ISA Dialog Manager

COBOL Interface

A.06.03.d

In this manual the API (application programming interface) of the ISA Dialog Manager for applications written in COBOL is described. The data types, all API functions of the COBOL interface, and compiling and linking of the applications are elucidated.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Germany

Notation Conventions

DM will be used as a synonym for Dialog Manager.

The notion of UNIX in general comprises all supported UNIX derivates, otherwise it will be explicitly stated.

< >	to be substituted by the corresponding value
color	keyword
.bgc	attribute
{ }	optional (0 or once)
[ ]	optional (0 or n-times)
<A> | <B>	either <A> or <B>

Description Mode

All keywords are bold and underlined, e.g.

variable **integer** **function**

Indexing of Attributes

Syntax for indexed attributes:

[I]

[I,J] meaning [row, column]

Identifiers

Identifiers have to begin with an uppercase letter or an underline ('_'). The following characters may be uppercase or lowercase letters, digits, or underlines.

Hyphens ('-') are ***not*** permitted as characters for specifying identifiers.

The maximal length of an identifier is *31* characters.

Description of the permitted identifiers in the Backus-Naur form (BNF)

<identifier>	::=	<first character>{<character>}
<first character>	::=	_ | <uppercase>
<character>	::=	_ | <lowercase> | <uppercase> | <digit>

<digit>	::=	1 | 2 | 3 | … 9 | 0
<lowercase>	::=	a | b | c | … x | y | z
<uppercase>	::=	A | B | C | … X | Y | Z

Table of Contents

Notation Conventions 3

Table of Contents 5

1 Introduction 11

2 COBOL Interface for Micro Focus Visual COBOL 12

2.1 Unicode Support 12

2.1.1 Activating Unicode 12

2.1.2 Enhancement of CopyFiles 13

2.1.3 Call Parameters of COBOL Functions 13

2.1.4 Functions with Records as Parameters 13

2.2 Data Type DT-anyvalue 14

2.3 Support of Collections 14

2.3.1 Data Function in COBOL 14

2.3.2 Structure DM-ValueIndex 15

2.3.3 Functions for Working with Collections 15

3 Basic Working Method 16

3.1 Definition of the Dialog and of the Operation Control 16

3.2 Definition of the COBOL Programs 23

3.2.1 Mapping the Dialog Data Types 23

3.2.2 Central Data Structures 24

3.2.3 Main Program 24

3.2.3.1 Local Applications 25

3.2.3.2 Distributed Applications 27

3.2.4 Auxiliary Means for the COBOL Programming 28

3.2.5 Functions for the Tablefield 29

3.2.5.1 Function FILLTAB 30

3.2.5.2 Function TABFUNC 34

3.2.6 Functions with Records as Parameters 37

3.2.6.1 Function GETADDR 37

3.2.6.2 Function PUTADDR 38

3.3 Compiling and Linking 39

4 Data Types ... 40
4.1 Mapping the Dialog Data Types on COBOL Data Types ... 40
4.1.1 Basic Data Types ... 40
4.1.2 Records ... 41
4.1.3 Collection Data Types ... 41
4.2 Standard Argument ... 42
4.3 Handling of String Parameters ... 45
4.4 Using the anyvalue Data Type ... 46
4.5 Inquiring and Changing Attributes ... 47
4.5.1 DM-Value ... 48
4.5.2 DM-ValueIndex ... 52
4.5.3 DM-ValueArray ... 52
4.5.4 Data Types in the Structures DM-Value and DM-ValueArray ... 57
4.6 Object Callback ... 59
4.7 Reloading Functionality ... 61
4.8 Data Function Structure DM-Datafunc-Data ... 63
4.9 NULL Object ... 65
4.10 Access to Attributes ... 65
4.11 Return Values of the DM-functions DM-success / DM-error ... 67
5 Linking the COBOL Program ... 68
5.1 Main Program ... 68
5.1.1 Local COBOL Programs ... 68
5.1.2 Distributed COBOL Programs ... 68
5.2 Realization of COBOL Functions ... 68
5.3 Object Callback Functions ... 71
5.4 Reloading Functions ... 71
5.5 Data Functions ... 71
5.6 Canvas Functions ... 72
5.7 Format Functions ... 72
5.8 Transfer of Function Addresses ... 72
5.8.1 Functions without Records as Parameters ... 72
5.8.1.1 Structure of the Function Definition File ... 73
5.8.1.2 Generating the Function Definition File ... 73
5.8.1.2.1 Unix ... 73
5.8.1.2.2 Microsoft Windows ... 74
5.8.1.3 Example ... 74
5.8.1.4 BindThruLoader Functionality ... 75
5.8.2 Functions with Records as Parameters ... 76
5.8.2.1 Generating the Trampoline Module ... 76

5.8.2.2 Compiling the C Module 77
5.8.2.3 Example 77
5.8.3 Dynamic Binding of Record Functions 80

6 Functions of the COBOL Dialog Manager Interface 82

6.1 Overview of Functions 82
6.2 Initializing and Starting the Dialog Manager 85
6.3 Finishing the ISA Dialog Manager 85
6.4 Access Functions 86
6.4.1 Accessing the Dialog Manager Identifiers 86
6.4.2 Accessing Object Attributes 86
6.4.3 Creating and Destroying Objects 86
6.4.4 Utilities 87
6.4.5 Access to Listbox and Tablefield 88
6.5 Working with Collections 88
6.6 Error Handling 89
6.6.1 Information in the Error Code 89
6.7 Special Functions 90
6.8 Functions in Alphabetical Order 91
6.8.1 COBOLMAIN 91
6.8.1.1 COBOLAPPINIT 92
6.8.1.2 COBOLAPPFINISH 93
6.8.2 DM_BindCallBacks 95
6.8.3 DM_BindFunctions 95
6.8.4 DM_BootStrap 95
6.8.5 DMcob_CallFunction 96
6.8.6 DMcob_CallMethod 98
6.8.7 DMcob_CallRule 101
6.8.8 DMcob_Control 104
6.8.9 DMcob_CreateFromModel 108
6.8.10 DMcob_CreateObject 110
6.8.11 DMcob_DataChanged 112
6.8.12 DMcob_DeleteArgString 114
6.8.13 DMcob_Destroy 115
6.8.14 DMcob_DialogPathToID 116
6.8.15 DMcob_ErrMsgText 118
6.8.16 DMcob_EventLoop 119
6.8.17 DMcob_ExecRule 120
6.8.18 DMcob_FatalAppError 122
6.8.19 DMcob_FreeVector 123
6.8.20 DMcob_GetArgCount 124

6.8.21 DMcob_GetArgString 125
6.8.22 DMcob_GetValue 128
6.8.23 DMcob_GetValueBuff 130
6.8.24 DMcob_GetVector 132
6.8.25 DMcob_GetVectorValue 136
6.8.26 DMcob_Initialize 138
6.8.27 DMcob_InitVector 139
6.8.28 DMcob_LGetValueBuff 141
6.8.29 DMcob_LGetValueLBuff 144
6.8.30 DMcob_LGetVector 147
6.8.31 DMcob_LGetVectorValue 151
6.8.32 DMcob_LGetVectorValueBuff 153
6.8.33 DMcob_LInitVector 155
6.8.34 DMcob_LoadDialog 157
6.8.35 DMcob_LoadProfile 159
6.8.36 DMcob_LSetValueBuff 160
6.8.37 DMcob_LSetValueLBuff 162
6.8.38 DMcob_LSetVector 164
6.8.39 DMcob_LSetVectorValue 168
6.8.40 DMcob_LSetVectorValueBuff 170
6.8.41 DMcob_OpenBox 171
6.8.42 DMcob_OpenBoxBuff 173
6.8.43 DMcob_PathToID 176
6.8.44 DMcob_PutArgString 178
6.8.45 DMcob_QueryBox 179
6.8.46 DMcob_QueryError 181
6.8.47 DMcob_QueueExtEvent 182
6.8.48 DMcob_ResetValue 184
6.8.49 DMcob_SetValue 185
6.8.50 DMcob_SetValueBuff 187
6.8.51 DMcob_SetVector 189
6.8.52 DMcob_SetVectorValue 192
6.8.53 DMcob_ShutDown 194
6.8.54 DMcob_StartDialog 195
6.8.55 DMcob_StopDialog 196
6.8.56 DMcob_TraceMessage 197
6.8.57 DMcob_ValueChange 198
6.8.58 DMcob_ValueChangeBuffer 200
6.8.59 DMcob_ValueCount 202
6.8.60 DMcob_ValueGet 204
6.8.61 DMcob_ValueGetBuffer 206

6.8.62 DMcob_ValueGetType 208
6.8.63 DMcob_ValueIndex 209
6.8.64 DMcob_ValueInit 211
6.8.65 DMmfviscob_BindThruLoader 213
6.8.66 DMufcob_BindThruLoader 214
6.9 Options of the Interface Functions 214

7 Attributes and Definitions 219

7.1 Attribute Definitions and their Data Types 219
7.2 Class Definition 219

8 COBOL in Distributed Environment 221

9 Compiling and Linking Dialog Manager Programs 231

9.1 Copy Files 231
9.2 Compiling the Generated C Files 232
9.3 Compiling the COBOL Module 232
9.3.1 Micro Focus COBOL 232
9.4 Linking 233
9.5 Examples for Makefiles 234
9.5.1 Micro Focus COBOL on Microsoft Windows 234
9.5.1.1 Peculiarities 234
9.5.1.2 General Part of Declarations 234
9.5.1.3 Compiling a COBOL Source 235
9.5.1.4 Compiling the Generated C Source 235
9.5.1.5 Linking the Application 236
9.5.2 Micro Focus COBOL on Unix 237
9.5.2.1 Peculiarities 237
9.5.2.2 General Part of Declarations 238
9.5.2.3 Compiling a COBOL Source 238
9.5.2.4 Compiling a Generated C Source 238
9.5.2.5 Linking the Application 238

Index 243

1 Introduction

This manual describes the application programming interface (usually called "API") offered by the Dialog Manager to application programs written in the COBOL language. Since major differences exist in the way COBOL compilers interface to other languages e.g. the C programming language, there are several APIs available. This manual describes the implementation and use of the Dialog Manager API to such COBOL programs compiled with MICRO FOCUS Compiler.

Apart from certain control functions that influence the overall operation of the Dialog Manager, the API offers the fundamental functionality available in the Rule Language of the application program. Thus the application program is capable of accessing and modifying data of the running dialog. This can best be shown in a short example:

An expression written in the dialog rules as:

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

gets the data currently present for the attribute *.content* of the object *EdittextTitle* and sets the attribute *.title* of the object *MainWindow* to these data.

The same could also be achieved by using the API calls

DMcob_GetValue and **DMcob_SetValue**

by an application program.

The intended audience for this manual are programmers familiar with the Dialog Manager development environment, the MICRO FOCUS COBOL system. We also presume good understanding of the operating system being used and the tools the operating system provides.

Please pay special attention to the chapter "Calling Subprograms". You should completely understand the rules governing the loading of programs in the COBOL system, since these features are basic to the API described here.

2 COBOL Interface for MICRO FOCUS VISUAL COBOL

Availability

IDM for MICROSOFT WINDOWS from IDM version A.06.01.d.

2.1 Unicode Support

The COBOL Interface for MICRO FOCUS VISUAL COBOL supports Unicode as character encoding. A Unicode string is represented by MICRO FOCUS VISUAL COBOL as *National Character*, where the encoding complies with the IDM code page *CP-utf16*. Within MICRO FOCUS VISUAL COBOL this is represented as *PIC N NATIONAL*.

2.1.1 Activating Unicode

The use of Unicode is activated by setting the application code page. This can be accomplished through a command line option or within the application using the DMcob_Control function:

```
...
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.
...
```

Notes

- COBOL compiler directives can be used to control the code page for *National Character*. The set application code page must match this coding. A National Character requires 2 bytes. Currently, the IDM code pages *CP-utf16*, *CP-utf16b*, *CP-utf16l* and *CP-utfwin* use a 2-byte representation.
- There are enhancements in the IDM to selectively change the code page used for individual functions of an application. This enables an incremental transition to Unicode.

2.1.2 Enhancement of CopyFiles

The copy files are defined in a way that texts can be stored either as Character or as National Character. The name of the Character entry remains. To access the National Character entry, a "-u" has to be appended to the name.

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.`

For the following structure elements of the ValueRecord, the length or size is defined as the number of characters, not of bytes:

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

The same applies to all other structures based on the ValueRecord.

Note

The IDM reads or writes the respective values based on the currently valid application code page.

2.1.3 Call Parameters of COBOL Functions

For all texts either Character or National Character can be used. The IDM interprets the text according to the currently valid application code page. The length specification refers to the number of characters, not of bytes. This applies both to the IDM interface functions as well as to the COBOL functions called by the IDM. It is important that the currently used application code page complies with the text definition.

2.1.4 Functions with Records as Parameters

For functions with ***record*** objects as parameters, the **pidm** application creates a copy file by means of the command line option **+writetrampolin**. In order not to interfere with existing applications, these copy files are created without support for *National Character* by default. Support for it has to be explicitly specified by using the **-mfviscob-u** option instead of the COBOL compiler option **-mfviscob:**

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

The generated copy file may then contain either character or national character. Again, the IDM will access based on the current application code page.

2.2 Data Type DT-anyvalue

The data type *DT-anyvalue* is supported through the *POINTER* data type of MICRO FOCUS VISUAL COBOL.

See also

Chapter “Using the anyvalue Data Type”

2.3 Support of Collections

The colection data types of the IDM and the related functions for (managed) IDM values (**Managed DM-Values**) are supported through the pointer data type (POINTER) of MICRO FOCUS VISUAL COBOL.

DM Data Type	Visual COBOL Data Type
hash	POINTER
list	POINTER
matrix	POINTER
refvec	POINTER
vector	POINTER

A **Managed DM-Value** is passed as pointer to MICRO FOCUS VISUAL COBOL:

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

To use such a value within the **DM-Value** structure, the value is copied to *DM-value-pointer* and *DM-datatype* is set to *DT-anyvalue*.

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

2.3.1 Data Function in COBOL

With the support of the collection data types, it is also possible to implement data functions in COBOL.

See also

Chapter “Data Functions”

Chapter “Data Function Structure DM-Datafunc-Data”

Function DMcob_DataChanged

2.3.2 Structure DM-ValueIndex

This structure allows for a full-fledged index argument. It is used by the **DMcob_Value*** functions of the COBOL Interface for MICRO FOCUS VISUAL COBOL to process the **collection data types**. However, the structure is available for all supported COBOL variants and is not restricted to these application areas. It is identical to the DM-Value structure.

2.3.3 Functions for Working with Collections

- DMcob_ValueChange
  With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection.
- DMcob_ValueChangeBuffer
  With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection. Unlike **DMcob_ValueChange**, this function provides a buffer for string return values that is larger than the standard buffer.
- DMcob_ValueCount
  Returns the number of values in a collection (without the default values). It is also possible to return the index type or the highest index value.
- DMcob_ValueGet
  This function allows to retrieve a single element value that belongs to a defined index from collections.
- DMcob_ValueGetBuffer
  This function allows to retrieve a single element value that belongs to a defined index from collections. Unlike **DMcob_ValueGet**, this function provides a buffer for string return values that is larger than the standard buffer.
- DMcob_ValueGetType
  This function queries the data type of a value managed by the IDM.
- DMcob_ValueIndex
  This function can be used to determine the corresponding index for a position in a collection.
- DMcob_ValueInit
  With this function a value can be converted into a local or global value reference managed by the IDM. This allows the further manipulation of the value by **DMcob_Value…()** functions and its transfer as parameter or return value.

3 Basic Working Method

In this chapter we describe the way how the COBOL interface works in principle. You should get an idea of the philosophy by which the programs written with the Dialog Manager have to be realized.

When realizing the application programs you should always bear in mind Seeheim's layer model which demands a strict separation of the various software layers. In doing so, the presentation layer as well as the dialog layer can be realized almost completely in the Rule Language when using the Dialog Manager. Only the access to objects with listing character (tablefield, listbox and poptext) should be realized in COBOL due to efficiency. The other COBOL functions ideally should not have any information on the surface. Therefore they should always get all necessary information in the form of parameters and then should not execute any calls to the Dialog Manager; in other words they should not contain any "Dmcob_"functions. If functions are realized in this way, the actual application logic and processing is contained in the COBOL functions, which are independent of the used surface. This working method is obligatory when using the Distributed Dialog Manager (DDM) since the function call via the network is comparatively expensive and thus inefficient.

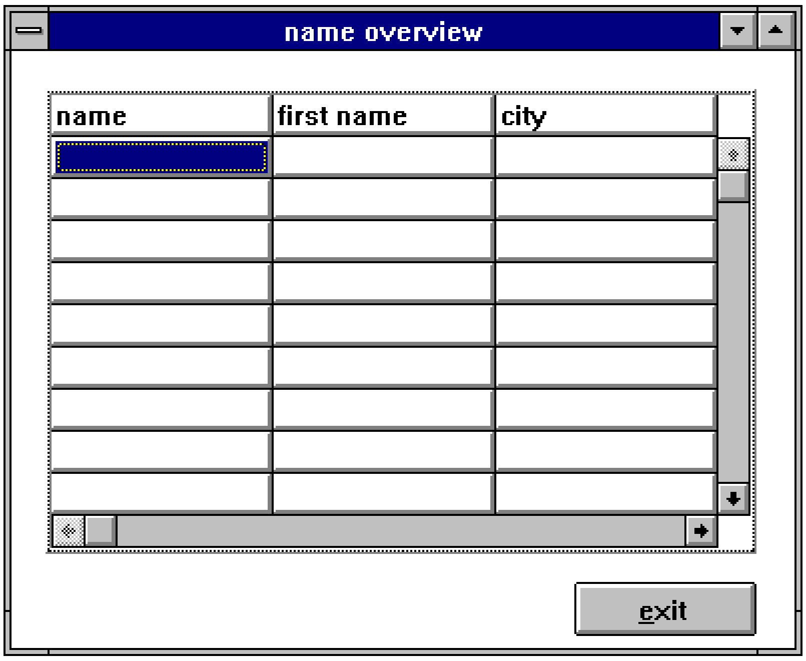
The only exception is the actual main program of the application because here the Dialog Manager is accessed and therefore information about the surface is needed.

To be able to write the COBOL program which suits the developed surface, first some data structures in the COBOL interface have to be clarified and the reproduction of the datatypes in the dialog for the datatypes in COBOL have to be defined. An example which stretches over the following chapters shall illustrate this basic working method.

3.1 Definition of the Dialog and of the Operation Control

In the Dialog Manager editor a dialog system shall be built which first offers the user an overview of the names which he can then modify in a sub-window. Since the overview may contain a lot of names, they should always be loaded and displayed only partially. To do so, a reloading function has to be defined in the table element. The entire data belonging to a name are displayed in a separate sub-window defined as a dialogbox where the user can modify them.

The overview window looks as follows:



In the dialog script this window is defined as follows:

```
window WnOverview
{
  .visible false;
  .active false;
  .xleft 359;
  .width 50;
  .ytop 2;
  .height 16;
  .iconic false;
  .iconifyable true;
  .title "name overview";

  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] "first name";
    .content[1,3] "city";
    .xraster 10;
    .yraster 16;
  }
  child pushbutton PENDE
  {
    .xleft 36;
    .width 11;
    .yauto 1;
    .text "&exit";
  }
}
```

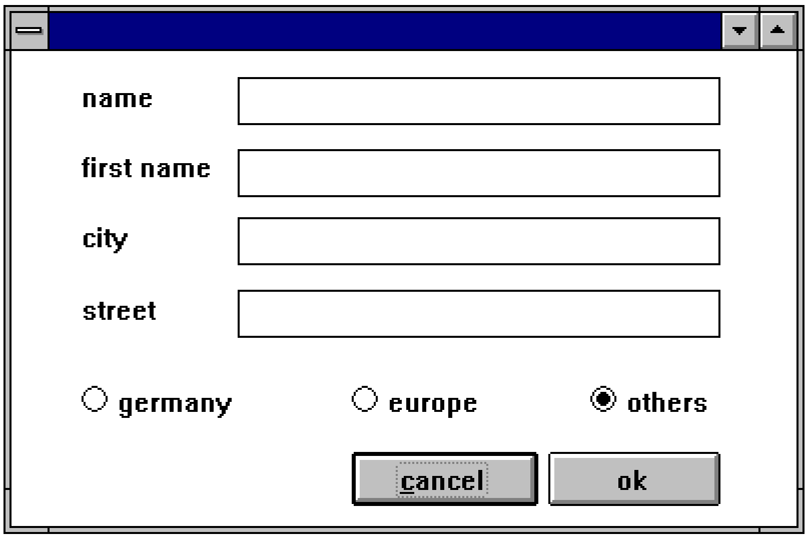
The definition of the window in which the data can be modified looks as follows:

```
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 "first name";
    .E.content "";
  }
  child Eintrag Lcity
  {
    .ytop 4;
    .S.text "city";
    .E.content "";
  }
  child Eintrag Lstreet
  {
    .ytop 6;
    .S.text "street";
    .E.content "";
  }
  child pushbutton PbCancel
  {
    .xleft 22;
    .width 11;
    .ytop 10;
    .text "&cancel";
  }
  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 "germany";
    .LandesCode := 1;
  }
  child radiobutton EUROPE
```

```
  {
    .userdata 2;
    .xleft 19;
    .ytop 8;
    .posraster true;
    .sizeraster true;
    .text "europe";
    .LandesCode := 2;
  }
  child radiobutton OTHER
  {
    .userdata 3;
    .xleft 33;
    .ytop 8;
    .text "others";
    .LandesCode := 3;
  }
}
```

The window then looks as follows:



For an easier definition of this window a model has been introduced which consists of a groupbox with a static and an editable text as children. The corresponding definition in the dialog script looks as follows:

```
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;
  }
}
```

For the communication with the COBOL program the following functions have been defined:

» TABFUNC
  This function is a reloading function which reloads the data in the tablefield.
  ```
  function cobol contentfunc TABFUNC();
  ```

» FILLTAB
  By means of this function the tablefield is initially filled partly.
  ```
  function cobol void FILLTAB(object Table input, integer Count input);
  ```

» GETADDR
  This function loads the entire data belonging to a row and shall transfer them to the dialog.
  ```
  function cobol void GETADDR(record Address input output);
  ```

» PUTADDR
  This function shall accept the data changed by the dialog and save them.
  ```
  function cobol void PUTADDR(record Address input);
  ```

The last two functions contain as a parameter one record respectively whose elements are defined as links on the corresponding elements in the window.

```
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;
}
```

To call the functions, the following rules have been defined:

» On starting the program the tablefield is filled partially and the relevant window is made visible.
  ```
  on dialog start
  {
      FILLTAB(T1, 20);
      T1.rowcount := 300;
  ```

```
    WnUebersicht.visible := true;
}
```

» By selecting the End pushbutton the program will be quit. The program will also be quit, if the main window is closed by the Closing entry in the system menu.

