraster        true;
    .titlebar        true;
    .closeable        true;
    .sizeable        true;
    .moveable        true;
    .visible        true;
    .sensitive        true;
    .xraster        8;
    .yraster        20;
}
default pushbutton
{
    .sizeraster    true;
    .posraster     true;
    .xauto         1;
    .yauto         1;
    .visible         true;
    .sensitive     true;
    .width        8;
}

/* default value for all edittexts */
default edittext
{
    .visible         true;        /* show the edittexts    */
    .maxchars         80;        /* set the maximum chars    */
    .sensitive     true;        /* make it sensitive    */
    .sizeraster     true;        /* dimension not in pixel*/
    .posraster     true;        /* position not in pixel    */
    .xauto         1;            /* left aligned        */
    .yauto         1;            /* top aligned            */
    .borderwidth     0;
    .multiline    false;
}

default statictext
{
```

```
    .sizeraster    true;
    .posraster    true;
    .xauto        1;
    .yauto        1;
    .sensitive    false;
    .visible        true;
}

default listbox
{
    .sizeraster    true;
    .posraster    true;
    .multisel        false;
    .visible        true;
    .sensitive      true;
    .borderwidth     1;
}

default menubox
{
    .visible         true;
    .sensitive     true;
}

default menuitem
{
    .visible         true;
    .sensitive     true;
}
default menusep
{
    .visible         true;
}

/* D E F I N I T I O N S   O F   M O D E L S */

model edittext EnterNumber
{
    .width        6;
    .maxchars        10;
    .format        "%5ud";
    .sensitive     false;
}

/* this model is only used to bind rules */
model menuitem ActionMenus
```

```
{
}

/* D E F I N I T I O N S   O F   O B J E C T S */
/* Definition of the main window */
window Test
{
    .title    "DIALOG MANAGER EXAMPLE";
    .xleft    0;
    .ytop    0;
    .width    78;
    .height    21;

    menu menubox FileMenu
    {
    .title     "File";
    child menuitem Exit
    {
    .text     "&Exit";
    .accelerator EXIT;
    }
    }
    child statictext
    {
        .text    "Dummy-String :";
        .xleft    1;
        .ytop     0;
    }
    child edittext File
    {
        .xleft    14;
        .ytop    0;
        .width    20;
        .content    "test-string";
    }
    child statictext
    {
        .text    "&Lines :";
        .xleft    60;
        .ytop    0;
    }

    child edittext Lines
    {
        .model    EnterNumber;
        .xleft    69;
```

```
        .ytop    0;
        .sensitive true;
        .content     "10";
    }
    child listbox F1
    {
    .xauto        0;
    .xleft        2;
    .xright        2;
    .yauto        0;
    .ytop        4;
    .ybottom        4;
    .accelerator    AP6;
    }

    child pushbutton Pb1
    {
        .xleft         2;
        .yauto         -1;
        .ybottom         0;
        .text        "&Set";
        .accelerator    AP1;
    }
    child pushbutton Pb2
    {
        .xauto        -1;
        .xright        2;
        .yauto        -1;
        .ybottom        0;
        .text        "&Get";
        .accelerator      AP2;
    }

    child statictext ActionLine
    {
        .xleft        2;
        .ytop        3;
        .text        "";
    }

}

/* Definition of the confirm window */
window Confirm
{
    .title         "Confirm exit";
```

```
    .xleft         10;
    .ytop         8;
    .width         40;
    .height         6;
    .visible         false;
    .dialogbox     true;

    /* this object is only visible on Alpha Terminals */
    child hp_softkeys
    {
        .text[1]     "";
        .text[2]     "";
        .text[3]     "";
        .text[4]     "";
        .text[6]     "";
        .text[7]     "";
        .text[8]     "";
        .text[9]     "";
    }
    child statictext
    {
        .text     "Would you like to save your changes?";
        .ytop     1;
        .xleft     2;
    }
    child pushbutton Yes
    {
        .ytop     3;
        .xleft     20;
        .width     8;
        .height    1;
        .text     "&Yes";
        .bgc     RED;
    }

    child pushbutton No
    {
        .ytop         3;
        .xleft         30;
        .width         8;
        .height         1;
        .text         "&No";
        .bgc         RED;
    }
}
```