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

» If one of the radiobuttons is selected, its value will be noted for the relevant window.

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

» On selecting the Cancel pushbutton the relevant window will be closed.

```
!!If the cancel pushbutton is selected,
!!simply close the relevant window.
on PbCancel select
{
    this.window.visible := false;
}
```

» If the OK pushbutton is selected, the function PUTADDR will be called and the values will be transmitted.

```
!!If the ok button is selected, close the window and
!!adopt the changes in the COBOL program.
on PbOk select
{
    PUTADDR(Address);
    this.window.visible := false;
}
```

» If the tablefield is selected by a doubleclick, all data belonging to this row will be transferred by the function GETADDR and the detail window will be opened.

```
!!If a doubleclick on the interior of the table element is
!!done, the relevant entry shall be processable in the
!!separate  window. Thus, the COBOL function has to be called
!!which gets the data and passes them to the new window.
on T1 dbselect
{
    !!Check first whether something has really been selected
    !!in the table.
```

```
        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];
            !! Take this entry and
            !! transfer it to the COBOL programm.
            GETADDR(Address);
            WnName.title := ("Name Number: "
                + itoa((first(this.activeitem)  1)));
            WnName.visible := true;
        else
            !! no useful entry found. Make a sound.
            beep();
        endif
    }
```

3.2 Definition of the COBOL Programs

To be able to write a COBOL program for this defined surface, you have to know the following:

- how dialog data types are mapped onto COBOL data types
- the important DM data structures which are used for the communication between the COBOL program and the Dialog Manager
- how the main program looks, if the application shall be written with the Dialog Manager
- how the COBOL subprograms look which have been called out of the Dialog Manager.

Each of these points are explained in the following chapters.

3.2.1 Mapping the Dialog Data Types

Based on the dialog description the data types used there can be mapped onto the data types usable in COBOL. This dialog mapping on COBOL is unambiguous; the next incomplete table provides some information:

Dialog Data Type	COBOL Data Type
object	PIC 9(9) binary.
integer	PIC 9(9) binary.
string[??]	PIC X(??).
boolean	PIC 9(4) binary.

3.2.2 Central Data Structures

In the COBOL interface of the Dialog Manager there are two central data structures via which the communication between application and the Dialog Manager is established.

The most important data structure is called **DM-StdArgs**, which the application has to transfer to every function called by the COBOL interface. In this structure the Dialog Manager informs the application whether any errors have occurred on calling the function. Furthermore, the application can inform the Dialog Manager about options which shall be considered during the realization of individual functions. The fields for the string handling *DM-truncspaces*, *DM-getsep* and *DM-setsep* should be only set before calling the initialization function, thus should only be set globally; in other words, it should not be changed during the course of the program. See also chapter “Handling of String Parameters”.

The DM-StdArgs structure thus looks as follows:

```
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.
```

The second important structure is the DM-Value structure, for in this structure the values are transferred from the COBOL program to the Dialog Manager. This structure contains the object, the attribute and the values an attribute can adopt. In the example this structure is needed to fill the tablefield and it will only be defined there.

3.2.3 Main Program

Programs which shall work with the Dialog Manager need special main programs. These are equal in principle and can therefore be copied from the provided examples.

On constructing the main program you have to decide, however, if you want to develop a local application or a server in a distributed environment. For a local application you have to provide for a function called COBOLMAIN, for a distributed application you have to provide for two functions called COBOLAPPINIT and COBOLAPPFINISH.

These main programs have to use in their WORKING STORAGE section the copy path "IDMcobws.cob" supplied by the Dialog Manager in order to be able to access the DM definitions.

3.2.3.1 Local Applications

The structure of the function COBOLMAIN here looks as follows:

- First the function DMcob_Initialize has to be called to initialize the Dialog Manager. This has to be the first function to be called in the Dialog Manager. All other functions lead to errors.
- After having initialized the Dialog Manager, the dialog belonging to the application can be loaded by means of the function DMcob_LoadDialog.
- If the dialog has been loaded successfully, the functions contained in the dialog have to be transferred from the COBOL program to the Dialog Manager. For functions with parameters this is done by calling the C function CobRecMInit<name of the dialog or module> generated by the simulation program; for functions which have no records as parameters the function BindFuncs is called. The function BindFuncs can be generated via the program gencobfx. For the MICRO FOCUS COBOL compiler on UNIX systems the functions can also be drawn and called dynamically by the runtime system. To use the same method also for DM calls to the application, the function **DMufcob_BindThruLoader** respectively **DMmfviscob_BindThruLoader** for MICRO FOCUS VISUAL COBOL has to be called. In doing so, all COBOL functions provided by function pointers are called dynamically by the COBOL runtime system.
- After binding the functions to the dialog, application-specific initializations have to be executed.
- After loading the dialog and binding the functions to the dialog, the dialog has to be started for the user by means of the function DMcob_StartDialog. On calling this function the start rule "on dialog start" in the dialog is executed, and all windows defined as visible in the dialog are made visible.
- After that the control is transferred to the Dialog Manager by calling the function DMcob_EventLoop. This function normally returns only at the program end so that instructions which are made after having called this function, will only be executed at the end of the application.

Example

```
identification division.
programid. CobolMAIN.

data division.
workingstorage section.
*This is the copy path which provides for the possibility
*that the values defined by the Dialog Manager can be
*accessed.
copy "IDMcobws.cob".
```

```
*Definition of a variable to save the dialog ID.
77    Dmdialogid    pic 9(9) binary value 0.
*Variable for intermediate data storage of the current
*function
*By this variable the error output will be facilitated.
77    Funcname        pic X(30) value spaces.

linkage section.
*Definition of the parameters which are passed on to
*the main program.
01    exitstatus pic 9(4) binary.

procedure division using exitstatus.
startup section.

*Initialization of the Dialog Manager
initializeIDM.
*Here the separator can be set
*by which the strings in COBOL can be finished.
  move "@" to DMsetsep.
  call "DMcob_Initialize" using DMStdArgs
        DMCommonData.
  perform errorcheck.

*Loading the dialog belonging to the application
loaddialog.
  call "DMcob_LoadDialog" using DMStdArgs DMdialogid
         by content "table.dlg@".
  perform errorcheck.

*Binding the functions by means of records as parameters.
bindrecords.
  call "CobRecMInitCobolBeispiel" using DMStdArgs
             DMdialogid
          DMStdArgs.
*Binding the functions without records as parameters.
  call "BindFuncs" using DMdialogID DMStdArgs.

*Starting the dialog
startdialog.
  call "DMcob_StartDialog" using DMStdArgs DMdialogid.
  perform errorcheck.

*Starting the processing in the dialog
eventloop.
```

```
      call "DMcob_EventLoop" using DMStdArgs.
      perform errorcheck.

    *Finishing the application
    dialogdone.
      display "Application finishes successfully.".
      goback.

    *Query for errors
    errorcheck.
      if DMStatus
        display "Error occurred in " funcname upon sysout
        display "Application terminates due to error."
        move 1 to exitstatus
        goback.
```

3.2.3.2 Distributed Applications

For distributed applications two COBOL functions have to be supplied, which can initialize or finish the application. The initialization function is called CobolAppInit, the end function CobolAppFinish.

The structure of the function CobolAppInit looks as follows:

- First the function DMcob_Initialize has to be called to initialize the Dialog Manager. This is a big difference compared to the distributed applications realized in C, for DM_initialize must not be called there. By calling DMcob_Initialize important adjustments are made in the COBOL interface so that COBOL server applications cannot do without this call.
- After initializing the COBOL interface the functions in the dialog have to be transferred from the COBOL program to the Dialog Manager. The functions contained in the dialog have to be transferred from the COBOL program to the Dialog Manager. For functions with parameters this is done by calling the C function CobRecMInit<name of the dialog or module> generated by the simulation program; for functions which have no records as parameters the function BindFuncs is called. This function BindFuncs can be generated via the program gencobfx. For the MICRO FOCUS COBOL compiler on UNIX systems the functions can also be drawn and called dynamically by the runtime system. To use the same method also for DM calls to the application, the function **DMufcob_BindThruLoader** respectively **DMmfviscob_BindThruLoader** for MICRO FOCUS VISUAL COBOL has to be called. In doing so, all COBOL functions provided by function pointers are called dynamically by the COBOL runtime system. In contrast to the nondistributed applications, here the application ID to which the functions belong has to be passed on as a parameter.
- After binding the functions the application-specific initializations should be executed.

In the function CobolAppInit those actions have to be carried out which bring about a controlled termination of the application. Calls to the Dialog Manager are not necessary here, the server application will automatically be terminated after the function return.

Example

```
identification division.
programid. COBOLAPPINIT.

data division.
workingstorage section.
* Using the copy path supplied by the Dialog Manager
copy "IDMcobws.cob".

linkage section.
* Declaration of the function parameters
* In the Exit state the result is returned,
* in DMapplid the application ID and in the
* DMdialogID the dialog ID is passed on.
01    Exitstatus    pic 9(4) binary.
77    Dmapplid    pic 9(9) binary.
77    Dmdialogid    pic 9(9) binary.

procedure division using exitstatus dmapplid
    DMdialogid.
startup section.

* Initialization of the COBOL interface
initialize-IDM.
  call "DMcob_Initialize" using DM-StdArgs
    DM-Common-Data.
  PERFORM ERROR-CHECK.

* Dynamic binding of the COBOL functions
* after the record functions have been bound.
BIND-FUNCTIONS.
  call "CobRecMInitCobolBeispiel" using DM-StdArgs
              DM-appl-id
              by content 0.
  call "DMufcob_BindThruLoader" using DM-StdArgs
              DM-appl-id.

* Setting the return value.
  MOVE DM-SUCCESS TO EXIT-STATUS.
  GOBACK.
```

3.2.4 Auxiliary Means for the COBOL Programming

To avoid having to input twice the definitions in the dialog for parameters of the COBOL functions, the simulator of the Dialog Manager can generate a necessary copy path from a dialog file. This is done by the start option

```
+writetrampolin <Basis-Name>
```

In doing so, the basis name of a file is indicated, to which the endings ".c", ".cob" and ".cpy" is attached to generate the files needed for the call of the COBOL functions.

For the example the command line looks as follows:

```
idm +writetrampolin tablec table.dlg
```

By this call the following files are created:

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

The generated C and COBOL files have to be translated and linked to the application. The CPY file should be used to be able to access the definitions of the transfer structure in the dialog.

This file looks as follows in the example:

```
01 RecAddress.
   05 NName      pic X(25).
   05 FirstName  pic X(15).
   05 City       pic X(25).
   05 Street     pic X(30).
   05 Country    pic 9(9) binary.
```

3.2.5 Functions for the Tablefield

Functions which can fill or query the contents of objects with list characters are an exception with regard to the layer model, for these functions can only be written with extensive knowledge of the dialog. This is necessary so that the functions can operate effectively. The tablefield has to be filled by means of so-called temporary areas. These temporary areas in the Dialog Manager are created, then filled before the application and finally allocated to an object. In doing so, the permanent flicker of the tablefield during filling can be avoided in local applications; for distributed applications a strong increase in performance compared to the single allocation can be achieved. The procedure consists of the following steps:

- First the Dialog Manager has to be informed that a temporary area has to be created. This can be done by means of the function DMcob_InitVector. If possible, the desired size of the area should be indicated here, i.e. how many elements this area shall have. This size, however, is just an auxiliary value for the Dialog Manager, for allocations on higher elements the area will be enlarged correspondingly. In addition, the datatype has to be indicated here, which shall be saved in the area to be created. To do so, the data types defined by the Dialog Manager e.g. *DT-string*, *DT-integer* or *DT-boolean* can be used. The data type *DT-void* here means in particular that an area shall be created which can adopt the attributes *AT-content*, *AT-userdata*, *AT-active* and *AT-sensitive*. This area can be allocated to a tablefield or to a listbox.

- This function returns an ID of the created temporary area as a result. This ID has to be transferred to all subsequent functions, if these want to access this area.
- After creating a temporary area, this area can be filled by means of the function DMcob_SetVectorValue. To do so, the field DM-Index in the DM-Value structure must contain the number of the element which shall be set at that moment. In the DM-Value parts of the structure the actual value will be transferred. In the DM-Datatype the data type of the value has to be indicated. If the area has been created by the data type *DT-void*, an attribute which shall be filled must be created additionally in DM-Attributes.
- After filling the area it can be allocated to an object by means of the function DMcob_SetVector. Here you can control via parameters whether the new contents shall be appended to the object or whether the old contents shall be overwritten.
- If the temporary area is not needed any more, the area has to be freed absolutely by the function DMcob_FreeVector.

The values are read out analogous to filling the object:

- fetching the values from the object by the function DMcob_GetVector
- reading out the single values by the function DMcob_GetVectorValue
- freeing the temporary area by the function DMcob_FreeVector.

3.2.5.1 Function FILLTAB

This function takes on the filling of the tablefield. The tablefield ID and the number of elements to be filled are transferred to this function.

Usually the definition of the module is made first (it should have the same name as the function in the 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 "First Name ".
77  DLG-WOHNORT PIC X(80) value "Place of Residence ".
```

In the linkage section a copy is made on the file supplied by the DM IDMcobls.cob, so that the definitions in the COBOL program can be accessed.

```
LINKAGE SECTION.
*By this copy path the definitions in the
*Dialog Manager
*are available in this COBOL module.
COPY "IDMcobls.cob".
*In this parameter the number of rows is transferred
*which are to be initially filled.
77  DLG-CNT PIC 9(9) binary.
*In this variable the table is transferred
*which shall be filled.
77  DLG-OBJECT PIC 9(9) binary.
```

On defining the procedure division it shall be considered that this division has one parameter more than defined in the dialog. The first parameter of a COBOL function called by the DM is always the DM-Common-Data and these will not be defined in the dialog.

```
*This COBOL function creates a temporary memory area in
*the Dialog Manager and allocates this area to a
*tablefield.

PROCEDURE DIVISION USING DM-COMMON-DATA
    DLG-OBJECT DLG-CNT.
ORGANIZE-IN SECTION.
  MOVE 0 TO BASE.
  MOVE DLG-CNT TO ICOUNT.
*Initialization of a memory area in the DM
```

In this function a temporary area will be created, filled and then allocated to a tablefield.

```
  CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
    DT-String
    ICOUNT.
*Initialization of the DM-Value structure
  MOVE DT-STRING TO DM-DATATYPE.
*Filling the temporary area which shall then
*be displayed in the table.
```

```
PREPARE-DATA SECTION.
*Setting the individual contents
  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
*First the name is set.
        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

*After that the first name is set.
        ADD 1 TO KL
    MOVE KL TO DM-INDEX
    MOVE DLG-VORNAME TO STR-TAB
*Editing of a modified string for the display
    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

*Finally the city is set.
    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
*Computing a modified string for the display
    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 of loop to compute the data.
  END-PERFORM.

*Transferring the saved values to the display
  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.
*The entire table contents shall be replaced by the
*contents of the temporary area.
*Therefore the last three parameters have the value 0.
  CALL "DMcob_SetVector" USING DM-StdArgs DM-Value
        DM-POINTER
        by content 0
        by content 0
        by content 0.
```

Finally this area will be released in the Dialog Manager.

```
    *Freeing the memory area

      CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
      GOBACK.
```

3.2.5.2 Function TABFUNC

By means of this function the reloading of the tablefield will be achieved. Whenever the user scrolls in an area which has not been filled yet, the Dialog Manger calls this function.

The parameters are uniquely defined and therefore cannot be changed. These reloading functions get as a parameter a structure DM-Content-Data in which the necessary information is saved.

```
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 the DM-Co-Object the ID of the tablefield is transferred. The fields DM-Co-Visfirst and DM-Co-Vislast contain the first and last visible line, the fields DM-Co-Loadfirst and DM-Co-Loadlast contain the first and last line to be loaded. But these are only minimal values, the application may at any time load more than the indicated lines.

```
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.
```

In order to access the DM-Content-Data structure, the file IDMcoboc.cob per COPY must be drawn. Otherwise this function works the same way as the function FILLTAB.

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

*This COBOL function creates a temporary memory area
*in the Dialog Manager and allocates this area to a     *tablefield.

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.
*Initialization of a memory area in the DM
  CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
    DT-String
    ICOUNT.
*Initialization of the DM-Value structure
  MOVE DT-STRING TO DM-DATATYPE.
  PERFORM FILLDATA.

*Transferring the saved values to the display
  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.

*Freeing the memory area
  CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
  GOBACK.
```

```
FILLDATA.
*Setting the individual contents
  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
*Computing a modified string for the display
    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
*Computing a modified string for the display
    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
*Computing a modified string for the display
    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 Functions with Records as Parameters

In contrast to the functions for the tablefield, the subsequent functions can do without the information about the dialog, which is even much better. These functions are therefore written in standard COBOL without any call to the Dialog Manager. Only the copy paths show that these are functions which are called by the Dialog Manager.

3.2.6.1 Function GETADDR

This function shall bring the rest of the record to the surface on a given name. In order to access the dialog definition, a copy will be added to the file tablec.cpy generated by the Dialog Manager.

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

    COPY "IDMcobls.cob".
    *The following copy belongs to this application. It was
    *generated by the DM from the dialgo and should be used
    *in the COBOL modules where the corresponding structures
    *shall be accessed.
    COPY "tablec.cpy".

    PROCEDURE DIVISION USING DM-COMMON-DATA RECADDRESS.

    SEC-RECADDRESS SECTION.
    *Normally the values for the individual entries should be
    *fetched from the database. This is not necessary here.
```

```
*Instead dummy values are allocated to the corresponding
    *structure elements.
        MOVE "NAME" TO NNAME.
        MOVE "FIRST NAME" TO FIRSTNAME.
        MOVE "STREET" TO STREET.
        MOVE 3 TO COUNTRY.
        GOBACK.
```

3.2.6.2 Function PUTADDR

This function shall save the data changed by the user. In order to access the dialog definitions, a copy is added to the file tablec.cpy generated by the Dialog Manager.

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

    DATA DIVISION.

    WORKING-STORAGE SECTION.

    LINKAGE SECTION.

    COPY "IDMcobls.cob".
    *The following copy belongs to this application. It was
    *generated by the DM from the dialog and should be used
    *in the COBOL modules where the corresponding structures
    *can be accessed.
    COPY "tablec.cpy".

    PROCEDURE DIVISION USING DM-COMMON-DATA RECADDRESS.

    SEC-RECADDRESS SECTION.

    *Normally the values for the individual entries should be
    *fetched from the database. This is not necessary here.
    *Instead dummy values are allocated to the corresponding
    *structure elements.
        MOVE "      " TO STREET.
        MOVE 0 TO COUNTRY.
        GOBACK.
```

3.3 Compiling and Linking

Finally the created sources have to be translated. The individual compiler options here depend on the environment and can only be indicated as examples. The makefiles in the example should be used and adapted to the local conditions.

In principle the following steps are needed to construct an executable application:

- By the simulation program of the Dialog Manager the necessary copy paths are generated.

  ```
  idm  +writetrampolin  tablec  table.dlg
  ```

- The generated C file is translated by means of the C compiler and the usual compiler options for the Dialog Manager. In addition it has to be indicated in one of the options for which COBOL system the file is to be translated. For doing so, there are the possibilities
  -DUFCOB for MICRO FOCUS COBOL.
  Moreover the option **-DUFCOB_LINK** has to be set at Microsoft Windows. For the UNIX systems this switch can be used if an executable program which can run without the COBOL runtime system is to be linked together.

  ```
  cc -c -DMOTIF -DHP9800 -DHPUX -DUFCOB table_.cob
  ```

- The COBOL module has to be translated in the usual manner by means of the COBOL compiler.

  ```
  cob -x -c FILLTAB.cbl
  ```

- Afterward the translated files have to be linked together to form an executable. On the UNIX-based systems this can be done by means of the shell script **ufdmlink** for MICRO FOCUS programs. On the other systems the linker is called directly.

In contrast to the pure C applications the file "startup.obj" must not be bound to the program. Instead, however, the files and libraries in charge of COBOL have to be linked to the program. During linking the indicated order has absolutely to be adhered to.

4 Data Types

To realize the COBOL functions defined in the dialog, first a mapping rule is needed to define the way the data types defined in the dialog can be mapped onto the data types available in COBOL. Then follows an explanation of the structures which have been defined for the communication between the Dialog Manager and the application and which become available by copying the file **IDMcobws.cob**. They serve for the transfer of the values to the DM as well as for the return of the DM values to the application.

4.1 Mapping the Dialog Data Types on COBOL Data Types

4.1.1 Basic Data Types

The basic data types used in the dialog are mapped in COBOL as follows:

DM Data Type	COBOL Data Type
boolean	pic 9(4) binary
cardinal	pic 9(4) binary
enum	pic 9(4) binary
datatype	pic 9(4) binary
index	no equivalent
object	pic 9(9) binary
string [?]	pic X(?) COBOL Interface for 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	no equivalent COBOL Interface for MICRO FOCUS VISUAL COBOL: pointer

4.1.2 Records

Besides these basic data types even the records are converted to COBOL data types. The conversion is done according to the pattern above for each element individually.

The definitions necessary for the COBOL program are made available by a generated COBOL copy file. This generation is done by means of the simulation program and the start option **+/-writetrampolin**.

```
idm +writetrampolin <base-name> <dialog-file-name>
```

Example

A record shall contain a string and a number.

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

The accordingly generated COBOL definition then looks as follows:

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

Note

In order to support the data type *National Character* with MICRO FOCUS VISUAL COBOL, the option **-mfviscob-u** must be specified in addition. The COBOL definition then looks like this:

```
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 Collection Data Types

The colection data types of the IDM and the related functions for (managed) IDM values (**Managed DM-Values**) are supported through the pointer data type (POINTER) of MICRO FOCUS VISUAL COBOL.

DM Data Type	Visual COBOL Data Type
hash	POINTER
list	POINTER
matrix	POINTER
refvec	POINTER
vector	POINTER

A **Managed DM-Value** is passed as pointer to MICRO FOCUS VISUAL COBOL:

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

To use such a value within the **DM-Value** structure, the value is copied to *DM-value-pointer* and *DM-datatype* is set to *DT-anyvalue*.

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

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

4.2 Standard Argument

The Dialog Manager standard argument **DM-StdArgs** is transferred to all functions. It is used to check the return value of the called function and to set global information.

```
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 the COBOL Interface for 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.
```

Description of the Structure Elements

DM-status

This contains the error state of the call to the Dialog Manager function. Possible values are:

- *DM-success*
  The function found no error.
- *DM-error*
  The function detected an error.

All Dialog Manager functions set this field to tell whether the call was successful or not.

DM-options

In this option field, function specific information can be set. The Dialog Manager function always sets this field at zero. The possible values are described by the functions, because the valid values are function specific.

DM-rescode

This field contains the return code of some Dialog Manager functions.

The next three elements of the structure belong together with regard to their significance and are due to the implementation of two different programming languages. In the application the programming is done by COBOL, which defines strings when specifying the length; the Dialog Manager, however, considers strings dynamical, i.e. it adjusts the length always to the actual conditions. This is why the length has always to be specified, if this string is to be transferred to a COBOL program. This interaction will be explained in detail in the next chapter. In principle these three elements should be set only before initializing the Dialog Manager and then not changed any more afterward; otherwise difficulties may occur in the various COBOL programs.

DM-setsep
DM-setsep-u

The character stored in this field tells the Dialog Manager that your application terminates the strings with this character.

DM-getsep
DM-setsep-u

The character stored in this field tells the Dialog Manager to terminate strings with this character.

DM-truncspaces

If this field is set, the Dialog Manager removes all blanks in character strings.

DM-usercodepage
DM-usercodepage-u

In this field the code page for strings is defined.

Note

Before IDM version A.06.01.d the structure **DM-StdArgs** had this definition:

```
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 Handling of String Parameters

When handling string parameters in the COBOL interface you have to note that the DM is internally written in C! Unfortunately, these two programming languages have a very different internal representation of strings.

	C	COBOL
Length	dynamic, any length	statically defined length
Terminator	'\0'	none
Fill grade	length	length is different to
	usually corresponds to the string	definition - usually filled up with blanks

To bring these two different representations of strings together, there are the following procedures in DM when handling string parameters:

1. On the initialization call to the DM with **DMcob_Initialize**, the DM can be given three values that navigate the string handling:
   a. `DM-setsep of DM-StdArgs`
      The included character is the character with the help of which the COBOL program terminates its real strings. The DM searches for these characters and deletes all following characters in the transferred string.
   b. `DM-getsep of DM-StdArgs`
      The included character is the character with the help of which the DM fills up the strings up to their definition length before they are transferred to the COBOL program.
   c. `DM-truncspaces of DM-StdArgs`
      With this parameter you can tell the DM that all blanks at the end of a string can be regarded as non-existent and thus be deleted.

   These settings with **DMcob_Initialize** are valid for all functions directly called by the DM and their return values respectively.
2. These values can be re-defined locally and temporarily at all calls to the DM-interface functions. As described under 1, there are three possibilities:
   a. `DM-setsep of DM-StdArgs`
   b. `DM-getsep of DM-StdArgs`
   c. `DM-truncspaces of DM-StdArgs`

   Meaning see above, 1.

The following occupation is useful:

```
DM-SetSep @
DM-GetSep " "
DM-truncspaces 1
```

With this specification, the respective values can be directly processed by the COBOL program and in the DM.

Example

In the dialog an input field is defined with a length which is limited to a length of *20* by the attribute *.maxchars*. The COBOL function which is to check this input, is assigned a string parameter of length *20*.

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

edittext EtEingabe
{
  .maxchars 20;
}
```

The following table shows the influence of the single elements on the transfer of strings before and after COBOL.

Dialog	**COBOL**	
	DM-GetSep Space	**Result**
AB in the input field	AB is filled up with 18 blank spaces	AB + 18 blank spaces

In the COBOL program this string will then be processed and a new content will be set in the string.

COBOL	**Dialog**		
Command	**Contents of String**	**DM-TruncSpaces 1**	**Result**
`MOVE "NEW" TO InString`	NEW + 17 blank spaces	NEW	NEW

4.4 Using the *anyvalue* Data Type

If the type *anyvalue* is defined as a parameter of a COBOL function in the IDM dialog script, then any data type is permitted for this parameter. In the COBOL program, such a parameter is passed to the program as a *pointer*. To use a value of this kind in interface functions that expect a **DM-Value** structure, the parameter can be assigned to *DM-value-pointer* and *DM-datatype* can be set to *DT-anyvalue*. This can be understood in a way that the **DM-Value** structure may contain any data type and that the actual value is stored as a pointer.

Regardless of how an input parameter had been defined, a return value or output parameter whose data type can be stored directly in the **DM-Value** structure will actually be stored directly. For instance, *DT-anyvalue* would become *DT-string*.

Collection data types cannot be stored directly, so they are always returned as *DT-anyvalue*. Such a *DT-anyvalue* value may be processed with the **DMcob_Value*** functions.

Please Note

The data type *anyvalue* indicates in the IDM dialog script that any data type is allowed. At runtime, however, an attribute, a variable, a parameter, or a return value always has a value with a specific data type that does not equal *anyvalue*.

Example

The following definition of a rule indicates that the return value may be of any data type:

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

However, at runtime, not *anyvalue* will be returned, but always the type of the `return` statement, for example `print typeof(RuleAny(1));` will output the value *"string"*.

An exception to the rules above is a Managed Value created with **DMcob_ValueInit**. A Managed Value is always preserved. If a Managed Value is used as *DM-value-pointer* in a **DM-Value** structure (*DM-datatype* must be set to *DT-anyvalue*), then the data type *DT-anyvalue* is retained. This also applies to return values and output parameters.

Life Time of DM-value-pointer Values

A Managed Value that has not been created globally is only valid until the end of the function in which it was created. It can still be returned to the IDM however.

The same applies to pointer values passed as *anyvalue* parameters to a function. They only remain valid until the end of the function as well. Returning them to the IDM is allowed.

All other pointer values, which for example were created to return a collection data type, are only valid until the next time a **DMcob*** function is invoked. They also must not be returned to the IDM. To enable these values to be processed, the **DMcob_Value*** functions do not destroy their validity.

It is recommended to use Managed Values for collection data types right from the start.

4.5 Inquiring and Changing Attributes

To set and inquire attributes in the Dialog Manager two different data structures are used. In the **DM-Value** structure exactly one value can be inquired or set. In the **DM-ValueArray** structure, however, up to *16* values can be inquired or set by one call. The **DM-ValueIndex** structure is used to pass an additional index value.

These structures are defined in **IDMcobls.cob** and **IDMcobws.cob**.

4.5.1 DM-Value

By means of this structure exactly one value can be set or inquired in the Dialog Manager. This structure has the following definition:

```
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 the COBOL Interface for 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.
```

Description of the Structure Elements

DM-object

This is the object whose attribute is to be changed or returned.

DM-attribute

This is the attribute which is to be set or returned.

DM-indexcount

This is the number of indexes which should be used by Dialog Manager when accessing or setting an attribute.

DM-index

This is the first index of the attribute which should be set or returned.

This number is only valid or used if the attribute is an indexed attribute or if *DM-indexcount* is set at a value greater than *0*.

DM-second

This is the second index of the attribute which should be set or returned. This number is only valid or used if *DM-indexcount* is set at a value greater than *1*.

DM-long-first

This element represents the first index. It is only used if *DM-indexcount* is greater than *1* and a function is called that processes the long index. These functions can be recognized by the name "DMcob_L*". This allows index values in the range from *0* to *65535* to be passed to the Dialog Manager.

DM-long-second

This element represents the second index. It is only used if *DM-indexcount* is greater than *1* and a function is called that processes the long index. These functions can be recognized by the name "DMcob_L*".

DM-datatype

This describes which part of the structure is filled and should be used to get the value.

DM-value-integer

This is used to set or get an integer attribute.

DM-value-object

This is used to set or get an attribute which is a Dialog Manager object like color, font, or object.

DM-value-boolean

This is used to set or get boolean attributes.

DM-value-cardinal

This is used to pass the enumeration and cardinal values.

DM-value-classid

This is used to get the class to which the object belongs.

DM-value-first, DM-value-second

These two elements are always used together. They pass an index value on to the DM and return respectively an index value from the DM to the COBOL function.

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

These two elements are always used together. They contain a long index value with the value range *0 … 65535* that is passed to the Dialog Manager or returned by the Dialog Manager to the COBOL function.

DM-value-attribute

This element is allocated if the identifier of a user-defined attribute is to be queried.

DM-value-method

This element is used for the handling of methods implemented in DM.

DM-value-pointer

This element is used to pass collections (value references) and data of the data type *DT-anyvalue*.

It is only used by the COBOL Interface for MICRO FOCUS VISUAL COBOL.

DM-value-string-putlen

This can be set to show the Dialog Manager the end of a string if the string is to be terminated by a value which differs from the definitions. The number of characters is to be specified.

DM-value-string-getlen

This contains the length of the returned string from the Dialog Manager. The number of characters is specified.

DM-value-string-size

Internal field, not to be changed directly! The number of characters is specified.

DM-value-string

This is used to set or get string attributes. These strings must not be longer than *80* characters.

DM-value-string-u

In this element, strings can be passed as *National Character* (*PIC N*) when working with Unicode strings (UTF-16) (MICRO FOCUS VISUAL COBOL only).

Note

Before IDM version A.06.01.d the structure **DM-Value** had this definition:

```
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

This structure allows for a full-fledged index argument. It is used by the **DMcob_Value*** functions of the COBOL Interface for MICRO FOCUS VISUAL COBOL to process the **collection data types**. However, the structure is available for all supported COBOL variants and is not restricted to these application areas. It is identical to the DM-Value structure.

4.5.3 DM-ValueArray

For the call of some Dialog Manager functions an array of up to *16* structure elements is needed. To do so, the following structure is available:

```
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 the COBOL Interface for 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.
```

Description of the Structure Elements

DM-value-size

This is the total size of the array. It is set at *16* by the Dialog Manager as a standard. This value may be changed by the user to allow to set more then sixteen values by one function call.

DM-value-count

This is the number of currently used values in the array. The value has to be set and has to be less than or equal to the *DM-value-size*.

DM-va-object

This is the object whose attributes are to be changed or returned.

DM-va-attribute

This is the attribute which should be set or returned.

DM-va-indexcount

This is the number of indexes which are to be used by accessing or setting the attribute.

DM-va-index

This is the first index of the attribute which should be set or returned. This number is only valid if *DM-va-indexcount* is specified by a value greater than *0*.

DM-va-second

This is the second index of the attribute which should be set or returned. This number is only valid if *DM-va-indexcount* is specified by a value greater than *1*.

DM-va-datatype

This describes which part of the structure is filled and should be used to get the value.

DM-va-value-integer

This is used to set or get an integer attribute.

DM-va-value-object

This is used to set or get an attribute which is a Dialog Manager object like color, font or object.

DM-va-value-boolean

This is used to set or get boolean attributes.

DM-va-value-cardinal

This is used to pass the enumeration and cardinal values.

DM-va-value-classid

This is used to get the class to which the object belongs.

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

These two elements are always used together. They pass an index value on to the DM and return respectively an index value from the DM to the COBOL function.

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

These two elements are always used together. They contain a long index value with the value range *0 … 65535* that is passed to the Dialog Manager or returned by the Dialog Manager to the COBOL function.

DM-va-value-attribute

This element is allocated if the identifier of a user-defined attribute is to be queried.

DM-va-value-method

This element is used for the handling of methods implemented in DM.

DM-va-value-pointer

This element is used to pass collections (value references) and data of the data type *DT-anyvalue*.

It is only used by the COBOL Interface for MICRO FOCUS VISUAL COBOL.

DM-va-value-string-putlen

This can be set to show the Dialog Manager the end of a string, if the string to be terminated differs from the definitions.

DM-va-value-string-getlen

This contains the length of the returned string from the Dialog Manager.

DM-va-value-string-size

Internal field, not to be changed directly!

DM-va-value-string

This is used to set or get string attributes. These strings must not be longer than *80* characters.

DM-va-value-string-u

In this element, strings can be passed as *National Character* (*PIC N*) when working with Unicode strings (UTF-16) (MICRO FOCUS VISUAL COBOL only).

Note

Before IDM version A.06.01.d the structure **DM-ValueArray** had this definition:

```
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 Data Types in the Structures DM-Value and DM-ValueArray

If values are inquired from the DM with **DM-Value** or **DM-ValueArray**, the DM sets the structure element *DM-datatype* or *DM-va-datatype* according to the inquired attribute.

The following definitions are possible.

Identifier (ID)	COBOL Data Type	Meaning/Structure Element
DT- accel	pic 9 (4) binary	Accelerator reference. DM value-object / DM-va-value-object.
DT-anyvalue	pointer	Data of any data type. DM-value-pointer / DM-va-value-pointer
DT-application	pic 9 (4) binary	Application reference. DM-value-object / DM-va-value-object.
DT-attribute	pic 9 (4) binary	Data type for an attribute in the Dialog Manager.
DT-boolean	pic 9 (4) binary	Boolean value. DM-value-boolean / DM-va-value-boolean.
DT-class	pic XX	Describes the classes of the various objects. DM-value-classid / DM-va-value-classid.
DT-color	pic 9 (4) binary	Color reference. DM-value-object / DM-va-value-object.
DT-cursor	pic 9 (4) binary	Cursor reference. DM-value-object / DM-va-value-object.
DT-enum	pic 9 (4) binary	Enumeration type in the Dialog Manager. DM-value-cardinal / DM-va-value-cardinal.
DT-event	pic 9 (4) binary	Defines the type of an event. DM-value-cardinal / DM-va-value-cardinal.
DT-font	pic 9 (4) binary	Font reference. DM-value-object / DM-va-value-object.
DT-hash	pointer	Hash reference. DM-value-pointer / DM-va-value-pointer
DT-index	pic 9 (4) binary	Describes a two-dimensional index. Thus, always two elements have to be evaluated in the value structure. DM-value-first & DM-value-second / DM-va-value-first & DM-va-value-second.

Identifier (ID)	COBOL Data Type	Meaning/Structure Element
DT-instance	pic 9 (4) binary	Instance reference. DM-value-object / DM-va-value-object.
DT-integer	pic 9 (4) binary	Number value. DM-value-integer / DM-va-value-integer.
DT-list	pointer	List reference. DM-value-pointer / DM-va-value-pointer
DT-matrix	pointer	Matrix reference. DM-value-pointer / DM-va-value-pointer
DT-object	pic 9 (4) binary	Object reference. DM-value-object / DM-va-value-object.
DT-refvec	pointer	Refvec reference. DM-value-pointer / DM-va-value-pointer
DT-rule	pic 9 (4) binary	Rule reference. DM-value-object / DM-va-value-object.
DT-scope	pic 9 (4) binary	Defines whether we see a default, a model or a normal object. DM-value-cardinal / DM-va-value-cardinal.
DT-string	pic X (??)	Transferring a string. DM-value-string / DM-va-value-string.
DT-text	pic 9 (4) binary	Text reference. DM-value-object / DM-va-value-object.
DT-tile	pic 9 (4) binary	Tile reference. DM-value-object / DM-va-value-object.
DT-timer	pic 9 (4) binary	Timer reference. DM-value-object / DM-va-value-object.
DT-undefined	none	Data Type is undefined.
DT-var	pic 9 (4) binary	Variable reference. DM-value-object / DM-va-value-object.
DT-vector	pointer	Vector reference. DM-value-pointer / DM-va-value-pointer
DT-void	none	Data Type is unknown or unspecified.

When setting attributes, there are several possible data types. Therefore, the element *DM-datatype* has to be set according to the used data type, so that the DM can read the value out of the corresponding element.

The possible values for *DT-scope* are:

1	default object
2	model
3	instance

4.6 Object Callback

All COBOL callback functions are called with **two** parameters: the first is the common-data structure which was transferred by the application to the Dialog Manager on calling DMcob_Initialize; the second parameter is a structure which is defined by the DM. In the second structure, the *DM-ObjectCallback-Data*, the data belonging to this callback are transferred to the application.

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

Through the *DM-ocb-object* field, the object linked to this function is transferred.

If the object has more than one item (e.g. selection in a listbox or a poptext), the index of the selected item is contained in the *DM-ocb-index* field.

In *DM-ocb-evbits*, the event mask of the triggering event is passed on. Since exactly one bit is set to each event in this mask, these events can also be used in combination. Below are the following possibilities as defined in **IDMcobws.cob**. The accel-field is set only when the callback for a key-event is called.

In the *DM-ocb-status* field, the application can tell the Dialog Manager to execute the rules belonging to this event or not to process the rules. If the value of this status field is *DM-success*, the Dialog Manager will execute all rules belonging to this event. If the application functions return "not *DM-success*", the Dialog Manager will not execute the rules belonging to this event.

Eventbit	Meaning
EM-activate	Object was activated
EM-charinput	Character input into an edittext
EM-close	Window was closed
EM-dbselect	Object was selected by a double click

Eventbit	Meaning
EM-deactivate	Object was deactivated
EM-deiconify	Icon was turned into a window
EM-deselect-enter	Edittext was exited with `Return` key (`Enter`)
EM-deselect	Object was deselected
EM-finish	Dialog processing was finished
EM-focus	Input focus was set to the object
EM-help	Help for the object was requested
EM-hscroll	Object was scrolled horizontally
EM-iconify	Window was iconified
EM-key	Character input
EM-modified	Edittext content was changed
EM-move	Object was moved
EM-resize	Object size was changed
EM-scroll	Object was scrolled
EM-select	Object was selected
EM-start	Dialog processing was started
EM-vscroll	Object was scrolled vertically

Note

The first parameter of an object callback in COBOL is always the **DM-COMMON-DATA**!

The COBOL object callback must be defined in the dialog script like this:

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

In COBOL this function then has to be defined as follows:

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

4.7 Reloading Functionality

All COBOL reloading functions are called with **two** parameters: the first is the general data structure which is passed on from the application to the Dialog Manager when calling DMcob_Initialize; the second parameter is a structure, the **DM-Content-Data** structure, which was defined by Dialog Manager. Reloading functions are always called by the DM when the user scrolls in a tablefield and parts become visible to which no content has been allocated yet.

```
01  DM-Content-Data
    02  DM-CO-OBJECT     pic 9(4) 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

Here, the object is transferred to which the function is bound , i.e. the identifier of the tablefield to be edited.

DM-CO-REASON

Here you can indicate the reason why the function has been called. The value CFR-LOAD is possible here. This value shall indicate that the content is to be reloaded in the tablefield.

The following parameters depend on how the tablefield was defined:

- If the tablefield is defined with .direction = 1, all following values are rows.
- If the tablefield is defined with .direction = 2, all following values are columns.

DM-CO-VISFIRST

Describes the first visible row/column.

DM-CO-VISLAST

Describes the last visible row/column.

DM-CO-FIRSTLOAD

Describes the first row/column to be loaded.

DM-CO-LASTLOAD

Describes the last row/column to be loaded.

DM-COUNT

Describes the total number of defined rows/columns.

DM-HEADER

Describes the number of rows/columns defined as headers.