```
/* R U L E S   O F   T H E   D I A L O G U E */

/* dialog start rule */
on dialog start
{
    TestAppl.active := true;
    if (not TestAppl.active) then
    TestAppl.local := true;
        TestAppl.active := true;
    endif
}

/* this rule reads the rest of the file into the listbox */
on Test close
{
    exit();
}

on Pb1 select
{
    FillListbox(F1, atoi(Lines.content), File.content,     length(File.content)
+ 1);
}

on Pb2 select
{
    GetListbox(F1);
}
on Exit select
{
  exit();
}
```

Zugehöriges COBOL-Programm

```
   *SET OSVS
    IDENTIFICATION DIVISION.
    PROGRAM-ID. FILLLISTBOX.
    AUTHOR. "MD".

    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    DATA DIVISION.
    FILE SECTION.

    WORKING-STORAGE SECTION.
    01  STR-TAB.
```

```
     05 STRFIELD PIC X OCCURS 80.
 01  INT-TAB.
     05 INTFIELD        PIC 9 OCCURS 5.
 77  COUNTER         PIC 9(4) VALUE 0.
 77  DM-POINTER        PIC 9(4) BINARY VALUE 0.
 77  ICOUNT        PIC 9(4) BINARY VALUE 0.
 77  I             PIC 99 VALUE ZERO.
 77  J            PIC 99 VALUE ZERO.

 LINKAGE SECTION.
 COPY "IDMcobls.cob".

 77  DLG-COUNT PIC 9(9) binary.
 77  DLG-OBJECT PIC 9(9) binary.
 77  DLG-STRING PIC X(80).
 77  DLG-STR-LEN PIC 9(9) binary.

*Diese COBOL-Funktion legt einen temporaeren Speicher
*bereich im Dialog Manager an und weist diesen dann einer
*Listbox zu.

 PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT
         DLG-COUNT
     DLG-STRING DLG-STR-LEN.
 ORGANIZE-IN SECTION.
     MOVE DLG-COUNT TO ICOUNT.
*Initialisierung eines Speicherplatzes im DM
     CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
         DT-String ICOUNT.
*Initialisierung der DM-Value Struktur
     MOVE DT-STRING TO DM-DATATYPE.
     MOVE DLG-STRING TO DM-VALUE-STRING.

*Setzen der einzelnen Inhalte
     PERFORM VARYING COUNTER FROM 1 BY 1
         UNTIL COUNTER = DLG-COUNT
     MOVE COUNTER TO DM-INDEX
     MOVE DLG-STRING TO STR-TAB
*Aufbereiten eines veraenderten Strings fuer die Anzeige
     MOVE COUNTER TO INT-TAB
     MOVE DLG-STR-LEN TO J
     MOVE SPACE TO STRFIELD(J)
     ADD 1 TO J
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
         MOVE INTFIELD(I) TO STRFIELD(J)
         ADD 1 TO J
```

```
     END-PERFORM

     MOVE STR-TAB TO DM-VALUE-STRING
     CALL "DMcob_SetVectorValue" USING DM-STDARGS
         DM-VALUE DM-POINTER
     END-PERFORM.
*Uebergeben der gespeicherten Werte an die Anzeige
     MOVE DLG-OBJECT TO DM-OBJECT.
     MOVE AT-CONTENT TO DM-ATTRIBUTE.
     MOVE 1 TO DM-INDEXCOUNT.
     MOVE 1 TO DM-index.
     CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
     DM-POINTER  0 0 0.

*Freigeben des Speicherbereiches
     CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.

     GOBACK.

     ENTRY "GETLISTBOX" USING DM-COMMON-DATA DLG-OBJECT.

     MOVE DLG-OBJECT TO DM-OBJECT.
     MOVE AT-CONTENT TO DM-ATTRIBUTE.
     MOVE 1 TO DM-INDEXCOUNT.
     MOVE 1 TO DM-index.
     CALL "DMcob_GetVector" USING DM-StdArgs DM-Value
         DM-POINTER ICOUNT 0 0.