The COBOL reloading function has to be defined as follows in the dialog script:

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

In COBOL as follows:

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

To set this structure, the file **IDMcoboc.cob** has to be copied in the source via the instruction

`COPY`

Example

```
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".

* reloading function
    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.

* Initialization of memory in DM
    CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER
    DT-STRING ICOUNT.

    PERFORM FILLDATA.

* Passing of the saved values to the display
    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.
```

```
* freeing of memory
    CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.

* deleting of contents in invisible area
    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 Data Function Structure DM-Datafunc-Data

This structure is always passed as a parameter to data functions.

The structure is shown simplified here. The complete definition can be found in the file **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.
*     Is identical to the structure DM-ValueArray.

   02 DM-df-index.
*     Is identical to the structure DM-Value.

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

Meaning of elements

DM-df-object

In this element, the object ID of the Data Model is passed to the data function.

DM-df-task

This element specifies the reason for the call. Currently the following values are possible: *MT-get*, *MT-set* and *MT-call*. For *MT-get*, the data function should return the necessary data for a Model attribute. *MT-set* forwards changes from the view to the data function. *MT-call* is used to implement data actions, i.e. calls that lead to a “manifold” manipulation of the data. It is reasonable to always signal changes to data via **DMcob_DataChanged**.

DM-df-attribute

For the call reasons *MT-get* and *MT-set*, this element contains the attribute on which the function shall be applied.

DM-df-method

For the call reason *MT-call*, this element contains the method that the data function shall execute. This is triggered by invoking a data action through the method *:calldata(<Method>,<Arg1>,...<Arg15>)*.

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

These elements contain the attribute identifier (from the *DM-df-attribute* element) or the method identifier (from the *DM-df-method* element) as a string to facilitate the implementation of user-defined attributes or to enable a module-independent comparison of attributes or methods.

In *DM-df-indentifier-u* identifiers may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

DM-df-args

In this element, the number of arguments (*DM-df-args-count*) and the argument list (*DM-df-valuerec*) are stored for the call reason *MT-call* (data action invoked via **:calldata()**). A return or change of arguments with this is not intended respectively will have no effect.

Note

The design of this structure is identical to **DM-ValueArray**.

DM-df-index

This element contains the index value for the attribute access that the call reason *MT-get* or *MT-set* refers to. If the *DM-df-index* element has the data type *DT-void*, it refers to the entire value. For indexed data values, the relationship kinds and also the processing of incomplete index values (e.g. *[0]*, *[?,0]* oder *[0,?]*) should be taken into account.

Note

The design of this structure is identical to **DM-Value**.

DM-df-data-pointer

For the call reason *MT-set*, this element contains the data value to be processed by the data function.

Note

The data is passed as value of type *DT-anyvalue*, since it will usually be collections. The value(s) can be retrieved with **DMcob_ValueGet** and the other **DMcob_Value*** functions.

DM-df-retval-pointer

For the call reason *MT-get*, in this element the data value for the specified attribute and the given index has to be returned.

Note

The return value is passed as value of type *DT-anyvalue*, since it will usually be collections. The return value can be changed and populated with **DMcob_ValueChange**.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

4.9 NULL Object

This is an object with a special identifier by means of which missing resource attributes and the resetting of resource attributes are realized.

It is called "NULL object" and has the Dialog Manager ID = 0.

It can be received from **DMcob_SetValue**, and can be used with **DMcob_GetValue,** or for function arguments.

The NULL object itself is type-independent, i.e. there is only one NULL object for all resources. The NULL object can be applied to all data types >= DT-instance. With GetValue, the NULL object real type depends on the attribute type.

Implications

- Interface functions can have a NULL object as a return value. The NULL object can also occur as a (input and output) parameter.
- If a NULL object is obtained as a return value of **DMcob_Get Value**, it will always have the most precise data type possible, e.g. *DT-text* instead of *DT instance*.
- If an attribute is to be set at zero by the application, the data type is to be given as precisely as possible, i.e. e.g. *DT-accelerator* instead of *DT-instance*. *DT-instance* can of course also be used. The Dialog Manager internally transforms it to the correct data type; but this method is more sensitive to programming errors.

4.10 Access to Attributes

The standard DM attributes are accessed according to the definitions included in the copy files **IDM-cobws.cob** and **IDMcobls.cob**. These attribute definitions can then simply be set in the structure **DM-Value** and be passed on to the DM.

Example

Setting the attribute "visible" in the structure **DM-Value**.

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

You can find a complete list of the defined standard attributes in the “Attribute Reference”.

User-defined attributes are defined by each dialog programmer and thus cannot be made globally available. If these user-defined attributes are to be accessed from the COBOL interface, their actual internal value has to be determined first. This actual internal value depends on the dialog, but because of the dialog life span it is static. After determining this actual internal value, you can work with it in the same way as with the attributes defined as static by the DM.

You can determine this value with the function **DMcob_GetValue**: to do so, *DT-string* has to be set in DM-Datatype, and *AT-label* has to be set in DM-Attribute. The object has to be the dialog to which the attribute belongs. The indexcount has to be 2, the index values 0. The name of the attribute to be determined has to be included in DM-value-string.

Necessary values:

DM-Object	Dialog
DM-Attribute	AT-label
DM-Datatype	DT-string
DM-Value-String	Name of the attribute to be determined
DM-Indexcount	2
DM-Index	0
DM-Second	0

Example

Inquiring a user-defined attribute of an object.

```
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 Return Values of the DM-functions DM-success / DM-error

Most DM-functions return in the standard argument (**DM-StdArgs**), if the function call was performed with or without error.

If the function was performed successfully, *DM-Status* is set at *DM-success*. If an error occurred when calling the function, *DM-Status* is set at *DM-error*.

The value *0* corresponds to *DM-success*, any other value corresponds to *DM-error*.

5 Linking the COBOL Program

5.1 Main Program

This chapter describes how the functions defined in DM are carried out in COBOL and are linked to the DM.

In principle, each COBOL program that is named "COBOLMAIN" in a non-distributed environment, is named "COBOLAPPINIT" in a distributed environment.

This main program is called by the DM and can carry out all necessary initialization steps.

5.1.1 Local COBOL Programs

In the local solution of the DM, the program "COBOLMAIN" is regarded as the actual program. Thus, it has to carry out the following steps:

- initialize DM (**DMcob_Initialize**)
- load the dialog (**DMcob_LoadDialog**)
- initialize the application
- transfer the function addresses (BindFuncs)
- start the dialog (**DMcob_StartDialog**)
- start the processing (**DMcob_EventLoop**)

After the successful execution of these steps, the user can work with the program.

5.1.2 Distributed COBOL Programs

In the distributed solution of the DM, the COBOL main program, "COBOLAPPINIT" has to carry out the following steps:

- initialize the COBOL interface of DM (**DMcob_Initialize**)
- transfer the function addresses (BindFuncs)
- initialize the application

5.2 Realization of COBOL Functions

During the realization of the actual COBOL functions, the parameter data types defined in the dialog have to be mapped onto the COBOL data types. The functions can be realized as "ENTRY" or as program.

When implementing the COBOL functions it has to be noted that the first argument of the function is always the **DM-COMMON-DATA**. By doing so, there will always be a parameter shift of 1 compared with the parameter definition in the dialog script.

Example

Dialog File

```
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.
```

If records are used on the function call, the generated COBOL data type of the record has to be defined as parameter type.

Example

Dialog File

```
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 Main Program

```
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 Subprograms

```
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 Object Callback Functions

Functions which are declared as object callback functions in the dialog description, have to be declared in COBOL as follows:

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

This function has to be defined in the dialog script as follows:

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

Note

In contrast to the C interface of the Dialog Manager, the COBOL callback function has two parameters.

5.4 Reloading Functions

Functions which are declared as reloading functions in the dialog description have to be declared in COBOL as follows:

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

This function has to be defined in the dialog script as follows:

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

Note

In contrast to the C interface of the Dialog Manager, the COBOL reloading function has two parameters.

5.5 Data Functions

Functions that are declared as data functions in the dialog description must be declared in COBOL as follows:

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

The function represents a Data Model (Model component) that provides the presentation objects (View component) with data values or stores and manages them.

The parameters of this function are predefined by the ISA Dialog Manager and cannot be changed.

This function is called when a synchronization between View and Model components is required. This may either happen automatically, according to the control options in the *.dataoptions[]*attribute of the involved components. The call can also be triggered by explicit invocation of the **:collect()**, **:propagate()**, **:apply()** and **:represent()** methods. For each Model attribute there is a separate call.

Remark

In contrast to the C Interface of the ISA Dialog Manager, the COBOL data function has two parameters.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Chapter “Data Function Structure DM-Datafunc-Data”

5.6 Canvas Functions

Functions which are declared as canvas functions in the dialog description have to be programmed in "C". COBOL support is not available for these functions.

5.7 Format Functions

Functions which are declared in the dialog description as format functions have to be programmed in "C". COBOL support is not available for these functions.

5.8 Transfer of Function Addresses

To be able to call the application functions the Dialog Manager has to know the addresses of the application functions. To get these addresses a structure named **DM_FuncMap** is defined. This structure cannot be used in COBOL because COBOL is not able to transfer addresses of functions to any other function. The transfer of the function addresses has to be written in C. On passing on the function addresses, two types of function have to be distinguished.

5.8.1 Functions without Records as Parameters

In this case, the transfer of the function address can follow the C source file that was generated by the program **gencobfx**: by calling the generated C functions from the COBOL main program.

Therefore the following steps are necessary:

1. definition of the functions in the dialog script
2. structuring the input file for generating the C-module
3. generating the C module with **gencobfx**

4. compiling the C module
5. calling the generated C functions from the COBOL main program.

5.8.1.1 Structure of the Function Definition File

The function definition file that is necessary to link COBOL functions to the DM can be generated automatically. The input file structure is system-independent. However, the way the C source file is to be maintained depends on the system.

The file has to contain all COBOL functions that are to be called directly by the DM. The first column has to contain the function name in the dialog file, the second column can contain the function name in the COBOL program. The script takes the first name to generate the COBOL function name if the second column for a function is missing.

5.8.1.2 Generating the Function Definition File

5.8.1.2.1 Unix

On UNIX, you can use a program named **gencobfx** to generate the function definition file. One of the arguments has to be a file that contains all names of your functions. This file can contain comments. The line which is to be a comment has to begin with “*”.

The syntax of the script is as follows:

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

-mf

By indicating this parameter a function file is generated which is suitable for the use with the MICRO FOCUS COBOL compiler.

-mfvis

By indicating this parameter a function file is generated which is suitable for the use with the MICRO FOCUS VISUAL COBOL compiler.

outfile

By indicating this parameter the function table can be stored in any file. Usually, the DM takes the name of the description file and replaces the characters after the point with "“c”".

func-entry

By indicating this parameter you can specify the function name that has to be called by your COBOL main program. This function links the functions to the DM by calling DM_BindCallBacks (see manual “C Interface - Functions”). DM usually takes the name “BindFuncs”.

descriptionfile

This file has to contain the names of all functions.

5.8.1.2.2 Microsoft Windows

On MICROSOFT WINDOWS, you can use a program named **gencobfx.exe** to generate the function definition file. One of the arguments has to be a file that contains all names of your functions. This file can contain comments. The line which is to be a comment has to begin with "*".

The syntax of the script is as follows:

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

-mf

By indicating this parameter a function file is generated which is suitable for the use with the MICRO FOCUS COBOL compiler.

-mfvis

By indicating this parameter a function file is generated which is suitable for the use with the MICRO FOCUS VISUAL COBOL compiler.

outfile

By indicating this parameter the function table can be stored in any file. Usually, the DM takes the name of the description file and replaces the characters after the point with “c”.

func-entry

By indicating this parameter you can specify the function name that has to be called by your COBOL main program. This function links the functions to the DM by calling DM_BindCallBacks (see manual “C Interface - Functions”). DM usually takes the name “BindFuncs”.

descriptionfile

This file has to contain the names of all functions.

5.8.1.3 Example

Function definition file

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

Generated C-file

```
#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 Functionality

If **BindThruLoader** is used then **gencobfx** is no longer needed and the call of **BindFuncs** can be omitted.

The functionality is enabled through:

- `CALL "DMufcob_BindThruLoader" USING DM-STDARGS DIALOG-ID.`
  for MICRO FOCUS COBOL on UNIX systems
- `CALL "DMmfviscob_BindThruLoader" USING DM-STDARGS DIALOG-ID.`
  for MICRO FOCUS VISUAL COBOL

Alternatively the following version may be used, which is more portable:

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

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

Although calling **DMcob_Control** is possible for all COBOL variants, there will still be some bindings where the dynamic connection does not work.

Note

If an ***application*** object is used in the dialog, then the functionality must be enabled at the ***application*** object and not at the dialog:

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

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

This APPL-ID has to be specified instead of the DIALOG-ID (see above).

Availability

DMF-CobolBindForLoader is available since IDM version A.06.01.d

5.8.2 Functions with Records as Parameters

In this case, the transfer of the function addresses follows automatically by calling the initialization function in the record module.

To do so, the following steps are necessary:

1. definition of the records in the dialog script
2. definition of the functions in the dialog script
3. generating the C-module by the simulator with the option **+writetrampolin**
4. compiling the C-module
5. calling the generated C-functions from the COBOL main program
6. linking of the program

5.8.2.1 Generating the Trampoline Module

To supply functions for dialogs with records by an application, the C-modules generated by the DM have to be translated and linked. The generation is necessary for every module which exists in the dialog and which contains definitions for functions. These modules are generated by calling the simulation with the option **+writetrampolin**:

```
idm +writetrampolin <outfile> <dialogfile>
```

The indicated dialog script may be a module.

This statement generates the necessary modules out of a dialog script for calling the functions. According to the kind of functions which use such records, the respective header files (C and/or COBOL) are generated.

When implementing these functions, the generated header file has to be included in the respective module with the following statement:

```
COPY
```

If such functions and records are used in an application, these records have to be initialized by calling the function

```
cobRecInit<ModuleName>
```

The initialization should usually be carried out before the function call BindFuncs!

If such structures are used in DM applications, i.e. if the distributed DM is used, the application for which the files are to be generated has to be indicated in addition.

The name of the initialization function is set up by the application name. Thus,

```
cobRecInit<ApplicationName>
```

has to be called.

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

To generate a copy file that also contains *National Character* (MICRO FOCUS VISUAL COBOL only), this has to be specified with the option **-mfviscob-u** .

5.8.2.2 Compiling the C Module

When compiling the generated C module, a flag has to be set according to the used COBOL.

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

On the PC platforms MICROSOFT WINDOWS the switch `-DUFCOB_LINK` has to be set additionally when using the MICRO FOCUS COBOL compiler.

5.8.2.3 Example

Dialog File

```
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;
}
```

Generated COBOL Copy File

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

Generated C Program

```
#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

#ifdef XLCOB
#        define WriteAddr writeaddr
#        define ReadAddr readaddr
#endif

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

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

DeclCobol(WriteAddr)

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

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

CallCobol1(;,WriteAddr,&record0);
}

DeclCobol(ReadAddr)

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

    CallCobol2(;,ReadAddr,&record0, Pos);

DM_SetRecord(arg0,&record0,RecCobInfoAddress,
    sizeof(RecCobInfoAddress)/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 RecInitRecTest __1((DM_ID, dialog))
{
    return DM_BindCallBacks(RecMap, RecMapCount, dialog,
          DMF_Silent);
}

void cobrecinitrectest __2((short *, dialog), (boolean *,
          result))
{
    *result = RecInitRecTest((DM_ID) *dialog);
}
```

5.8.3 Dynamic Binding of Record Functions

Availability

Since IDM version A.06.01.d

“Dynamic binding” refers to a binding type for application functions that does not require explicitly setting the function pointer of the application functions by means of the functions DM_BindFunctions or DM_BindCallBacks. This includes, for example, connecting functions from dynamic libraries (transport *"dynlib"*) or calling COBOL functions by their names through the “BindThruLoader Functionality”. However, since application functions with **record parameters** require an additional stub function for marshalling the record structure, no structure has been transferred so far when these record functions were called.

As of IDM version A.06.01.d, the structure of the record parameters for dynamically bound record functions is transferred without a stub function. Such record functions can be used without generating a C or COBOL code. It is however useful to generate the function prototypes as well as the structure definition via **+writeheader <basefilename>**.

Example

```
dialog D
record RecAdr {
  string[80] Name := "";
  string[120] Street := "";
  string[40] City := "";
  integer ZIP := 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 "Application(s) not properly connected!";
  elseif not Search("Joe", RecAdr) then
    print "Joe not found, adding him";
    RecAdr.Name := "Joe";
    RecAdr.Street := "765 East Walnut Lane";
    RecAdr.City := "Oak Park";
    RecAdr.ZIP := 48237;
    New(RecAdr);
  else
    print "Joe found, lives in: "+RecAdr.City;
  endif
  exit(();
}
```

The prototypes can now simply be generated with **+writeheader** instead of **+writeproto**, **+write-funcmap** and **+/-writetrampolin**:

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

The actual implementation in C and COBOL can now find the function prototypes or record definitions for “RecRecAdr” in **adr.h** (C header file) and **adr.cpy** (COBOL copy file).

To generate a copy file that also contains *National Character* (MICRO FOCUS VISUAL COBOL only), this has to be specified with the option **-mfviscob-u** .

6 Functions of the COBOL Dialog Manager Interface

In this chapter the functions of the DM interface are described. After a short summary (chapter “Overview of Functions”) the different function tasks are explained (from chapter “Initializing and Starting the Dialog Manager” to “Access to Listbox and Tablefield”). In chapter “Functions in Alphabetical Order” you will find a detailed description of the functions.

The following symbols are used in the parameter description:

->	input parameter
<-	output parameter
<->	input and output parameter

6.1 Overview of Functions

Function Name	Short Description
COBOLAPPFINISH	Finish routine (distributed applications)
COBOLAPPINIT	Start routine (distributed applications)
COBOLMAIN	COBOL main program of a local application
DMcob_CallFunction	Function call in any part of the application (distributed application)
DMcob_CallMethod	Calling methods of IDM objects
DMcob_CallRule	Calling rule by parameters
DMcob_Control	Triggering an action
DMcob_CreateFromModel	Creates an object that is derived from a Model
DMcob_DataChanged	Signals that the value of the specified attribute (Model attribute) has changed on a particular Data Model (Model component).
DMcob_CreateObject	Creates an object of a given class
DMcob_DeleteArgString	Deleting an argument which was passed on the program
DMcob_Destroy	Destroying of objects

Function Name	Short Description
DMcob_DialogPathToID	Changing external object name to internal DM ID
DMcob_ErrMsgText	Text belonging to an error code
DMcob_EventLoop	Starting the dialog processing
DMcob_ExecRule	Starting of named rules
DMcob_FatalAppError	Error handling
DMcob_FreeVector	Freeing of temporary memory
DMcob_GetArgCount	Inquiring number of arguments passed to the program
DMcob_GetArgString	Inquiring an argument which was passed to the program
DMcob_GetValue	Inquiring DM object attributes
DMcob_GetValueBuff	Inquiring DM object attributes
DMcob_GetVector	Querying a vector attribute in a temporary memory
DMcob_GetVectorValue	Sets a value in a temporary memory
DMcob_Initialize	Initializing DM
DMcob_InitVector	Creating a temporary memory
DMcob_LoadDialog	Loading of dialog in DM
DMcob_LoadProfile	Loading of user-specific settings
DMcob_OpenBox	Open messagebox or dialogbox
DMcob_OpenBoxBuff	Open messagebox or dialogbox using a buffer for string values
DMcob_PathToID	Changing external object name to internal DM ID
DMcob_PutArgString	Replacing an argument which was transferred to the pro-gram
DMcob_QueryBox	Open messagebox
DMcob_QueryError	Inquiring errors
DMcob_QueueExtEvent	Sending an external event
DMcob_ResetValue	Resetting DM object attributes to model or default value

Function Name	Short Description
DMcob_SetValue	Changing DM object attributes
DMcob_SetValueBuff	Changing DM object attributes with unspecified transfer area
DMcob_SetVector	Assigning a temporary memory to an object
DMcob_SetVectorValue	Querying a value in a temporary memory
DMcob_ShutDown	Controlled finishing of the dialog manager application
DMcob_StartDialog	Starting the actual dialog
DMcob_StopDialog	Stopping the actual dialog
DMcob_TraceMessage	Writing a trace message into the trace file
DMcob_ValueChange	Replacing the entire value or an item value in a collection
DMcob_ValueChangeBuffer	Replacing the entire value or an item value in a collection using a transfer buffer of definable size
DMcob_ValueCount	Returning the number of values (without the default values), the index type, or the highest index value of a collection
DMcob_ValueGet	Retrieving a single item value from a collection at a defined index
DMcob_ValueGetBuffer	Retrieving a single item value from a collection at a defined index using a transfer buffer of definable size
DMcob_ValueGetType	Querying the data type of a value managed by the IDM
DMcob_ValueIndex	Determining the index in a collection belonging to a position
DMcob_ValueInit	Converting a value into a local or global value reference managed by the IDM
DMmfviscob_BindThruLoader	Marking all functions defined in the dialog so that they will be called by the COBOL runtime system and are not linked to the application (MICRO FOCUS VISUAL COBOL)
DMufcob_BindThruLoader	Marking all functions defined in the dialog so that they will be called by the COBOL runtime system and are not linked to the application (MICRO FOCUS Server Express)

6.2 Initializing and Starting the Dialog Manager

The following functions are necessary for initializing a dialog and executing a dialog. You have to include these functions in your main program:

- DMcob_EventLoop
  This function starts the processing in the Dialog Manager. Without this call, no interaction would be possible with the user interface.
- DMcob_Initialize
  This function initializes the Dialog Manager.
- DMcob_LoadDialog
  This function loads a dialog into the Dialog Manager. This dialog description can be an ASCII-file or a binary file.
- DMcob_LoadProfile
  This function loads the user-dependent configuration file by means of which specially marked variables can be changed.
- DMcob_StartDialog
  This function starts the dialog. It initializes the window system and makes the windows visible which were defined as visible in the file. It also executes the "dialog start rule".
- DMmfviscob_BindThruLoader
  This function is used for initialization, when the COBOL runtime system shall be enabled to address functions without the need for a function table. This function is intended for MICRO FOCUS VISUAL COBOL.
- DMufcob_BindThruLoader
  This function is used for initialization, when the COBOL runtime system shall be enabled to address functions without the need for a function table. This function is intended for MICRO FOCUS Server Express.

6.3 Finishing the ISA Dialog Manager

These functions are used to stop the execution of a dialog and to terminate the IDM in a controlled manner:

- DMcob_ShutDown
  This function shuts down the IDM and revokes global initializations. Usually it is only needed, when the **Main** program of the IDM is replaced by an application-specific one.
- DMcob_StopDialog
  This function is used to stop the current or a specified dialog. The `on dialog finish` rule is executed.

6.4 Access Functions

By means of the following functions, you can manipulate the Dialog Manager objects and their attributes.

6.4.1 Accessing the Dialog Manager Identifiers

If you want to access the Dialog Manager identifier for an object, you need the name of the object. Together with this name and a call of the following functions, you get the Dialog Manager internal identifier of the object.

- DMcob_DialogPathToID
  This function returns the Dialog Manager identifier of an object, too, but this function is able to handle more than one dialog.
- DMcob_PathToID
  This function returns the Dialog Manager identifier of the object. This ID has to be used for all other calls to the Dialog Manager.

6.4.2 Accessing Object Attributes

To access any object attribute, you have to know the identifier, the data type and the name of the required attribute.

- DMcob_DataChanged
  This function is used to signal that the value of the specified attribute (Model attribute) has changed on a particular Data Model (Model component).
- DMcob_GetValue
  This function returns the value of the required attribute of any object.
- DMcob_GetValueBuff
  This function returns the string value of an attribute in a separate buffer.
- DMcob_ResetValue
  This function resets the attribute to the value of the object model or default.
- DMcob_SetValue
  This function changes the attribute to the passed value.
- DMcob_SetValueBuff
  This function changes the attributes to the passed value. This function requires a special buffer which can have any length you want.

6.4.3 Creating and Destroying Objects

Objects can be dynamically created and destroyed with the following functions.

- DMcob_CreateFromModel
  This function creates an object that is derived from a Model and returns the identifier of the created object.
- DMcob_CreateObject
  This function creates an object of the specified class and returns its identifier. After the object has been successfully created, it can be accessed by a Dialog Manager function.
- DMcob_Destroy
  This function destroys any object. After a call to this function, the objects cannot be accessed by any Dialog Manager function.

6.4.4 Utilities

- DMcob_CallFunction
  Function used to call other functions anywhere in the application.
- DMcob_CallMethod
  With this function, methods of IDM objects can be called from then application.
- DMcob_CallRule
  By means of this function, specified rules can be called up with parameters from the application. In contrast to **DMcob_ExecRule**, **DMcob_CallRule** allows parameters and return values when calling dialog rules.
- DMcob_Control
  This function allows the application to refresh the screen, to switch the currently used code page, or to lock the keyboard.
- DMcob_ExecRule
  This function starts the processing of the specified rule as if it was called by "perform" in another rule.
- DMcob_OpenBox
  With this function a specified messagebox or dialogbox can be opened.
- DMcob_OpenBoxBuff
  With this function a specified messagebox or dialogbox can be opened using a buffer for the transfer of string values.
- DMcob_QueryBox
  The use of this function allows to open messageboxes.
- DMcob_QueueExtEvent
  With this function an event can be put into a queue. In doing so, a rule related to an external event is executed.
- DMcob_TraceMessage
  This function allows the application to write any string into the tracefile.

6.4.5 Access to Listbox and Tablefield

By means of the following functions you can efficiently handle contents of listboxes and tablefields. This functions should be used for setting and requesting contents of listboxes and tablefields especially when employing the distributed Dialog Manager

- DMcob_FreeVector
  Freeing of a temporary memory in Dialog Manager.
- DMcob_GetVector
  By means of this function you can query an arbitrary vector of object attribute values by a function call.
- DMcob_GetVectorValue
  With this function you can query an element of a vector of object attribute values.
- DMcob_InitVector
  This function puts up a temporary memory in Dialog Manager which can take any object attribute values of an object.
- DMcob_SetVector
  With this function you can pass a temporary memory to an object.
- DMcob_SetVectorValue
  With this function you can store an object attribute value in a temporary area.

6.5 Working with Collections

- DMcob_ValueChange
  With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection.
- DMcob_ValueChangeBuffer
  With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection. Unlike **DMcob_ValueChange**, this function provides a buffer for string return values that is larger than the standard buffer.
- DMcob_ValueCount
  Returns the number of values in a collection (without the default values). It is also possible to return the index type or the highest index value.
- DMcob_ValueGet
  This function allows to retrieve a single element value that belongs to a defined index from collections.
- DMcob_ValueGetBuffer
  This function allows to retrieve a single element value that belongs to a defined index from collections. Unlike **DMcob_ValueGet**, this function provides a buffer for string return values that is larger than the standard buffer.
- DMcob_ValueGetType
  This function queries the data type of a value managed by the IDM.

- DMcob_ValueIndex
  This function can be used to determine the corresponding index for a position in a collection.
- DMcob_ValueInit
  With this function a value can be converted into a local or global value reference managed by the IDM. This allows the further manipulation of the value by **DMcob_Value…()** functions and its transfer as parameter or return value.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

6.6 Error Handling

If a function call to the Dialog Manager fails, the application can query the error by means of the following utilities:

- DMcob_ErrMsgText
  This function returns the text string to an error code.
- DMcob_FatalAppError
  This function is to be called if the application detects a serious error and is not able to continue. It finishes all work in the Dialog Manager.
- DMcob_QueryError
  This function returns the number of currently present errors and their error codes.

6.6.1 Information in the Error Code

The error code is a combination of:

- the severity of the error,
- the error code itself,
- and the module in which the error has occurred.

To extract the severity of an error, you have to use the data field *DM-error-severity* of the structure **DM-ErrorDetail**.

The following values are possible for severity:

DM-SeveritySuccess	no error occurred
DM-SeverityWarning	warning
DM-SeverityError	error
DM-SeverityFatal	fatal error, Dialog Manager cannot continue.
DM-SeverityProgErr	program error

To get the module which caused the error, the data structure **DM-error-package** can be used.

The following modules are possible:

DM-ModuleIDM	the error was inside the Dialog Manager
DM-ModuleUnix	the error occurred by a function call to the operation system UNIX.

The error code itself is in the data structure **DM-error-code**.

The possible error codes are defined in **IDMcobws.cob**.

6.7 Special Functions

With the following functions you are able to access and manipulate the arguments which were transferred to the program.

- DMcob_DeleteArgString
  This functions deletes an argument.
- DMcob_GetArgCount
  This function returns the number of arguments which were transferred to the program.
- DMcob_GetArgString
  This function returns the argument string of any argument.
- DMcob_PutArgString
  This function replaces an argument with a new value.

6.8 Functions in Alphabetical Order

6.8.1 COBOLMAIN

This function is the main function of the application. It is called by the DM immediately after the program has started. This function should carry out the necessary function calls to the application and to the Dialog Manager. It gets the same parameters as the normal "main" function of a usual COBOL program.

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

Example

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

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

77    DM-dialogid    pic 9(4) 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.
```

If an application is not linked directly to the DM part of the display, i.e. if it works in a distributed environment, the functions

COBOLAPPINIT and COBOLAPPFINISH

have to be used instead of the function COBOLMAIN.

See Also

Object Application in the “Object Reference”

Manual “Distributed Dialog Manager (DDM)”

6.8.1.1 COBOLAPPINIT

This function starts the application and executes the steps necessary for the initialization. This function is only used in the distributed version!

```
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.
```

Parameters

<-> EXIT-STATUS

Return value of the COBOL function. If this value is set at *DM-success*, the application was initialized correctly. If this value is set at any other value other than *DM-success*, the application could not be initialized without any error.

-> DM-APPL-ID

Identifier of the application which is to be initialized.

-> DM-DIALOGID

Identifier of the dialog to which the application belongs.

If the function returns *DM-success*, the application has been started successfully. If the initialization was not successful, an error code should be returned.

Example

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

Note

COBOLAPPINIT has to call the **DMcob_Initialize** to pass the *DM-Common-Data* to the Dialog Manager!

6.8.1.2 COBOLAPPFINISH

This function finishes the application in a distributed environment.

```
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.
```

Parameters

<-> EXIT-STATUS

Return value of the COBOL function. If this value is set at *DM-success*, the application was initialized correctly. If this value is set at any other value other than *DM-success*, the application could not be initialized without any error.

-> DM-APPL-ID

Identifier of application which is to be initialized.

-> DM-DIALOGID

Identifier of the dialog to which the application belongs.

Example

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

6.8.2 DM_BindCallBacks

For information about this function, please refer to the description of DM_BindCallBacks in manual "C Interface - Functions".

6.8.3 DM_BindFunctions

For information about this function, please refer to the description of DM_BindFunctions in manual "C Interface - Functions".

6.8.4 DM_BootStrap

For information about this function, please refer to the description of DM_BootStrap in manual "C Interface - Functions".

6.8.5 DMcob_CallFunction

To call any functions that are known to the DM in other parts of the application, you have to use the function **DMcob_CallFunction**. This function then calls the corresponding function.

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

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

Parameters

<- DM-Value

In this parameter the Dialog Manager returns the return value of the function if the function call could be executed. To access this value, the data type has to be checked first. With this value the structure can be accessed.

-> DM-funcID

This is the identifier of the function which should be called.

<-> DM-ValueArray

This parameter is an array of values which shall be used as parameters for the function call. The length of this vector may be up to 16. Depending on the parameter type, the corresponding structure element has to be filled. If the function has arguments declared as input and output, the calling function can access these arguments after the return of the DMcob_CallFunction.

-> DM-value-count of DM-ValueArray

In this parameter the number of function parameters has to be passed to the Dialog Manager. This number has to be equal to the declaration of the function inside of the dialog script; otherwise the Dialog Manager does not call the function.

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

In this parameter the data type of the "index" parameter is transferred to the Dialog Manager. Depending on the value of this structure element, the value has to be stored in the corresponding structure element.

Note

The data type of the parameter may be changed by the Dialog Manager! So if a function has input/output parameters, the value in this structure element has to be checked before the real value can be accessed.

-> DM-Options of DM-StdArgs

Normally the options should be set to DMF-GetMasterString to get strings as return values and not any TextIDs. Any option can be added to this.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string* or *DM-va-value-string*, it is also possible to use *DM-value-string-u* or *DM-va-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* and *DM-va-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	Function could not be called.
DM-success	Function was successfully called.

Example

To call the function

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

the COBOL program should look as follows:

```
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).
```

See Also

Object Application

Manual “Distributed Dialog Manager (DDM)”

6.8.6 DMcob_CallMethod

By means of this function, you can call methods by parameters from the application. The parameters depend on the object and the called method.

```
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.
```

Parameters

<- DM-Value

In this parameter, the return value of the method is returned, if the method call was carried out. To access this value, first of all the data type has to be checked. Then, according to this value, you have to access the structure.

-> DM-objectID

This is the identifier of the object at which the method is to be called.

-> DM-method

This is the method that is to be called.

These methods are defined in the copy file **IDMcobls.cob** or **IDMcobws.cob**.

The defined constants have the following meaning:

- *MT-insert*
  inserting of rows/columns of a tablefield
- *MT-delete*
  deleting of rows/columns of a tablefield
- *MT-clear*
  deleting of contents in rows/columns of a tablefield

<-> DM-ValueArray

This parameter is an array of values which are to be taken as parameters for the method call. The maximal length of this vector can be 16. Depending on the parameter data type, the corresponding structure element has to be filled.

-> DM-count of DM-ValueArray

In this parameter the actual number of set values in the array has to be indicated. This number has to be the identical with the parameters permitted for the method. If this is not the case, the method will not be called by the Dialog Manager.

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

In this element the data type of the parameter with the number "index" has to be transferred to the Dialog Manager. Depending on this value, the corresponding structure element has to be set.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with 0.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string* or *DM-va-value-string*, it is also possible to use *DM-value-string-u* or *DM-va-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* and *DM-va-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	Method could not be called.
DM-success	Method could be carried out.

The following table lists the meanings of the parameters for the tablefield methods:

	MT-insert	MT-delete	MT-clear
DM-va-datatype(1)	DT-integer	DT-integer	DT-integer
DM-va-value-integer (1)	Position, where rows/-columns are to be inserted	Position, where rows/-columns are to be deleted	Position, from which on the contents of rows or columns is to be deleted
DM-va-datatype(2)	DT-integer	DT-integer	DT-integer
DM-va-value-integer (2)	Number of the rows/-columns to be inserted	Number of the rows/-columns to be deleted	End, up to which the contents of rows or columns is to be deleted
DM-va-datatype(3)	DT-boolean	DT-boolean	DT-boolean

	MT-insert	MT-delete	MT-clear
DM-va-value-boolean (3)	Describes whether it is to be inserted against the dynamic direction (TRUE) or with the direction	Describes whether it is to be deleted against the dynamic direction (TRUE) or with the direction	Describes whether the contents are to be deleted against the dynamic direction (TRUE) or with the direction

Example

Unloading a tablefield in a reloading function.

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

* Deleting of contents in invisible area

    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

Specified rules and parameters can be called from the application by this function.

Whereas DMcob_ExecRule allows only rules without parameters to be called from the application, DMcob_CallRule allows parameters and return values when calling dialog rules.

```
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.
```

Parameters

<- DM-Value

In this parameter the Dialog Manager transfers the return value of the rule, if the rule call could be executed. To access this value, first the data type has to be checked. The structure can be accessed according to this value.

-> DM-ruleID

This is the identifier of the function which should be called.

-> DM-objectID

This is the identifier of the object which should be used as the "this" object in the rule.

<-> DM-ValueArray

In this parameter the rule arguments are transferred to the Dialog Manager. The length of this vector can be up to 16 parameters. Depending on the parameter type the corresponding element of the structure have to be filled. If the rule has arguments declared as input and output, the calling function can access these arguments after the return of the DMcob_CallRule.

-> DM-count of DM-ValueArray

In this parameter the number of rule parameters has to be passed to the Dialog Manager. This number has to be equal to the declaration of the rule inside of the dialog script; otherwise the Dialog Manager does not call the rule.

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

In this parameter the data type of the "index" parameter is passed to the Dialog Manager Corresponding to the value of this structure element the value has to be stored in the correct structure element.

Note

The data type of the parameter may be changed by the Dialog Manager! So if a function has input/output parameters, the value in this structure element has to be checked before the real value can be accessed.

-> DM-Options of DM-StdArgs

If strings are used as parameters, the DMF-GetMasterString should be set so that no TextIDs are returned. However, any other option can be added to this value.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string* or *DM-va-value-string*, it is also possible to use *DM-value-string-u* or *DM-va-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen, *DM-value-string-putlen*, *DM-va-value-string-getlen* and *DM-va-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	Rule could not be called.
DM-success	Rule was successfully called.

Example

To call the rule

```
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);
}
```

the COBOL program should look like this:

```
    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

With this function, general settings in the ISA Dialog Manager can be changed or actions can be triggered.

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

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

Parameters

-> DM-objectid

This parameter specifies the object to which the action to be triggered refers.

-> action

Specifies the action the Dialog Manager is to execute. To do so, there are different constants defined in the copy file **IDMcobws.cob**. These constants are described in the table below.

-> DM-Options of DM-StdArgs

This parameter possibly is used for an argument required by *action* (see table below).

Return Value

DM-status

DM-error	The action could not be executed.
DM-success	The action was executed successfully.

The following table shows the valid assignments of the individual parameters and describes their meanings. When nothing else is stated with the action, *DM-objectid* should be *0*.

action	options	Meaning
DMF-UpdateScreen	0	All internal SetVal calls are to be brought on screen. To do so, the first parameter has to be specified with the dialog.
DMF-SignalMode	0	The signals are intercepted by the function "signal".
	1	The signals are intercepted by the function "sigaction".

action	options	Meaning
DMF-SetCodePage		With this action the code page for the transfer of strings between application and IDM can be set. Usually, IDM expects and returns strings that are encoded according to the ISO 8859-1 standard. With this action a different character encoding can be defined. As of IDM version A.06.01.d, it is possible to specify an ***Application*** object in the *objectid* parameter. This changes the application-specific code page that is required for processing strings. The change of an application-specific code page within one of the functions of the corresponding application has an immediate effect. However, the call on a DDM server side does not support changing the application code page, but only affects the network application anyway.
DMF-SetFormatCodePage		Defines the code page in which format functions expect and return strings.
The options below apply to DMF-SetCodePage and DMF-SetFormatCodePage		
	CP-ascii	ASCII character encoding.
	CP-iso8859	Western European Latin-1 encoding according to ISO 8859-1.
	CP-cp437	English character encoding according IBM code page 437 (MS-DOS).
	CP-cp850	Western European character encoding according to IBM code page 850 (MS-DOS).
	CP-iso6937	Western European character encoding with variable length according to ISO 6937..
	CP-winansi	Microsoft Windows character encoding.
	CP-dec169	Character encoding according to DEC code page 169.
	CP-roman8	8-bit character encoding according to HP code page Roman-8.

action	options	Meaning
	CP-utf8	8-bit Unicode encoding with variable length, corresponds to ASCII encoding for the characters 0 to 127.
	CP-utf16 CP-utf16b CP-utf16l	16-bit Unicode encoding with character widths from 2 up to 4 bytes. There are two variants: » BE – big-endian, bytes with higher numerical significance first. » LE – little-endian, bytes with lower numerical significance first. UTF-16 without a specified byte order corresponds to the LE variant on Microsoft Windows and to the BE variant on Unix systems.
	CP-cp1252	Western European character encoding according to Microsoft Windows code page 1252.
	CP-acp	Currently used “ANSI code page” of an application on Microsoft Windows. Can only be used on Microsoft Windows.
	CP-hp15	Western European 16-bit character encoding used by HP systems.
	CP-jap15	Japanese 16-bit character encoding used by HP systems.
	CP-roc15	Simplified Chinese 16-bit character encoding used by HP systems.
	CP-prc15	Traditional Chinese 16-bit character encoding used by HP systems.

Note

Switching to a code page is valid until another code page is set. All strings have to be transferred in this code page and all strings which IDM transfers to the application use this code page.

Example

To set “IBM 437” as code page, the call should look like this:

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

move CP-cp437 to action.
```

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

6.8.9 DMcob_CreateFromModel

This function can be used to create an object based on any model as instance or model.

The parameters define the model from which the newly created object is derived, as well as the parent, and the object type (instance or model) of the newly created object.

```
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.
```

Parameters

-> DM-Options of DM-StdArgs

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

Option	Meaning
0	Creates an instance.
DMF-CreateModel	Creates a model.
DMF-CreateInvisible	The newly generated object is created invisible, regardless of the setting at its Model or Default.

<- DM-objectid

In this parameter, the identifier of the newly created object is returned.

-> DM-parentid

This parameter defines the parent of the newly created object.

-> DM-modelid

This parameter defines the model from which the newly created object is derived. If it is a hierarchical model, its children are also created in the newly generated object.

Return value

DM-status of DM-StdArgs

DM-error	Object could not be created.
DM-success	Object was successfully created.

Availability

The function **DMcob_CreateFromModel** is available since IDM version A.06.01.g.

See also

Function DMcob_CreateObject

Built-in function create() in manual “Rule Language”

Method :create()

6.8.10 DMcob_CreateObject

This function can be used to create an object of any object class as instance or model.

The parameters define the object class (pushbutton, window, etc.), the parent, and the object type (instance or model) of the newly created object.

```
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.
```

Parameters

-> DM-Options of DM-StdArgs

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

Option	Meaning
0	Creates an instance.
DMF-CreateModel	Creates a model.
DMF-CreateInvisible	The newly generated object is created invisible, regardless of the setting at its Default.
DMF-InheritFromModel	This option was intended to create an object which was based on the Model specified in the *classid* parameter. However, *DMF_InheritFromModel* should **not** be used. Instead, objects derived from a model should be created using the DMcob_CreateFromModel function.

<- DM-objectid

In this parameter, the identifier of the newly created object is returned.

-> DM-parentid

This parameter defines the parent of the newly created object.

-> DM-classid

This parameter determines the class of the new object. All class definition constants are contained in the **IDMcobws.cob** file that comes with the IDM.

Value range

- 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

Return value

DM-status of DM-StdArgs

DM-error	Object could not be created.
DM-success	Object was successfully created.

Example

Creating a new window.

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

See also

Function DMcob_CreateFromModel

Built-in function create() in manual “Rule Language”

Method :create()

6.8.11 DMcob_DataChanged

This function is used to signal that the value of the specified attribute (Model attribute) has changed on a particular Data Model (Model component). This change is put into event processing as a *datachanged* event. Not until further event processing the linked presentation objects (View components) are triggered to fetch the new data values and have them displayed.

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

Parameters

<-> DM-options of DM-StdArgs

Option	Meaning
0	There will be no tracing of this function.
DMF-Verbose	Activates tracing of this function.

-> DM-object of DM-Value

This is the object ID of the Data Model object where a data value has changed. The object ID may be obtained as return value of the function DMcob_PathToID.

-> DM-attribute of DM-Value

This parameter indicates the attribute of the Data Model that has changed.

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

This parameter defines the index of the modified attribute.

Note

If no index shall be specified, *DM-datatype* of **DM-Value** has to be set to the value *DT-void*.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

6.8.12 DMcob_DeleteArgString

This function deletes an argument in the internal list. The argument was transferred by the user to the program.

All arguments of the command line which are processed by the application should be deleted; otherwise the Dialog Manager checks the arguments and processes them, if necessary.

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

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

Parameters

-> DM-Index

The number of the argument which you want to delete. The value of this parameter must be greater or equal zero and must be smaller or equal to the number of arguments.

Return Value

DM-status of DM-StdArgs

DM-error	Argument could not be deleted.
DM-success	Argument was successfully deleted.

Example

See function DMcob_GetArgString

6.8.13 DMcob_Destroy

Any object or model of the dialog can be deleted with this function. All object children are also deleted.

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

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

Parameters

-> DM-Object

The ID of the object you want to delete.

-> DM-Options

Controls the function behavior during the deletion.

DMF-ForceDestroy:

If *DMF-ForceDestroy* is given here, the object is deleted and all rule parts using this object are changed, so that the corresponding commands are removed.

If the object to be deleted is a model, using this function without *DMF-ForceDestroy* prohibits only a new reference to this model; the model itself remains. If *DMF-ForceDestroy* is used, the model is removed from all objects using it and the objects adopt the values of the next superordinate model or default.

Return Value

DM-status of DM-StdArgs

DM-error	Object could not be deleted.
DM-success	Object was successfully deleted.

DMcob_Destroy() invokes the :clean() method of the object to be destroyed.

Example

Deleting a window “MyWindow”.