*Erfragen der einzelnen Inhalte
     PERFORM VARYING COUNTER FROM 1 BY 1
         UNTIL COUNTER = ICOUNT
     MOVE COUNTER TO DM-INDEX
     CALL "DMcob_GetVectorValue" USING DM-STDARGS
         DM-VALUE DM-POINTER
     END-PERFORM.
*Freigeben des Speicherbereiches
     CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
     GOBACK.
```

9 Übersetzen und Linken von DM-Programmen

In diesem Kapitel wird beschrieben, wie mit dem DM entwickelte Programme übersetzt und gelinkt werden müssen.

9.1 Copy-Dateien

Alle Source-Dateien der Anwendung, die irgendwelche Beziehungen zum Dialog Manager haben, müssen die vom DM bereitgestellten Dateien "IDMcobws.cob" und "IDMcobls.cob" beinhalten. Abhängig davon, wo diese Dateien installiert wurden, muss der Include-Pfad für den Compiler gesetzt werden.

- IDMcobws.cob
  Diese Datei beinhaltet alle Definitionen des Dialog Managers und muss einmal in die "Working storage section" der Applikation kopiert werden. COBOL allokiert in dieser Datei Speicher für die definierten Strukturen, so dass die Applikation darin auf die Daten zugreifen kann. Diese Datei muss je Anwendung genau einmal in ein Modul kopiert werden üblicherweise in das Hauptprogramm.
- IDMcobls.cob
  Diese Datei enthält alle Definitionen des Dialog Managers ohne die Werte. Diese Datei kann in die "Linkage storage section" aller anderen COBOL-Unterprogramme kopiert werden. Mit dieser Datei können Sie die Definitionen des Dialog Managers erhalten ohne Speicherplatz zu allokieren. Diese Datei kann daher beliebig oft in der Anwendung verwendet werden.
- IDMcoboc.cob
  Diese Datei muss in die COBOL-Programme kopiert werden, die die COBOL-Callback-Funktion oder Nachlade-Funktionen des Dialog Managers verwenden. Da auch in dieser Datei keine Wertdefinitionen, sondern nur eine Strukturdefinition enthalten ist, muss diese Datei in die "Linkage storage section" kopiert werden.

Falls in der Anwendung COBOL-Funktionen enthalten sind, die direkt vom Dialog Manager aufgerufen werden und einen Record als Parameter haben, muss die vom Dialog Manager über ihre Option **+writetrampolin** generierte COBOL-Copy-Datei ebenfalls in das entsprechende Modul in die "Linkage storage section" kopiert werden. Nur dann kann im COBOL-Programm auf diese Struktur zugegriffen werden.

Neben diesen COBOL-Copy-Dateien sind noch folgende C-Include-Dateien notwendig, damit ein ablauffähiges COBOL-Dialog Manager Programm erstellt werden kann.

- IDMuser.h
  Diese Datei beinhaltet alle Definitionen des Dialog Managers für die Programmiersprache C. Diese Datei wird zum Übersetzen der C-Module benötigt, die über das Programm **gencobfx** oder die Option **+writetrampolin** generiert wurden. Diese Datei wird dort included.

» IDMcobol.h
Diese Include-Datei wird nur benötigt, wenn Records an COBOL-Funktionen übergeben werden sollen. Das Modul, das über die Option "+writetrampolin" generiert wurde, included diese Datei.

9.2 Übersetzen der generierten C-Dateien

Um die für das COBOL-Programm notwendigen C-Module zu übersetzen, muss der C-Compiler mit den in der Datei **MakeDefs** enthaltenen Definitionen für das Betriebssystem und die Hardware aufgerufen werden. Diese Definitionen entnehmen Sie bitte der bei Ihnen installierten Datei.

Beispiel

PC mit Microsoft Windows

```
cl -DDOS -DMSW -DMSC -DPC
```

Neben diesen Symbolen muss bei der Übersetzung des von der Option **+writetrampolin** erzeugten Modul noch definiert werden, für welches COBOL das Modul übersetzt werden soll.

Dazu müssen folgende Flags gesetzt werden:

Compiler	Schalter beim Übersetzen der C-Sourcen
MICRO FOCUS COBOL	-DUFCOB

Beispiele

» IBM RS6000, AIX 3.2, MICRO FOCUS COBOL

```
-DAIX -DRIOS -DVERMINOR=2 -DSYSV -I$(TOPDIR)/INCLUDE -DUFCOB
```

» PC, Microsoft Windows, MBP COBOL, MSC 8.0

```
cl -DDOS -DMSW -DMSC -DVISCOB
```

9.3 Übersetzen der COBOL-Module

Damit die COBOL-Module mit dem Dialog Manager zusammenarbeiten können, müssen die Module abhängig vom eingesetzten Compiler übersetzt werden. Ebenso sind die in den COBOL-Modulen zu setzenden Optionen abhängig vom COBOL-Compiler.

9.3.1 MICRO FOCUS COBOL

In der Source-Datei sind keine speziellen Einstellungen notwendig. Die Dialog Manager Definitionen werden durch das Kommando "COPY" verfügbar.

Einstellungen	keine
Copy-Dateien	`COPY <FILENAME>`

Flags	**-u** für interpretierbare Dateien **-x** für Objekt-Dateien
Übersetzung	`cob-c -x <file>`

Anmerkung

In einer Microsoft Windows Umgebung muss beim Übersetzen der COBOL-Module zusätzlich der Schalter `/LITLINK` gesetzt sein, sonst kann das COBOL-Programm den Dialog Manager nicht aufrufen.

Die Umgebungsvariable `COBDIR` muss das Include-Directory für die COBOL Copy-Dateien beinhalten.

9.4 Linken

Um die COBOL-Anwendung zusammen zu linken, müssen zunächst die erzeugten C-Source-Dateien mit Hilfe des C-Compilers übersetzt werden. Nach der erfolgreichen Übersetzung der COBOL-Programme müssen diese mit den entsprechenden Libraries des COBOL- und des Fenstersystems sowie der Dialog Manager Library und dem entsprechendem COBOL-Modul gelinkt werden.

Umgebung	Compiler	Einsatz	Dateien
PC Windows	cob	lokal	ufcob.obj dm.lib +
PC Windows	vcob	Client	ufcob.obj dm.lib dmndx.lib +
UNIX	cob	lokal/Motif	ufcob.obj libIDM.a +
UNIX	cob	Server	ufcobnet.o libIDMnet.

Dabei bedeutet in der Tabelle "+", dass hier noch die jeweiligen Libraries des Fenstersystems dazu gelinkt werden müssen. Überall muss die Library libIDMinit.a bzw dminit.lib hinzu gelinkt werden.

Für das Linken stehen systemspezifische Shell-Skripts, Jobs bzw. Batchdateien zur Verfügung, die das Zusammenlinken der Applikation übernehmen. Diese Dateien werden zusammen mit den Beispiel ausgeliefert.

Diese Dateien heißen bei

UNIX und MICRO FOCUS COBOL	ufdmlink*
PC und MICRO FOCUS COBOL	siehe Beispiel-Makefile

*) Wird hier beim Aufruf der Schalter -net angegeben, wird das Programm als Serverprozess ohne Display-Komponente zusammen gelinkt.

9.5 Beispiele für Makefiles

In dem folgenden Kapitel wird auf die einzelnen Umgebungen in Form von aufgelisteten Makefiles genauer eingegangen. Diese Makefiles sind Bestandteil der Distribution und brauchen daher nicht von Ihnen eingetippt werden. In dem Kapitel "Besonderheiten" werden immer die Besonderheiten des Systems erwähnt, im Kapitel "Allgemeiner Vereinbarungsteil" der Anfangsteil eines Makefiles dargestellt, im Kapitel "Übersetzen" eine Kommandozeile, wie die COBOL-Source übersetzt werden kann, im Kapitel "Übersetzen der C-Source" wie die generierte C-Datei übersetzt werden muss und abschließend im Kapitel "Linken", wie die Anwendung zusammen gelinkt werden muss. Dieses wird immer anhand eines Beispiels dargestellt.

9.5.1 MICRO FOCUS COBOL unter MICROSOFT WINDOWS

9.5.1.1 Besonderheiten

Beim Übersetzen der COBOL-Sourcen muss der Schalter **/LITLINK** gesetzt sein, beim Übersetzen der C-Sourcen die Schalter **-DUFCOB** und **-DUFCOB_LINK**.

Damit die COBOL-Programme den Dialog Manager aufrufen können, muss unbedingt das Modul **mfc6intw.obj** des COBOL-Compilers hinzugebunden werden. Wenn das Modul **coboljmp.obj** hinzugebunden wird, können die Interface-Funktionen im Dialog Manager ohne einen vorangestellten Unterstrich "_" aufgerufen werden.

9.5.1.2 Allgemeiner Vereinbarungsteil