```
move DMF-Forcedestroy to DM-Options.
Call "DMcob_DestroyObject" using DM-StdArgs DM-object.
```

See Also

Built-in function destroy() in manual “Rule Language”

Method :destroy()

6.8.14 DMcob_DialogPathToID

By means of this function you can inquire the identifier of an object if you have loaded more than one dialog and the searched object is not contained in the dialog which was loaded first.

```
77  DM-dialogid  pic 9(4) binary value 0.
77  DM-rootid    pic 9(4) binary value 0.
77  DM-objectid  pic 9(4) binary value 0.
77  DM-path      pic X(256).

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

The meaning of parameters is the same as in DMcob_PathToID.

Parameters

<- DM-objectid

Identifier of the specified object, if the object could be found in the specified dialog.

-> DM-dialogid

Identifier of the dialog in which the object is to be searched. You have received this value as the return value from DMcob_LoadDialog.

-> DM-rootid

This parameter controls from which object the Dialog Manager is to begin searching your desired object. You have the following options:

» *rootid = 0*
The Dialog Manager searches in the entire dialog definition for the specified object. This is the usual option. The identifiers of rules, functions, variables and resources can also be inquired in this way.

» *rootid != 0*
The Dialog Manager is to search the desired object on the next subordinate hierarchy level from the specified object.
This method is appropriate only if an object identifier occurs more than once in a dialog. Rules, functions, variables and resources **cannot** be inquired this way.

-> DM-path

This path describes the searched object. The path has to describe an object uniquely. If the object identifier exists only once in the dialog, giving the identifier is sufficient to get the desired reference. If the object identifier is not unique, the object has to be described by a path of object identifiers, separated by a dot.

The path should be terminated with a low value or the separator character. Otherwise no more than 256 characters are read.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

Return Value

DM-status of DM-StdArgs

DM-error	The object was not found or its identifier is not unique.
DM-success	Identifier of the searched object.

Example

Inquiring the ID of an object named "MyWindow" in the dialog "SecondDialog".

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

6.8.15 DMcob_ErrMsgText

This function returns the error string belonging to an error code.

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

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

Parameters

-> DM-Options of DM-StdArgs

» *DMF-IncludeModule*
If this option is set, the module in which the error occurred is added to the string.

» *DMF-IncludeSeverity*
If this option is set, the severity of the error is copied to the string.

» *DMF-IncludeText*
If this option is set, the error message itself is copied to the string.

-> DM-ErrorDetail

This is the error code to which the error message should be returned.

<- DM-string

The error message is copied into this parameter. The parameter must be defined with a length of 80 characters.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

Example

```
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

This function starts the dialog processing. It enables the user to work with the dialog. After this function is called, events are processed only by the DM because this function takes over dialog handling.

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

Parameters

-> DM-Options of DM-StdArgs

This parameter determines whether event processing is to be canceled when no further event exists, or if the process is to be carried on until the dialog ends. If the Dialog Manager does not have to wait, the constant

DMF-DontWait

has to be indicated here; otherwise *0* has to be given. If *DMF-DontWait* is given here, the calling program has to secure by means of a program loop that the DM event processing (DMcob_EventLoop) is called repeatedly.

Return Value

DM-ResCode of DM-StdArgs

DM-error	Event processing was canceled because the end of the dialog has been reached.
DM-success	Event processing was canceled because no further event was scheduled to be processed at the moment.

Example

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

6.8.17 DMcob_ExecRule

By means of this function, named rules can be processed through the COBOL interface. These rules are then executed as if they had been called within the Rule Language.

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

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

Parameters

-> DM-ruleid

This parameter identifies the rule to be processed. You have received this identifier via the function DMcob_PathToID.

-> DM-objectid

This parameter corresponds to *this* in the corresponding rule. This means that *this* in the rule is replaced by the object you specify here.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Return Value

DM-Status of DM-StdArgs

DM-error	Rule could not be executed.
DM-success	Rule could be executed.

Example

To call the rule

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

the COBOL program could look like

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

See Also

Function DMcob_CallRule

6.8.18 DMcob_FatalAppError

This function should be called if the application found an error and is not able to continue. This function finishes the Dialog Manager, closes all files, displays and returns the control to the application, if required.

```
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.
```

Parameters

-> DM-reaction

This parameter specifies the behavior of the function **DMcob_FatalAppError**. The following values are possible:

-1	The program will be exited by a core-dump, if the basic operation system allows it.
0	The user receives a message and then the program is closed as usual.
>0	The user receives a message and then the program is closed by the indicated value as exit code

->DM-string

Message which should be written into the tracefile. This is the reason why the application finishes. The message should be terminated with a low value or the separator character. Otherwise no more than 256 characters are read.

COBOL Interface for MICRO FOCUS VISUAL COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Availability

This function is available on UNIX and since IDM version A.06.01.d on MICROSOFT WINDOWS too.

6.8.19 DMcob_FreeVector

By means of this function, a temporary memory is freed again.

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

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

Parameters

-> DM-POINTER

Identifier of the memory to be freed. This ID comes from either DMcob_InitVector or DMcob_ GetVector.

-> DM-Opions of DM-StdArgs

Currently not used. Please specify with *0*.

Return Value

DM-Status of DM-StdArgs

DM-error	Given ID was no memory.
DM-success	Memory was deleted.

6.8.20 DMcob_GetArgCount

This function enables you to inquire the number of arguments which were passed to the program.

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

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

Parameters

<- DM-ArgCount

This parameter returns the number of arguments which were passed to the program. Please note that the first argument is always the program name itself, so every program has at least one argument.

Return Value

DM-Status of DM-StdArgs

DM-success	The argument count could be inquired successfully.

6.8.21 DMcob_GetArgString

With this function you can access an argument which was passed on to the program.

```
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.
```

Parameters

-> DM-ArgNumber

This parameter is the number of the arguments which shall be returned by this function.

Note

The executable here has the number *0*, the first argument (e.g. the dialog file) has the number *1* etc.

The parameter *DM-ArgCount* of the function DMcob_GetArgCount however includes the executable in its count (and starts at *1* respectively)!

<-> DM-Buffer

In this parameter, the function should return the value of the argument. So this buffer must be large enough to store the argument in it.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-BufferSize* parameter, still the defined size has to be specified, for example *33* for *PIC N (33)*.

-> DM-BufferSize

This parameter is the size of the buffer you passed on to the Dialog Manager. The size must be exactly the size of the buffer because the Dialog Manager has no possibility to check it.

Return Value

DM-Status

DM-error	The argument could not be queried successfully. Possible errors are that the buffer is not big enough or the inquired argument does not exist.
DM-success	The argument could be queried successfully.

Example

The COBOL program has to look as follows to get the dialog to be loaded from the command line.

```
77    ArgName    pic X(80) value SPACES.
77    Buffer    pic X(80) value SPACES.
77    DialogID    pic 9(4) binary value 0.
77    DM-buffersize    pic 9(4) binary value 80.
77    DM-ArgCount    pic 9(4) binary value 0.
77    ArgNummer    pic 9(4) binary value 0.
77    ArgCount    pic 9(4) binary value 0.
77    Count    pic 9(4) binary value 0.
77    Hcount    pic 9(4) binary value 0.
CheckCommand.
    MOVE ZERO TO ArgAnz.
    CALL "DMcob_GetArgCount" USING DM-StdArgs DM-ArgCount.
    MOVE DM-ArgCount TO ArgCount.
    IF ArgCount >= 1
        MOVE 1 TO ArgNummer
        MOVE ZERO TO Count
        MOVE SPACES TO DialogName
        PERFORM GET-DIALOGNAME UNTIL ArgNummer >= ArgCount.
    END-IF.
CheckArgCount.
    EVALUATE Count
    WHEN 1
        PERFORM Lade-Dialog
    WHEN 0
        DISPLAY "Please give a dialogfile 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 ArgNummer
                            buffer DM-buffersize.
    MOVE buffer to ArgName.
    INSPECT ArgName TALLYING HCount FOR ALL '.dlg'.
    IF HCount = 1
        MOVE ArgName TO DialogName.
        ADD 1 TO Count
        MOVE ZERO TO Count
        PERFORM DeleteArg
    END-IF.
    ADD 1 TO ArgNummer.
DeleteArg.
    CALL "DMcob_DeleteArgString" USING DM-StdArgs ArgNummer.
```

6.8.22 DMcob_GetValue

With this function you can inquire attributes of DM objects. For the permitted attributes of the respective object type, please refer to the charts in chapter “Attributes and Definitions”.

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

Parameters

-> DM-Object of DM-Value

This parameter describes the object whose attribute you want to request. You have received this identifier as return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to inquire. All permitted attributes are defined in the **IDMcobws.cob** file.

-> DM-Index of DM-Value

This parameter is analyzed only in vector attributes of objects and describes the index of the searched sub-object (e.g. text in listbox).

<- DM-Datatype of DM-Value

In this parameter, you get the data type of the inquired attribute. You should take care to read this before you access the data. For the data type of each attribute please refer to chapter “Attributes and Definitions”.

-> DM-Indexcount of DM-Value

This parameter indicates how many index values are to be noticed when calling the function:

- 2-dimensional attributes -> value = 2 has to be set
- 1-dimensional attributes -> value = 1 has to be set
- non-indexed attributes -> value = 0 has to be set

-> DM-value-string-getlen of DM-Value

This parameter indicates how long the string to be inquired can be.

<- 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 one of these parameters you get the value of the inquired attribute. The value which is set depends on the data type of the attribute. For the data type of each attribute, please refer to chapter “Attributes and Definitions”.

-> DM-Options

With this parameter the form of the texts returned by the DM is controlled if the corresponding attribute has the type Text.

- *DMF-GetMasterString*
  The original string is returned if *DMF-GetMasterString* is specified here.
- *DMF-GetLocalString*
  The string is returned in the currently set language if *DMF-GetLocalString* is specified.
- *DMF-GetTextID*
  Only the text ID is returned if *DMF-GetTextID* is specified. This text ID can be used for settings with DMcob_SetValue.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-Status of DM-StdArgs

DM-error	The attribute is not permitted for this object.
DM-success	The object could be inquired successfully.

Example

To access the title of a window named "TestWindow" the COBOL program should look as follows:

```
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

With this function, you can inquire attributes of DM objects. The difference to DMcob_GetValue is that you can pass your own buffer for string return values to this function. This buffer can be greater than the standard buffer. For the permitted attributes of the respective object type, please refer to the charts in chapter “Attributes and Definitions”.

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

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

Parameters

-> DM-Object of DM-Value

This parameter describes the object whose attribute you want to inquire. You have received this identifier as return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to inquire. All permitted attributes are defined in the **IDMcobws.cob** file.

-> DM-Index of DM-Value

This parameter is analyzed only in vector attributes of objects and describes the index of the searched sub-object (e.g. text in listbox).

<- DM-Datatype of DM-Value

In this parameter you get the data type of the requested attribute. You should take care to read this before you access the data. For the data type of each attribute please refer to chapter “Attributes and Definitions”.

<- 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 one of these parameters you get the value of the inquired attribute. The value which is set depends on the data type of the attribute. For the data type of each attribute please refer to chapter “Attributes and Definitions”.

<- DM-string

In this parameter, you get the string of the inquired attribute. You should take care to read this because this is only valid if the returned data type is *DT-string*.

-> DM-stringlen

In this parameter, you pass the length of the passed string. This length must not be greater than the real string length of the passed string.

-> DM-Options of DM-StdArgs

The form of texts returned by the DM is controlled by this parameter if the corresponding attribute has the type Text.

» *DMF-GetMasterString*
The original string is returned if *DMF-GetMasterString* is specified here.

» *DMF-GetLocalString*
The string is returned in the currently set language if *DMF-GetLocalString* is specified.

» *DMF-GetTextID*
Only the text ID is returned if *DMF-GetTextID* is specified. This text ID can be used for settings with DMcob_SetValue.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return Value

DM-Status of DM-StdArgs

DM-error	The attribute is not permitted for this object.
DM-success	The object could be requested successfully.

Example

To access the contents of a listbox named "TestListbox" with large entries the COBOL program should look like this:

```
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

With this function, a vector attribute of an object is copied in a temporary memory.

```
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.
```

Parameters

-> DM-object of DM-Value

Identifier of the object the attributes of which are to be copied in the temporary memory.

-> DM-attribute of DM-Value

Object attribute that is to be copied in the temporary memory. All vector attributes of an object are permitted.

-> DM-indexcount of DM-Value

Number of indexes which are to be evaluated at the function call.

You have to indicate *1* if it is a normal attribute vector. If the field is two-dimensional (attributes of tablefield), you have to indicate *2*.

-> DM-index of DM-Value

Start index from which on the attribute is to be queried. If two indexes are necessary for the labeling, you have to indicate the row here.

-> DM-second of DM-Value

Second start index from which on the attribute is to be queried. This parameter is only evaluated if it is an attribute with two indexes and if *DM-indexcount* is set at *2*. It describes the column from which on the values are to be queried.

<- DM-POINTER

Identifier of the temporary memory that was created for the object attributes.

-> DM-ENDCOL

End index up to which the object attributes are to be queried. Value 0 means that everything from the start index on is to be queried. Value != 0 means that the respective attribute values at the object are queried.

-> DM-ENDROW

Second end index if it is a two-dimensional attribute. Value 0 means that everything from the start index on is to be queried. Value != 0 means that the respective attribute values at the obejct are queried.

<- COUNT

Returns the number of the queried values.

-> DM-options of DM-Value

Options to be used for the call.

Return Value

DM-Status of DM-StdArgs

DM-error	Values could not be queried.
DM-success	Values could be queried successfully.

Example

Setting a listbox contents with an unknown number of strings and queries of contents.

```
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(4) 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(4) 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.
  *Initialization of memory in DM
    CALL "DMcob_InitVector" USING DM-StdArgs DLG-ID DT-String
        ICOUNT.
  *Initialization of DM-Value structure
    MOVE DT-STRING TO DM DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.
  *Setting of individual contents
    PERFORM VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER = DLG-COUNT
        MOVE COUNTER TO DM-INDEX
        MOVE DLG-STRING TO STR-TAB
  *Preparation of a changed string for display
        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.
  *Transfer of the stored values to the display
    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.
  *Freeing of the memory
    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.
  *Inquiring individual contents
    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.
*Freeing of the memory
        CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
      GOBACK.
```

6.8.25 DMcob_GetVectorValue

With this function values from a temporary memory can be put into the COBOL program.

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

call "DMcob_GetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameters

-> DM-Index of DM-Value

Informs the function about the value of the temporary memory to be queried. Note that the first valid value has the index *1*. In addition, the value must not be larger than the temporary memory.

-> DM-Attibute of DM-Value

This parameter is only considered if the temporary memory was created with *DT-void*.

The following attributes are permitted: *AT-content*, *AT-userdata*, *AT-sensitive* and *AT-active*.

<- DM-Datatype of DM-Value

The data type of the queried attribute is set in this element.

<- DM-Value… of DM-Value

The actual value of the attribute is stored in this structure. Note that you can only access the field corresponding to the data type.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-Status of DM-StdArgs

DM-error	Attribute could not be queried.
DM-success	Attribute could be queried successfully.

Example

Request of the fifth string in a temporary memory.

```
77    DM-POINTER        pic 9(4) binary.
MOVE 5 TO DM-INDEX.
CALL     "DMcob_GetVectorValue" USING
    DM-StdArgs DM-Value DM-Pointer.
```

6.8.26 DMcob_Initialize

With this function, DM-internal data structures are initialized. This function must be called exactly once before the application is started.

```
call "DMcob_Initialize" using
            DM-StdArgs
            DM-Common-Data.
```

Parameters

-> DM-Common-Data

This parameter contains the common data structure of the COBOL program. This value is passed to any callback function to handle global data. The data structure can be changed by the user to add application specific data to this.

-> DM-setsep of DM-StdArgs

Notifies the Dialog Manager of the character which is to be the end of the strings passed on by the COBOL program.

-> DM-getsep of DM-StdArgs

Notifies the Dialog Manager of the character with which the strings - which are passed to the COBOL program - are to be filled.

-> DM-truncspaces of DM-StdArgs

Informs the Dialog Manager about whether blanks at the end of strings passed by the COBOL program are to be generally ignored.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-setsep* or *DM-getsep*, it is also possible to use *DM-setsep-u* or *DM-getsep-u* when working with Unicode texts (UTF-16).

Return Value

DM-status of DM-StdArgs

DM-error	DM was unable to initialize its internal structures correctly. In this case, the application should be canceled because it cannot continue working usefully.
DM-success	DM has successfully executed the initialization.

Example

```
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

With this function you can generate a temporary memory in the Dialog Manager to store there a larger amount of attribute values of an object. This memory should be used especially when setting contents for tablefields and listboxes.

```
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.
```

Parameters

<- DM-POINTER

Returns the newly generated memory. This value has to be given at all accesses to this memory.

-> DM-VALUETYPE

Specifies the data type of the attribute values to be stored. The necessary definitions are in the copy files **IDMcobws.cob** and **IDMcobls.cob**. Thus, possible values are *DT-String*, *DT-boolean*, *DT-instance*, *DT-integer*.

DT-void is an exception - if this value is given, space is created for four attributes (*.content, .user-data*, *.sensitive*, and *.active*).

-> DM-COUNT

Specifies the number of attributes which are internally stored. This number should correspond exactly with what is really needed. If *0* is given here, the section is created with a default size. If later on the generated section is not big enough, it is automatically enlarged by the Dialog Manager.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Return Value

DM-status of DM-StdArgs

DM-error	Memory could not be created.
DM-success	Memory could be created successfully.

Note

Memory created with **DMcob_InitVector** always has to be freed with DMcob_FreeVector!

Example

Memory for 100 string values is to be created.

```
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

With the help of this function you can request attributes of DM objects. The difference to DMcob_GetValue is that you can pass index values up to 65535 to this function.

Please refer to the chapter **"Attributes and definitions"** for the permissible attributes for the respective object type.

```
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.
```

Parameters

-> DM-Object of DM-Value

This parameter describes the object whose attributes you want to query. You have received this identifier as a return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to query. All allowed attributes are defined in the **IDMcobws.cob** file.

-> DM-LONG-FIRST of DM-Value

This parameter is evaluated only in vector attributes of objects and describes the index of the searched subobject (e.g. text in listbox).

-> DM-LONG-SECOND of DM-Value

This parameter is evaluated only for two-dimensional attributes.

<- DM-Datatype of DM-Value

With this parameter you get the data type of the attribute you are looking for. Thereby you have to consider the data type received from the DM before you access the data.

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

With one of these parameters you get the value of the requested attribute. The value that is set depends on the data type of the attribute.

For the data type of each attribute please refer to the chapter “Attributes and Definitions”.

<- DM-string

With this parameter you get the string of the requested attribute. You should pay attention to this, because it is valid only if the returned data type is *DT-string*.

-> DM-stringlen

With this parameter you pass the length of the passed string. This length must not be greater than the real string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter is used to control the form in which texts are returned from Dialog Manager if the corresponding attribute is of type Text.

- *DMF-GetMasterString*
  If *DMF-GetMasterString* is specified here, the original string is returned.
- *DMF-GetLocalString*
  If *DMF-GetLocalString* is specified here, the string is returned in the currently set language.
- *DMF-GetTextID*
  If *DMF-GetTextID* is specified here, the Dialog Manager returns only the ID of the text. This can then be used directly for setting via the function DMcob_SetValue.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-status of DM-StdArgs

DM-error	The attribute is not allowed for the object.
DM-success	The attribute could be queried successfully.

Example

To access the contents of the list box "TestListbox" with very long entries, the COBOL program must look something like the following:

```
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

With the help of this function you can request attributes of DM objects. The difference to DMcob_GetValue is that you can pass index values up to 65535 to this function and the buffer can have a size up to 65535 characters.

Please refer to the chapter **"Attributes and definitions"** for the permissible attributes for the respective object type.

```
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.
```

Parameters

-> DM-Object of DM-Value

This parameter describes the object whose attributes you want to query. You have received this identifier as a return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to query. All allowed attributes are defined in the **IDMcobws.cob** file.

-> DM-LONG-FIRST of DM-Value

This parameter is evaluated only in vector attributes of objects and describes the index of the searched subobject (e.g. text in listbox).

-> DM-LONG-SECOND of DM-Value

This parameter is evaluated only for two-dimensional attributes.

<- DM-Datatype of DM-Value

With this parameter you get the data type of the attribute you are looking for. Thereby you have to consider the data type received from the DM before you access the data.

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

With one of these parameters you get the value of the requested attribute. The value that is set depends on the data type of the attribute.

For the data type of each attribute please refer to the chapter “Attributes and Definitions”.

<- DM-string

With this parameter you get the string of the requested attribute. You should pay attention to this, because it is valid only if the returned data type is *DT-string*.

-> DM-stringlen

With this parameter you pass the length of the passed string. This length must not be greater than the real string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter is used to control the form in which texts are returned from Dialog Manager if the corresponding attribute is of type Text.