```
#####################################################################

TOPDIR=\idm

#####################################################################

BINDIR=$(TOPDIR)
LIBDIR=$(TOPDIR)\lib
IDMCOBDIR=$(TOPDIR)\cobol
INCLDIR=$(TOPDIR)\include

#####################################################################

IDM=wx $(BINDIR)\idm

#####################################################################

CC=cl
COPTS= -AL -Zp -Gct5 -W4 -nologo -G3 -f- -GA -Owcegilnot \
-Ob1 -Gsy
```

```
COUTOPTS= -Fo
CDEFS=-DDOS -DPC -DVERMAJOR=3 -DVERMINOR=1 -DMSW -DMSC \
-D_WINDOWS -DUFCOB -DUFCOB_LINK
CINCL=-I. -I$(INCLDIR) -I$(TOPDIR)\cobol
CFLAGS=$(CDEFS) $(CINCL) $(LOCAL_DEFINES)
CL=$(COPTS)
Cobol=cobol
CobolC=c:\cobol\lib\mfc6intw.obj
GENPRG=$(TOPDIR)\cobol\gencobfx

LD=link
LDFLAGS=/ONERROR:N /ALIGN:32 /NOE /SEG:512 /MAP /NOD
STACKSIZE=27648
HEAPSIZE=5120

RC=rc
RCFLAGS= -T -K -30

CP=copy
MV=rename
DEL=del

#####################################################################

SYSSTARTUP=
SYSLIBS= llibcew.lib libw.lib
SYSLIBS_DLL=llibcew.lib libw.lib
COBLIBS= lcobolw.lib lcobol.lib cobw.lib
DDELIBS= ddeml.lib
MATHLIBS=
SOCKETLIBS=

#####################################################################

STARTUP_DEV=$(IDMCOBDIR)\ufcob.obj
BIND_COB=$(IDMCOBDIR)\coboljmp.obj
DMLIBS=$(LIBDIR)\dminit.lib $(LIBDIR)\dm.lib
```

9.5.1.3 Übersetzen einer COBOL-Source

```
obj\table.obj: src\table.cob
    @echo [cobol table]
    @$(Cobol) src\table.cob, obj\table.obj,,,/LITLINK
```

9.5.1.4 Übersetzen der generierten C-Source

```
obj\tablec.obj : src\tablec.c
    $(CC) -c $(CFLAGS) -DUFCOB -DUFCOB_LINK src\tablec.c
```

9.5.1.5 Linken der Anwendung

```
exe\table.exe: $(TABLE_OBJ)
    @echo [generating table.lnk]
    @echo $(SYSSTARTUP) +> table.lnk
    @echo $(STARTUP_DEV) +>> table.lnk
    @echo $(BIND_COB) +>> table.lnk
    @echo obj\table.obj +>> table.lnk
    @echo obj\putaddr.obj +>> table.lnk
    @echo obj\getaddr.obj +>> table.lnk
    @echo obj\table_.obj +>> table.lnk
    @echo obj\FuncTabl.obj +>> table.lnk
    @echo obj\tablec.obj +>> table.lnk
    @echo obj\FillTab.obj +>> table.lnk
    @echo obj\TabFunc.obj +>> table.lnk
    @echo $(COBOLC) >> table.lnk
    @echo exe\table.exe $(LDFLAGS)>> table.lnk
    @echo table.map>> table.lnk
    @echo $(DMLIBS) +>> table.lnk
    @echo $(SYSLIBS) +>> table.lnk
    @echo $(COBLIBS)>> table.lnk
    @echo table.def>> table.lnk
    @echo [linking table]
    @$(LD) @table.lnk
    @$(RC) $(RCFLAGS) $@
    @$(DEL) table.*

table.def: mkdef.bat
    @echo [generating table.def]
    mkdef table $(STACKSIZE) $(HEAPSIZE) $(TOOLKIT)
```

Dabei ist **mkdef.bat** das mit dem Dialog Manager ausgelieferte Skript und sieht wie folgt aus:

```
@echo off

if not X%3 == X goto OK
    echo usage: mkdef "def-name" "stacksize" "heapsize"
    goto END

:OK
echo ; module-definition file for DM -- used by LINK.EXE > %1.def
echo NAME         %1 >>%1.def
```

```
echo DESCRIPTION  'ISA Dialog Manager Application' >>%1.def
echo EXETYPE WINDOWS;required for Windows appl >>%1.def
echo STUB 'WINSTUB.EXE' ; >>%1.def
echo ;CODE can be moved in memory and discarded >>%1.def
echo CODE  PRELOAD MOVEABLE DISCARDABLE >>%1.def
echo ;DATA must be MULTIPLE if program can run more than once >>%1.def
echo DATA  PRELOAD FIXED MULTIPLE >>%1.def
echo HEAPSIZE %3 >>%1.def
echo STACKSIZE %2; recommended minimum for DM appl >>%1.def

echo ; All functions that will be called by any >>%1.def
echo ; routine Windows MUST be exported. >>%1.def

echo EXPORTS >>%1.def
echo   YITOPWINPROCHOOK@1  ; subclass function top-window-function >>%1.def
echo   YIWINPROCHOOK@2  ; function to subclass all other window-functions
>>%1.def
echo   YIWINDOWPROC@3  ; window-function for window >>%1.def
echo   YIWINDOWFRAMEPROC @4  ; window-function for mdi-frame-window >>%1.def
echo   YIWINDOWCHILDPROC @5  ; window-function for mdi-child-window >>%1.def
echo   YICLIENTWINPROCHOOK @6  ; function to subclass mdi-client-window-
functions >>%1.def
echo   YINODEBOXPROC @7  ; window-function for nodebox >>%1.def
echo   YICANVASPROC        @8  ; window-function for canvas >>%1.def
echo   YPRINTABORTDLG      @9  ; window-Dialog for printing >>%1.def
echo   YPRINTABORTPROC     @10 ; window-function for printing >>%1.def
echo   YITABLEPROC         @11 ; window-function for tablefield >>%1.def
echo   YITABLEEDITPROCHOOK @12 ; function to subclass edittext assigned to
tablefield >>%1.def

if not X%5 == XEDT goto END
echo   YEDFONTENUM         @13 ; function to get fontnames for editor >>%1.def

:END
```

9.5.2 MICRO FOCUS COBOL unter UNIX

9.5.2.1 Besonderheiten

Auf den Unix-Systemen kann das dynamische Nachladen von COBOL-Modulen vom COBOL-Runtimesystem ausgenutzt werden. Dazu müssen die Namen der im Dialog definierten Funktionen den Namen der jeweiligen COBOL-Modulen und den darin enthaltenen Funktionen entsprechen. Wenn dieses Verhalten ausgenutzt werden soll, müssen die COBOL-Sourcen nur in ein Zwischenformat ".gnt" oder ".int" übersetzt werden und brauchen nicht zu der Anwendung hinzu gelinkt werden. Zum eigentlichen Ablaufzeitpunkt müssen diese Zwischendateien vom COBOL-Runtimesystem gefunden

werden. Dazu muss die Umgebungsvariable `COBLIB` entsprechend gesetzt werden. Genauso besteht aber auch die Möglichkeit die COBOL-Sourcen zu Objekt-Dateien zu übersetzen und eine normale Anwendung zusammen zu linken.

Zum Übersetzen der generierten C-Datei muss der Schalter **-DUFCOB** gesetzt sein. Wenn ein ohne das COBOL-Runtimesystem ablauffähiges Programm gelinkt werden soll, muss beim Übersetzen der generierten C-Datei zusätzlich der Schalter **-DUFCOB_LINK** gesetzt werden.

9.5.2.2 Allgemeiner Vereinbarungsteil