- *DMF-GetMasterString*
  If *DMF-GetMasterString* is specified here, the original string is returned.
- *DMF-GetLocalString*
  If *DMF-GetLocalString* is specified here, the string is returned in the currently set language.
- *DMF-GetTextID*
  If *DMF-GetTextID* is specified here, the Dialog Manager returns only the ID of the text. This can then be used directly for setting via the function DMcob_SetValue.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-status of DM-StdArgs

DM-error	The attribute is not allowed for the object.
DM-success	The attribute could be queried successfully.

Example

To access the contents of the list box "TestListbox" with very long entries, the COBOL program must look something like the following:

```
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

This function is used to copy a vector attribute from an object to a temporary memory area. This function is able to work with index values up to 65535.

```
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.
```

Parameters

-> DM-object of DM-Value

Identifier of the object whose attributes are to be copied to the temporary storage area.

-> DM-attribute of DM-Value

Attribute of the object to be copied to the temporary storage area.

All vectorial attributes of an object are allowed.

-> DM-indexcount of DM-Value

Number of indices to be evaluated in the function call. If the query is a normal attribute vector, *1* must be specified here. However, if the query is a two-dimensional array of attributes in a tablefield, *2* must be specified.

-> DM-long-first of DM-Value

Start index from which the attribute is to be queried. If two indices are necessary for the designation, the line is specified here.

-> DM-long-second of DM-Value

Second start index from which the attribute is to be queried. This parameter is only evaluated if it is an attribute with two indices and *DM-indexcount* has been set to *2*. It then describes the column from which the values are to be queried.

<- DM-POINTER

Identifier of the temporary storage area created for the object's attributes.

-> DM-ENDCOL

End index up to which the attributes of the object are to be queried. If this value = *0*, everything is to be queried from the start index. If this parameter has a value *!= 0*, the corresponding attribute values of the object are queried.

-> DM-ENDROW

Second end index, if it is a two-dimensional attribute. If this value = *0*, everything from the start index is queried. If this parameter has a value *!= 0*, the corresponding attribute values of the object are queried.

<- COUNT

In this parameter the number of requested values is returned.

-> DM-options of DM-Value

This parameter is currently unused.

Return value

DM-Status of DM-StdArgs

DM-error	Values could not be obtained.
DM-success	Values could be obtained successfully.

Example

Set a listbox content with an unknown number of strings and query the content:

```
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.
*Initialization of memory within the DM
     CALL "DMcob_LInitVector" USING DM-StdArgs DLG-ID
             DT-String ICOUNT.
*Initialization of the DM-Value structure
     MOVE DT-STRING TO DM DATATYPE.
     MOVE DLG-STRING TO DM-VALUE-STRING.
*Setting the individual contents
     PERFORM VARYING COUNTER FROM 1 BY 1
     UNTIL COUNTER = DLG-COUNT
         MOVE COUNTER TO DM-LONG-FIRST
         MOVE DLG-STRING TO STR-TAB
*Preparing a modified string for the display
         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.
*Passing the stored values to the display
     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.
*Releasing the memory area
     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.

*Inquire about the individual contents
     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.
*RELEASE OF THE MEMORY
         CALL "DMcob_FreeVector" USING DM-StdArgs
             DM-POINTER.
     GOBACK.
```

6.8.31 DMcob_LGetVectorValue

This function can be used to fetch values from a temporary memory area into the COBOL program.

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

call "DMcob_LGetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameters

-> DM-long-first of DM-Value

With this parameter, the function is told how many values are to be retrieved from the temporary memory area. It must be noted that the first valid value has the index *1*. In addition, the value must not be larger than the temporary memory area.

-> DM-Attribute of DM-Value

This parameter is only observed if the temporary memory area was created with DT-void.

The following attributes are then permitted: *AT-content*, *AT-userdata*, *AT-sensitive* und *AT-active*.

<- DM-Datatype of DM-Value

With this element the data type of the requested attribute is set.

<- DM-Value… of DM-Value

The actual value of the attribute is stored in this structure. It must be noted that only the field corresponding to the data type may be accessed.

-> DM-Options of DM-StdArgs

This parameter is currently unused.

COBOL Interface for MICRO FOCUS VISUAL COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return value

DM-Status of DM-StdArgs

DM-error	Attribute could not be obtained.
DM-success	Attribute could be obtained successfully.

Example

Query the 5th string in a temporary memory area.

```
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

With the help of this function values can be fetched from a temporary memory area into the COBOL program. In contrast to DMcob_GetVectorValue, index values up to 65535 and a data area with up to 65535 characters can be specified with this function.

```
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.
```

Parameters

-> DM-Long-First of DM-Value

With this parameter, the function is told how many values are to be retrieved from the temporary memory area. It must be noted that the first valid value has the index *1*. In addition, the value must not be larger than the temporary memory area.

-> DM-Attribute of DM-Value

This parameter is only observed if the temporary memory area was created with DT-void.

The following attributes are then permitted: *AT-content*, *AT-userdata*, *AT-sensitive* und *AT-active*.

<- DM-Datatype of DM-Value

With this element the data type of the requested attribute is set.

<- DM-Value… of DM-Value

The actual value of the attribute is stored in this structure. It must be noted that only the field corresponding to the data type may be accessed.

-> DM-string

In this parameter you pass the range for a string. The current value is then copied into this range. This string can have any length.

-> DM-stringlen

With this parameter you pass the string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter is currently unused.

COBOL Interface for MICRO FOCUS VISUAL COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-Status of DM-StdArgs

DM-error	Attribute could not be obtained.
DM-suc-cess	Attribute could be obtained successfully.

Examples

Query the 5th string in a temporary memory area.

```
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

This function can be used to generate a temporary memory area in the Dialog Manager in order to store a larger number of attribute values of an object. This memory area should be used especially for objects such as tablefield and listbox for setting contents. In contrast to the DMcob_InitVector function, this function can be used to specify ranges up to 65535.

```
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.
```

Parameters

<- DM-POINTER

With this parameter the newly generated memory area is returned. This value must be specified for all accesses to this area.

-> DM-VALUETYPE

This parameter controls the data type of the attribute values to be stored. The necessary definitions are located in the copy files IDMcobws.cob and IDMcobls.cob. Possible values are e.g. *DT-string*, *DT-boolean*, *DT-instance*, *DT-integer*.

DT-void is an exception. If this value is specified, the space for four attributes (*.content*, *.userdata*, *.sensitive* und *.active*) is created.

-> COUNT

This parameter specifies the number of attributes to be stored internally. This number should correspond as closely as possible to what is really needed. If *0* is specified here, an area with a default size is created.

If the generated area is not sufficient later, it is automatically enlarged by the Dialog Manager.

-> DM-Options of DM-StdArgs

This parameter is currently unused.

Return value

DM-status of DM-StdArgs

DM-error	Area could not be created.
DM-success	Area could be created successfully.

Remark

Memory areas which were created with DMcob_LInitVector must always be released by DMcob_FreeVector!

Example

A range for 15000 string values is to be created.

```
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

With this function, a dialog can be loaded into the DM.

```
77  DM-dialogid  pic 9(4) binary value 0.
01  DM-path      pic X(256).

call "DMcob_LoadDialog" using
            DM-StdArgs
            DM-dialogid
            DM-path.
```

Parameters

<- DM-dialogid

This parameter contains the ID of the loaded dialog. The following values are valid:

0	The indicated file is not an error-free DM file, or the file was not found by the DM.
!= 0	Identifier of the dialog loaded by the DM.

<- DM-path

The file to be loaded is specified in this parameter. It can be a path to this file.

COBOL Interface for MICRO FOCUS VISUAL COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Return Value

DM-status of DM-StdArgs

DM-error	The indicated file is not an error free DM file, or the file was not found by the DM.
DM-succes	Identifier of the dialog loaded by the DM.

The Dialog Manager has the ability to load the dialog data from an ASCII file (dialog script) as well as from a binary file. The main advantage of a binary file is that it can be loaded in a significantly shorter time because no internal checks have to be carried out. A binary file is generated with this command:

```
idm +writebin <binary file name> <ASCII file name>
```

Example

```
    77    buffer     pic x(256) value spaces.
    77    DialogID     pic 9(4) binary value 0.
```

```
move "./cobol.dlg@" to buffer.
call "DMcob_LoadDialog" using DM-StdArgs DialogID buffer.
perform ErrorCheck.
```

6.8.35 DMcob_LoadProfile

With this function variables that can be changed by the end user can be read from a file and processed by the DM. This facility enables end users to influence dialog behavior when the dialog source or the DM are not available.

```
77  DM-dialogid  pic 9(4) binary value 0.
01  DM-path      pic X(256).

call "DMcob_LoadProfile" using
            DM-StdArgs
            DM-dialogid
            DM-path.
```

Parameters

<- DM-dialogid

Name of the dialog to be configured.

<- DM-path

Name of the file to be read. The path should be terminated with a low value or the separator character. Otherwise no more than 256 characters are read.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Return Value

DM-status of DM-StdArgs

DM-error	The indicated file is not an error free DM profile, or the file was not found by the DM.
DM-success	The indicated file was an error free DM profile and was processed successfully.

Example

```
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

With the help of this function you are enabled to change attributes of DM-objects. The difference to DMcob_SetValue is that index values up to 65535 can be specified.

Please refer to the chapter **"Attributes and definitions"** for the permissible attributes for the respective object type.

```
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.
```

Parameters

-> DM-object of DM-Value

This parameter describes the object whose attribute you want to change.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to change. All allowed attributes are defined in the **IDMcobws.cob** file.

-> DM-LONG-FIRST of DM-Value

This parameter is evaluated only for vector attributes of objects. It describes the index of the searched subobject (e.g. text in listbox).

-> DM-LONG-SECOND of DM-Value

This parameter is evaluated only for two-dimensional vector attributes of objects. It describes the second index of the searched subobject (e.g. 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 this parameter the value is passed, which the attribute is to assume. You must make sure that you assign the correct element in this union.

For the data type of each attribute please refer to the chapter “Attributes and Definitions”.

-> DM-string

In this parameter you pass the string you want to set. This string can have any length.

-> DM-stringlen

In this parameter you pass the string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter controls whether the Dialog Manager should trigger rule processing or not by successfully setting the attribute.

- *DMF-Inhibit*
  The parameter must be set to *DMF-Inhibit* if no events are to be sent.
- *DMF-ShipEvent*
  If events are to be sent, this parameter must be assigned with *DMF-ShipEvent*.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-status of DM-StdArgs

DM-error	The attribute could not be set.
DM-success	The attribute could be set successfully.

Example

To change the first field in the tablefield "MyTable", the COBOL program must look something like this:

```
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

With the help of this function you are enabled to change attributes of DM-objects. The difference to DMcob_SetValue is that index values up to 65535 can be specified and that the data size can be up to 65535 characters.

Please refer to the chapter **"Attributes and definitions"** for the permissible attributes for the respective object type.

```
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.
```

Parameters

-> DM-object of DM-Value

This parameter describes the object whose attribute you want to change.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to change. All allowed attributes are defined in the **IDMcobws.cob** file.

-> DM-LONG-FIRST of DM-Value

This parameter is evaluated only for vector attributes of objects. It describes the index of the searched subobject (e.g. text in listbox).

-> DM-LONG-SECOND of DM-Value

This parameter is evaluated only for two-dimensional vector attributes of objects. It describes the second index of the searched subobject (e.g. 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 this parameter the value is passed, which the attribute is to assume. You must make sure that you assign the correct element in this union.

For the data type of each attribute please refer to the chapter “Attributes and Definitions”.

-> DM-string

In this parameter you pass the string you want to set. This string can have any length.

-> DM-stringlen

In this parameter you pass the string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter controls whether the Dialog Manager should trigger rule processing or not by successfully setting the attribute.

- *DMF-Inhibit*
  The parameter must be set to *DMF-Inhibit* if no events are to be sent.
- *DMF-ShipEvent*
  If events are to be sent, this parameter must be assigned with *DMF-ShipEvent*.

COBOL Interface for MICRO FOCUS VISUAL COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-status of DM-StdArgs

DM-error	The attribute could not be set.
DM-success	The attribute could be set successfully.

Example

To change the first field in the tablefield "MyTable", the COBOL program must look something like this:

```
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

This function is used to assign a temporary memory area to an object and an attribute. Areas up to an index value of 65535 can be specified.

```
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.
```

Parameters

-> DM-object of DM-Value

Identifier of the object to which the temporary memory area is to be assigned.

-> DM-attribute of DM-Value

Attribute of the object to which the temporary memory area is to be assigned. All vectorial attributes of an object are allowed.

-> DM-indexcount of DM-Value

Number of indices to be evaluated in the function call. If the assignment is a normal attribute vector, *1* must be specified here. However, if the assignment is a two-dimensional array of attributes in a tablefield, *2* must be specified here.

-> DM-long-first of DM-Value

Start index from which the attribute is to be assigned. If two indices are necessary for designation, the line is specified here.

-> DM-long-second of DM-Value

Second start index from which the attribute is to be assigned. This parameter is only evaluated if it is an attribute with two indices and *DM-indexcount* has been set to *2*. It then describes the column from which the values are to be set.

-> DM-POINTER

Identifier of the temporary memory area to be allocated to the object.

-> DM-ENDCOL

End index up to which the attributes are to be assigned to the object. If this value = *0*, everything from the start index is to be replaced. If this parameter has a value *!= 0*, the corresponding attribute values are replaced for the object. However, the size of the memory area created must then match the number of attributes to be set.

-> DM-ENDROW

Second end index, if it is a two-dimensional attribute. If this value = *0*, everything starting from the start index will be replaced. If this parameter has a value *!= 0*, the corresponding attribute values are set at the object.

-> COUNT

This parameter can be used to control the number of values to be set. If the value = *0*, the memory area as it was created is passed to the Dialog Manager. If the value *!= 0* and is smaller than the memory range, the specified number of values is passed to the Dialog Manager.

-> 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*
With this option no "ActiveVector" is assigned, i.e. the *.active* attribute is ignored in this vector.

» *DMF-OmitSensitive*
With this option no "SensitiveVector" is assigned, i.e. the attribute *.sensitive* is ignored in this vector.

» *DMF-OmitStrings*
With this option no strings are assigned, i.e. the *.content* attribute is ignored in this vector.

Return value

DM-Status of DM-StdArgs

DM-error	Values could not be set.
DM-success	Value could be set successfully.

Example

Set a listbox content with an unknown number of 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.
*Initialization of memory within the DM
    CALL "DMcob_LInitVector" USNG DM-StdArgs DLG-ID
        DT-String ICOUNT.
*Initialization of the DM-Value structure
    MOVE DT-STRING TO DM DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.

*Setting the individual contents
    PERFORM VARYING COUNTER FROM 1 BY 1
            UNTIL COUNTER = DLG-COUNT
        MOVE COUNTER TO DM-LONG-FIRST
        MOVE DLG-STRING TO STR-TAB
*Preparing a modified string for the display
        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.
*Passing the stored values to the display
```

```
    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.
*Release of the memory
    CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.
```

6.8.39 DMcob_LSetVectorValue

This function can be used to set individual attribute values in a temporary memory area. For this purpose, the data type of the values to be set must always match the data type with which the temporary memory area was generated.

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

call "DMcob_LSetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameters

-> DM-Datatype of DM-Value

This parameter tells the function which value in the value structure (DM-Value-...) is to be evaluated for setting the attribute. This data type must absolutely correspond to that of the temporary memory area. If the temporary memory area was created with the data type *DT-void*, the *DM-Datatype* may assume any data type.

-> DM-long-first of DM-Value

This parameter tells the function which value is to be set in the memory area. It must be noted that the first valid value has the index *1*. If a value is specified here that is greater than the initial value, the memory area is automatically increased.

-> DM-attribute of DM-Value

This parameter is only observed if the temporary memory area was generated with *DT-void*. It can then take the following four values:

- *AT-active*
  Datatype must then be *DT-boolean*.
- *AT-sensitive*
  Datatype must then be *DT-boolean*.
- *AT-content*
  Datatype must then be *DT-string*.
- *AT-userdata*
  Data type any, according to the value to be stored in the user data.

-> DM-Value-… of DM-Value

With this parameter the value is passed, which is to be stored. You must make sure that you assign the correct element in this structure.

-> DM-Options of DM-StdArgs

This parameter is currently unused.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return value

DM-Status of DM-StdArgs

DM-error	Attribute value could not be saved.
DM-success	Attribute value was saved successfully.

Example

To fill a memory area with 100 strings, the COBOL program must look something like this:

```
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

This function can be used to set individual attribute values in a temporary memory area. For this purpose, the data type of the values to be set must always match the data type with which the temporary memory area was generated. The difference to the function DMcob_SetVectorValue is that here index values up to 65535 can be addressed and data areas with up to 65530 characters can be passed.

```
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.
```

Parameters

-> DM-Datatype of DM-Value

This parameter tells the function which value in the value structure (DM-Value-...) is to be evaluated for setting the attribute. This data type must absolutely correspond to that of the temporary memory area. If the temporary memory area was created with the data type *DT-void*, the *DM-Datatype* may assume any data type.

-> DM-long-first of DM-Value

This parameter tells the function which value is to be set in the memory area. It must be noted that the first valid value has the index *1*. If a value is specified here that is greater than the initial value, the memory area is automatically increased.

-> DM-attribute of DM-Value

This parameter is only observed if the temporary memory area was generated with *DT-void*. It can then take the following four values:

» *AT-active*
Datatype must then be *DT-boolean*.

» *AT-sensitive*
Datatype must then be *DT-boolean*.

» *AT-content*
Datatype must then be *DT-string*.

» *AT-userdata*
Data type any, according to the value to be stored in the user data.

-> DM-Value-... of DM-Value

With this parameter the value is passed, which is to be stored. You must make sure that you assign the correct element in this structure.

-> DM-Options of DM-StdArgs

This parameter is currently unused.

COBOL Interface for Micro Focus Visual COBOL

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return value

DM-Status of DM-StdArgs

DM-error	Attribute value could not be saved.
DM-success	Attribute value was saved successfully.

Example

To fill a memory area with 100 strings, the COBOL program must look something like this:

```
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

With this function a specified messagebox or dialogbox (window with the attribute *.dialogbox* set to *true*) can be opened. The program pauses until the user has closed this messagebox or dialogbox.

Note

When using functions with records as parameters, please pay attention to chapter “Handling of String Parameters”.

```
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

Currently not used. Please specify with *0*.

-> DM-box

This parameter specifies the messagebox or dialogbox that shall be opened. The identifier can be received as return value of the function DMcob_PathToID.

-> DM-parent

This parameter defines the window int which the messagebox shall appear. If this parameter is specified, a window or NULL has to be specified. The identifier of the window can be received as return value of the function DMcob_PathToID.

The parameter may be ignored. If the window system is capable of it, the messagebox is displayed centered in front of the parent window. Otherwise the position of the messagebox is determined by the window system (e.g. in the middle of the screen).

<-> DM-Value

Contains the return value from the messagebox or dialogbox after that has been closed

- For ***messageboxes*** the parameter contains the number of the button pressed by the user. For this the following definitions exist:
  - *MB-abort*
  - *MB-cancel*
  - *MB-ignore*
  - *MB-no*
  - *MB-ok*
  - *MB-retry*
  - *MB-yes*
- For **dialogboxes** the parameter contains the value defined in the function closequery.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	The messagebox or dialogbox could not be opened.
DM-success	The messagebox or dialogbox has been opened.

See Also

Functions DMcob_OpenBoxBuff, DMcob_QueryBox

Object Messagebox in the “Object Reference”

Built-in function querybox in manual “Rule Language”

6.8.42 DMcob_OpenBoxBuff

With this function a specified messagebox or dialogbox (window with the attribute *.dialogbox* set to *true*) can be opened. The program pauses until the user has closed this messagebox or dialogbox.

The difference to DMcob_OpenBox is, that with this function a buffer for string values can be passed. This buffer can be bigger than the standard buffer.

Note

When using functions with records as parameters, please pay attention to chapter “Handling of String Parameters”.

```
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

Currently not used. Please specify with *0*.

-> DM-box

This parameter specifies the messagebox or dialogbox that shall be opened. The identifier can be received as return value of the function DMcob_PathToID.

-> DM-parent

This parameter defines the window int which the messagebox shall appear. If this parameter is specified, a window or NULL has to be specified. The identifier of the window can be received as return value of the function DMcob_PathToID.

The parameter may be ignored. If the window system is capable of it, the messagebox is displayed centered in front of the parent window. Otherwise the position of the messagebox is determined by the window system (e.g. in the middle of the screen).

<-> DM-Value

Contains the return value from the messagebox or dialogbox after that has been closed

- For ***messageboxes*** the parameter contains the number of the button pressed by the user. For this the following definitions exist:
  - *MB-abort*
  - *MB-cancel*
  - *MB-ignore*
  - *MB-no*
  - *MB-ok*
  - *MB-retry*
  - *MB-yes*
- For **dialogboxes** the parameter contains the value defined in the function closequery.

<- DM-string

This parameter contains the returned string value. It is valid only, when the returned value has the data type *DT-string*.

-> DM-stringlen

This parameter specifies the length of the transferred string. This length must not be greater than the actual length of the passed string.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return Value

DM-status of DM-StdArgs

DM-error	The messagebox or dialogbox could not be opened.
DM-success	The messagebox or dialogbox has been opened.

See Also

Functions DMcob_OpenBox, DMcob_QueryBox

Object Messagebox in the “Object Reference”

Built-in function querybox in manual “Rule Language”

6.8.43 DMcob_PathToID

With this function, the external object name which you already know is transformed into an internal identifier. This internal identifier is not changed during program execution, so you do not have to inquire the ID of a frequently used object on every access.

```
01  DM-objectid  pic 9(4) binary value 0.
01  DM-rootid    pic 9(4) binary value 0.
01  DM-path      pic X(256).

call "DMcob_PathToID" using
            DM-StdArg
            DM-objectid
            DM-rootid
            DM-path.
```

Parameters

<- DM-objectid

0	The object was not found or its identifier is not unique.
!= 0	Identifier of the searched object.

With the thus received identifier you may now access the attributes of the identified object.