```
shell= /bin/sh

NOECHO=
IDMSRC=/usr/local/IDM
include $(IDMSRC)/lib/IDM/MakeDefs

IDM_INC=-I$(IDMSRC)/include
IDM_LIB=$(IDMSRC)/lib/libIDM.a
GENCOBFX=$(IDMSRC)/lib/IDM/gencobfx
IDM=$(IDMSRC)/bin/idm
IDMCOBCOPY=$(IDMSRC)/include
IDM_BINDING=$(IDMSRC)/lib/ufcob.o
COBDEFS=-x -c
COBCPY=$$COBDIR/srclib:$(IDMCOBCOPY)::
COB=COBCPY="$(COBCPY)";export COBCPY; cob

VCDMLINK=ufdmlink -root $(IDMSRC)
VCDMLIBS=$(WINLIBS) $(SYSLIBS)
```

9.5.2.3 Übersetzen einer COBOL-Source

```
table.o:table.cbl
        @echo [Compiling table]
        $(NOECHO) $(COB) $(COBDEFS) table.cbl
```

9.5.2.4 Übersetzen der generierten C-Source

```
tablec.o:tablec.c
        $(CC) $(COPTS) $(CDEFS) $(IDM_INC) -c tablec.c
```

9.5.2.5 Linken der Anwendung

Wenn ein Serverstub zusammen gelinkt werden soll, muss der Schalter **-net** beim Aufruf des Skripts **ufdmlink** gesetzt sein.

```
tablenet:table.o tablec.o table_.o
```

```
        @echo [Linking tablenet]
        $(NOECHO) $(VCDMLINK) -net -o tablenet table.o
            tablec.o table_.o
```

Wenn eine lokale Anwendung zusammen gelinkt werden soll, muss der Schalter **-dvl** beim Aufruf des Skripts **ufdmlink** gesetzt sein.

```
table:table.o tablec.o table_.o
        @echo [Linking table]
        $(NOECHO) $(VCDMLINK) -dvl -o table table.o tablec.o
                table_.o
```

Dabei hat das Shell-Skrip **ufdmlink** folgendes Aussehen:

```
#!/bin/sh -e
#
#  Link Micro Focus COBOL executable for use with ISA Dialog
# Manager.Call this script without arguments to see a brief
# description or with -help for full documentation.
#

    cobswitches=""
    isasrc="FALSE"
    idmroot="/usr"
    version=""
    winlibs=""
    syslibs=""
    target=""

    #
    #  Parse command line.
    #

    while [ $# -gt 0 ]
    do
    case $1 in
        -dvl|-rls|-net)
        if [ "$version" = "" ]
        then
            version="$1"
        else
          echo >&2 "$0: Options -dvl, -rls and -net are mutually exclusive"
            exit 1
        fi
        ;;

        -root)
        shift
        idmroot="$1"
```

```
    ;;

    -n|-d)
    cobswitches="$cobswitches $1"
    ;;
    -o)
    shift
    target="-o $1"
    ;;

    *)
    break
    ;;
esac
shift
done

#
#  Check required arguments.
#

if [ "$version" = "" ]
then
echo >&2 "$0: One of the switches -dvl, -rls and -net must be given"
exit 1
fi

#
#  Determine location of files.
#

case $isasrc in
FALSE)
    case $version in
    -dvl)
    binding="$idmroot/lib/ufcob.o"
    library="$idmroot/lib/libIDM.a \
            $idmroot/lib/libIDMinit.a"
    ;;

    -rls)
    binding="$idmroot/lib/ufcobrt.o"
    library="$idmroot/lib/libIDMrt.a \
            $idmroot/lib/libIDMinit.a"
    ;;
```

```
        -net)
        binding="$idmroot/lib/ufcobnet.o"
        library="$idmroot/lib/libIDMnet.a \
                $idmroot/lib/libIDMinit.a"
        ;;
        esac
        ;;
    esac

args="$binding $args $library $*"

    #
    #  Link the program.
    #

    echo cob -x -F $target $cobswitches $args
    cob -x -F $target $cobswitches $args
```

Index

D

E

F

G

H