-> DM-rootid

This parameter controls from which object the Dialog Manager is to begin searching your desired object. You have the following options:

- *rootid = 0*
  The Dialog Manager searches in the entire dialog definition for the specified object. This is the usual option. The identifiers of rules, functions, variables and resources can also be inquired this way.
- *rootid != 0*
  The Dialog Manager is to search the desired object on the next subordinate hierarchy level from the specified object. This method is appropriate only if an object identifier occurs more than once in a dialog. Rules, functions, variables and resources can**not** be inquired this way.

-> DM-path

This path describes the searched object. The path has to describe an object uniquely. If the object identifier exists only once in the dialog, giving the identifier is sufficient to get the desired reference. If the object identifier is not unique, the object has to be described with a path of object identifiers, separated by periods.

The path should be terminated with a low value or the separator character. Otherwise no more than 256 characters are read.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

Return Value

DM-status

DM-error	The object was not found or its identifier is not unique.
DM-success	The identifier of the searched object is returned.

Note

The identifiers of **all** objects, resource variables, functions and rules can be inquired with this function!

Example

```
Call "DMcob_PathToID" using DM-StdArgs DM-object
        DM-rootID By content "ObjectToAccess".
```

6.8.44 DMcob_PutArgString

With this function, you can replace an argument which was passed on to the program.

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

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

Parameters

-> DM-ArgNumber

This parameter is the number of the argument which should be replaced by this function.

-> DM-Buffer

In this parameter the new value of the argument is stored.

Note

The buffer must be terminated by a **low value** or the **separator character**.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

Return Value

DM-Status

DM-error	The argument could not be replaced successfully. Possible errors are that the inquired argument does not exist or the Dialog Manager does not get enough memory to store the string.
DM-success	The argument could be replaced successfully.

6.8.45 DMcob_QueryBox

With this function a specified messagebox can be opened. The program waits until the user has closed this box.

```
77  DM-boxid     pic 9(4) binary.
77  DM-parentID  pic 9(4) binary.

call "DMcob_QueryBox" using
            DM-StdArgs
            DM-boxID
            DM-parentID.
```

Parameters

-> DM-boxID

This is the identifier of the message to be opened.

-> DM-parentID

Specifies the window in which the messagebox is to appear. This parameter can be ignored.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with 0.

Return Value

DM-status of DM-StdArgs

DM-error	The messagebox could not be opened.
DM-success	The messagebox has been opened.

DM-rescode of DM-StdArgs

Returns the number of the pressed button. There are the following button definitions:

- *MB-abort*
- *MB-cancel*
- *MB-ignore*
- *MB-no*
- *MB-ok*
- *MB-retry*
- *MB-yes*

Example

To open the 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";
}
```

the COBOL program could look like

```
    77    DM-Null-OBject    pic 9(4) binary value 0.
    move "@" to DM-SETSEP.
    call "DMcob_PathToID" using DM-StdArgs DM-object
                        DM-Null-Object,
                        By content "MyMessage@".
    call "DMcob_QueryBox" using DM-StdArgs DM-Object
                        DM-Null-Object.
    if DM-Rescode is equal to MB-ok then
        GoBack.
```

See Also

Functions DMcob_OpenBox, DMcob_OpenBoxBuff

Object Messagebox

Built-in function querybox in manual “Rule Language”

6.8.46 DMcob_QueryError

With the help of this function, the application can query the reason why the last DM call failed. The Dialog Manager returns the number of errors and the errors.

```
call "DMcob_QueryError" using
            DM-StdArgs
            DM-ErrorData.
```

Parameters

<- DM-ErrorData

This is an array of error codes. It is filled by the Dialog Manager. If the array is not large enough the last errors are ignored. To be sure that all error codes stored in the Dialog Manager can be passed to the application, this array should have the size of 32.

-> .DM-error-size of DM-ErrorData

This is the number of error codes which should be returned by this function.

<-> DM-error-count of DM-ErrorData

This is the real number of errors occurred which are filled into the error buffer.

<-> DM-Error-Detail of DM-ErrorData

This contains the array of the error codes in which the error code, the severity and the module are to be filled.

Return Value

DM-status of DM-StdArgs

DM-success	The function returns the error codes.

Example

```
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

Execution of a rule that depends on an external event is "put down" by the application with the interface function **DMcob_QueueExtEvent**. "Put down" means that the rule is not executed immediately, but that the external event is queued and then processed depending on the dialog event mechanisms. The event is processed by the DM with the usual event processing algorithm. The rule follows the scheme "on <object> extevent <number>".

```
77  DM-eventno   pic 9(4) binary.
77  DM-objectID  pic 9(4) binary.

call "DMcob_QueueExtEvent" using
            DM-StdArgs
            DM-objectID
            DM-eventno
            DM-ValueArray.
```

Parameters

-> DM-objectID

This is the identifier of the object, to which this external event shall be sent.

-> DM-eventno

This is the number of the external event to be triggered.

<-> DM-ValueArray

This parameter is an array of values which are to be used as parameters for the rule call. The length of this vector may be up to *16*. The structure element has to be filled depending on the corresponding parameter data type.

-> DM-count of DM-ValueArray

In this parameter the number of rule parameters has to be indicated. This number has to be equal to the declaration of the rule inside the dialog script, otherwise the Dialog Manager does not call the rule.

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

In this parameter the data type of the "index" parameter is transferred to the Dialog Manager. Depending on this value, the structure element has to be specified.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0.*

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-va-value-string*, it is also possible to use *DM-va-value-string-u* when working with Unicode texts (UTF-16).

DM-va-value-string-getlen and *DM-va-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	Event could not be stored.
DM-succes	Event was successfully stored.

Example

To call the following rule from COBOL

```
on MyObj extevent 4711 (string Arg1 input output, integer Arg2 input)
{
  print this;
  print Arg1;
  print Arg2;
  Arg1 := "New valid string";
}
```

the COBOL program could look like

```
     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

With this function you can reset attributes of DM objects at the value of the respective model or default. For the permitted attributes of the respective object type, please refer to the charts in chapter “Attributes and Definitions”.

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

Parameters

-> DM-object of DM-Value

This parameter describes the object whose attribute you want to reset. You have received this identifier as return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to reset. All permitted attributes are defined in the **IDMcobws.cob** file.

->DM-Index of DM-Value

This parameter is analyzed only in vector attributes of objects and describes the index of the searched sub-object (e.g. text in listbox).

-> DM-Options of DM-StdArgs

This parameter controls whether the DM is to trigger rule processing after an attribute was set successfully.

- *DMF-Inhibit*
  If no events are to be sent, this parameter has to be set with *DMF-Inhibit*.
- *DMF-ShipEvent*
  If there are events to be sent, the parameter has to be *DMF-ShipEvent*.

Return Value

DM-status of DM-StdArgs

DM-error	The attribute could not be reset.
DM-success	The attribute was reset successfully.

6.8.49 DMcob_SetValue

With this function you can change attributes of DM objects. For the permitted attributes of the respective object types, please refer to chapter "Attributes and Definitions".

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

Parameters

-> DM-object of DM-Value

This parameter describes the object whose attributes you want to change. You have received this identifier as the return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to change. All permitted attributes are defined in the **IDMcobws.cob** file.

-> DM-Index of DM-Value

This parameter is analyzed only in vector attributes of objects. It describes the index of the searched sub-object (e.g. text in listbox).

-> DM-Indexcount of DM-Value

This parameter indicates how many index values are to be noticed when calling the function:

- 2-dimensional attributes -> value = 2 has to be set
- 1-dimensional attributes -> value = 1 has to be set
- non-indexed attributes -> value = 0 has to be set

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

This parameter indicates how long the string to be set can be (0 = Dialog Manager has to search for the fill character).

-> 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 one of these values you can pass the value the attribute has to have. You should take care to set the appropriate element in this structure and the correct data type. For the data type of each attribute, please refer to chapter "Attributes and Definitions".

-> DM-Options of DM-StdArgs

This parameter controls whether the DM is to trigger rule processing after an attribute was set successfully.

- *DMF-Inhibit*
  If no events are to be sent, this parameter has to be specified by *DMF-Inhibit*.
- *DMF-ShipEvent*
  If there are events to be sent, it has to be specified by *DMF-ShipEvent*.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-status of DM-StdArgs

DM-error	The attribute could not be set.
DM-success	The attribute was set successfully.

Example

To change the title of a window named "TestWindow" the COBOL program is to look like this:

```
move 0 to DM-indexcount.
move AT-title to DM-Attribute.
move DT-string to DM-Datatype.
move DMF-Inhibit to DM-Options.
Call "DMcob_SetValue" using DM-StdArgs DM-Value.
```

6.8.50 DMcob_SetValueBuff

With this function you can change attributes of DM objects. The difference to DMcob_SetValue is that you can pass your own buffer to this function. This is necessary, if you want to set a string longer than *80* characters. For the permitted attributes of the respective object types, please refer to chapter “Attributes and Definitions”.

```
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.
```

Parameters

-> DM-object of DM-Value

This parameter describes the object whose attributes you want to change. You have received this identifier as return value from the DMcob_PathToID function.

-> DM-Attribute of DM-Value

This parameter describes the object attribute you want to change. All permitted attributes are defined in the **IDMcobws.cob** file.

-> DM-Index of DM-Value

This parameter is analyzed only in vector attributes of objects. It describes the index of the searched sub-object (e.g. 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 one of these values you pass the value that the attribute is to take. Please make sure to set the appropriate element of this union and the correct data type. For the data type of each attribute, please refer to chapter “Attributes and Definitions”.

-> DM-string

In this parameter you pass the string you want to set. This string can have any length you want.

-> DM-stringlen

In this parameter you pass the string length of the passed string.

-> DM-Options of DM-StdArgs

This parameter controls whether the DM is to trigger rule processing or not after an attribute was set successfully.

- *DMF-Inhibit*
  If no events are to be sent, this parameter has to be specified by *DMF-Inhibit*.
- *DMF-ShipEvent*
  If there are events to be sent, it has to be specified by *DMF-ShipEvent*.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

The parameter *DM-string* may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

In the *DM-stringlen* parameter, still the defined size has to be specified, for example *33* for *PIC N(33)*.

Return Value

DM-status of DM-StdArgs

DM-error	The attribute could not be set.
DM-success	The attribute was set successfully.

Example

To change the content (first field) of a tablefied named "MyTable" with large entries, the COBOL program should look like this:

```
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

With this function you can allocate a temporary memory to an object and an attribute.

```
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.
```

Parameters

-> DM-object of DM-Value

Identifier of the object to which the temporary memory is to be allocated.

-> DM-attribute of DM-Value

Object attribute to which the temporary memory is to be allocated.

All vector attributes of an object are permitted.

-> DM-indexcount of DM-Value

Number of indexes which are to be evaluated at the function call.

You have to indicate *1*, if it is a normal attribute vector. If the field is two-dimensional (attributes of tablefield), you have to indicate *2*.

-> DM-index of DM-Value

Start index from which on the attribute is to be allocated. If two indexes are necessary for the labeling, you have to indicate the row instead.

-> DM-second of DM-Value

Second start index from which on the attribute is to be allocated. This parameter is only evaluated if it is an attribute with two indexes and if *DM-indexcount* is set at *2*. It describes the column from which on the values are to be queried.

<- DM-POINTER

Identifier of the temporary memory that is to be allocated to the object.

-> DM-ENDCOL

End index up to which the object attributes are to be allocated. Value *0* means that everything from the start index on is to be queried. Value *! = 0* means that the respective attribute values at the object are queried. Then the size of the created memory has to correspond with the number of the attributes to be set.

-> DM-ENDROW

Second end index, if it is a two-dimensional attribute. If this value = *0*, everything from the start index onwards is substituted. If this parameter is *!= 0*, the relevant attribute values are set at the object.

-> COUNT

Controls the number of the values to be set. Value *0* means that the memory is given to the Dialog Manager as it was created. If the value is *!= 0* and smaller than the memory, the given number of values is passed to the Dialog Manager.

-> DM-options of DM-Value

Options to be used for the call.

- *DMF-OmitActive*
  No "ActiveVector" is assigned, the attribute *.active* in this vector will be ignored.
- *DMF-OmitSensitive*
  No "SensitiveVector" is assigned, the attribute *.sensitive* in this vector will be ignored.
- *DMF-OmitStrings*
  With this option no strings are assigned, the attribute *.content* will be ignored in this vector.

Return Value

DM-Status of DM-StdArgs

DM-error	Values could not be set.
DM-success	Values could be set successfully.

Example

Setting of listbox contents with an unknown number of 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(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(4) 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.
*Initialization of memory in DM
    CALL "DMcob_InitVector" USNG DM-StdArgs DLG-ID DT-String
        ICOUNT.
*Initialization of DM-Value structure
    MOVE DT-STRING TO DM DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.
*Setting of individual contents
    PERFORM VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER = DLG-COUNT
        MOVE COUNTER TO DM-INDEX
        MOVE DLG-STRING TO STR-TAB
*Preparation of a changed string for display
        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.
*Transfer of the stored values to the display
    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.
*Freeing of the memory
    CALL "DMcob_FreeVector" USING DM-StdArgs DLG-ID.
```

6.8.52 DMcob_SetVectorValue

You can set single attribute values in a temporary memory with this function. Thus the data type of the values to be set has to correspond with the data type by which the temporary memory was created.

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

call "DMcob_SetVectorValue" using
            DM-StdArgs
            DM-Value
            DM-POINTER.
```

Parameters

-> DM-Datatype of DM-Value

Provides the function with the values in the value structure (DM-Value-...) to be evaluated for setting the attribute.

This data type has to correspond absolutely with the temporary memory's data type. If the temporary memory was created with *DT-void*, *DM-datatype* may take any data type.

-> DM-index of DM-Value

Notifies the function of the value which is to be set in the memory. Note that the first valid value has the index = *1*. If a value is given here which is greater than the initial value, the memory is automatically resized.

-> DM-attribute of DM-Value

This parameter is only considered if the temporary memory was generated with *DT-void*. Thus it can take the following values:

» AT-active (data type has to be *DT-boolean*)

» AT-sensitive (data type has to be *DT-boolean*)

» AT-content (data type has to be *DT-string*)

» AT-userdata (any data type, corresponding to the value which shall be stored in the userdata))

-> DM-Value-… of DM-Value

The value that is to be stored is given in this parameter. Note that you use the correct element in this structure.

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

COBOL Interface for Micro Focus Visual COBOL

Instead of *DM-value-string*, it is also possible to use *DM-value-string-u* when working with Unicode texts (UTF-16).

DM-value-string-getlen and *DM-value-string-putlen* continue to indicate the number of characters and not the number of bytes.

Return Value

DM-error	Attribute value could not be stored.
DM-success	Attribute value could be stored successfully.

Example

The COBOL program has to look as follows to fill a memory with 100 strings.

```
77    DM-POINTER        pic 9(4) binary.
77    COUNTER        pic 9(4) binary.
CALL "DMcob_InitVector" using     DM-StdArgs DM-POINTER
                        DM-String 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

This function shuts down the DM. When the function is called all global initialization procedures are recalled. It is usually called by the same function that calls COBOLMAIN in the application. Therefore, **DMcob_ShutDown** may only be called if the “Main” program in the DM is replaced by an application-specific one.

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

Parameters

-> DM-Options of DM-StdArgs

Currently not used. Please specify with *0*.

Return value

None.

6.8.54 DMcob_StartDialog

The actual dialog application is started by means of this function. The DM creates all necessary resources (colors, cursor, fonts etc.) in the window system, puts all top level objects defined as visible in the dialog on the screen, and executes the start rule.

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

call "DMcob_StartDialog" using
            DM-StdArgs
            DM-dialogid.
```

Parameters

-> DM-dialogid

This is the identifier of the dialog to be started. You have received this identifier from the DMcob_ LoadDialog function.

-> DM-Options

Currently not used. Please specify with *0*.

Example

```
call "DMcob_StartDialog" using DM-StdArgs DialogID.
perform ErrorCheck.
```

See Also

Built-in function run in manual “Rule Language”

6.8.55 DMcob_StopDialog

This function terminates a dialog. If that dialog was the last running dialog the event loop is left. If no dialog is specified the current dialog is stopped. If the parameter *DM-Options* has the value *DMF-ForceDestroy* the dialog is deleted after its *finish* rule has been executed.

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

call "DMcob_StopDialog" using
            DM-StdArgs
            DM-dialogid.
```

Parameters

-> DM-dialogid

Identifier of the dialog to be stopped. You have received this identifier as return value from DMcob_LoadDialog.

-> DM-Options

This parameter decides whether the stopped dialog is to be deleted or not. If the dialog should stay in memory, please give here *0*. Otherwise, if the value is set at *DMF-ForceDestroy* the dialog is removed out of the memory.

Example

```
    move DMF-ForceDestroy to DM-Options.
    call "DMcob_StopDialog" using DM-StdArgs DialogID.
    perform ErrorCheck.
```

See Also

Built-in function stop in manual “Rule Language”

6.8.56 DMcob_TraceMessage

With this function, trace messages from the application can be written into the tracefile of the Dialog Manager.

```
01  DM-string  pic X(99).

call "DMcob_TraceMessage" using
            DM-StdArgs
            DM-string.
```

Parameters

-> DM-string

This string is written into the tracefile. The message should be terminated with a low value or the separator character. Otherwise no more than 256 characters are read.

COBOL Interface for Micro Focus Visual COBOL

The parameter may also be passed as *National Character* (*PIC N*) when working with Unicode texts (UTF-16).

-> DM-Options of DM-StdArgs

With this parameter the application can control whether the Dialog Manager writes the header "[UM]" at the beginning of the line or not. If the parameter is set at *DMF-InhibitTag* the header will not be printed.

Note

The traces are only printed, if the Dialog Manager is started with the trace option.

Example

```
    move "Now we have reached the cobol function " to DM-string.
    call "DMcob_TraceMessage" using DM-StdArgs DM-string.
```

6.8.57 DMcob_ValueChange

With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection.

If the *AnyValue* parameter is a collection, e.g. of type *DT-vector*, *DT-list*, *DT-hash*, *DT-matrix* or *DT-refvec*, an element value can be substituted by specifying the *DM-ValueIndex* parameter. Similar to predefined attributes, a collection can be extended by incrementing the index with *+1*. For associative arrays, simply a not yet assigned index key may be used.

If a collection in the *DM-Value*parameter is assigned to the target as a whole (i.e. with *NULL* as *DM-ValueIndex* parameter), the entire value with all value elements is copied. The conversion of an argument into a locally managed value also requires a complete copying to allow further manipulation.

```
77 AnyValue pointer value null.

call "DMcob_ValueChange" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-AppendValue	For collections, the data value from *DM-Value* is appended at the end. The index must be *NULL* for this.
DMF-SortBinary	In collections, the newly set value is finally sorted. May be used in combination with *DMF-SortReverse*.
DMF-SortLinguistic	In collections, the newly set value is finally sorted, for strings according to linguistic rules (see the built-in function sort). May be used in combination with *DMF-SortReverse*.
DMF-SortReverse	In collections, the newly set value is finally sorted in reverse order.

-> AnyValue

Handle of the Managed Value to be changed.

-> DM-ValueIndex

This parameter can be used to change element values in collections and specifies the index. Otherwise, *DM-idx-datatype* should be set to *DT-void*. This parameter does not need to be a managed value reference.

-> DM-Value

This parameter defines the value to be set. It may be a managed or an unmanaged value reference.

Return value

DM-status of DM-StdArgs

DM-error	Value could not be set. This may be due to an faulty call, an unmanaged or invalid value reference, or an incorrect indexing.
DM-success	The function has been completed successfully, setting the value succeeded or the value had already been set.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Function DMcob_ValueChangeBuffer

Chapter “Using the anyvalue Data Type”

6.8.58 DMcob_ValueChangeBuffer

With this function a value reference managed by IDM may be manipulated. Either the entire value can be replaced or a single element value in a collection. Unlike DMcob_ValueChange, this function provides a buffer for string return values that is larger than the standard buffer.

If the *AnyValue* parameter is a collection, e.g. of type *DT-vector*, *DT-list*, *DT-hash*, *DT-matrix* or *DT-refvec*, an element value can be substituted by specifying the *DM-ValueIndex* parameter. Similar to predefined attributes, a collection can be extended by incrementing the index with *+1*. For associative arrays, simply a not yet assigned index key may be used.

If a collection in the *DM-Value*parameter is assigned to the target as a whole (i.e. with *NULL* as *DM-ValueIndex* parameter), the entire value with all value elements is copied. The conversion of an argument into a locally managed value also requires a complete copying to allow further manipulation.

```
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.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-AppendValue	For collections, the data value from *DM-Value* is appended at the end. The index must be *NULL* for this.
DMF-SortBinary	In collections, the newly set value is finally sorted. May be used in combination with *DMF-SortReverse*.
DMF-SortLinguistic	In collections, the newly set value is finally sorted, for strings according to linguistic rules (see the built-in function sort). May be used in combination with *DMF-SortReverse*.
DMF-SortReverse	In collections, the newly set value is finally sorted in reverse order.

-> AnyValue

Handle of the Managed Value to be changed.

-> DM-ValueIndex

This parameter can be used to change element values in collections and specifies the index. Otherwise, *DM-idx-datatype* should be set to *DT-void*. This parameter does not need to be a managed value reference.

-> DM-Value

This parameter defines the value to be set. It may be a managed or an unmanaged value reference.

<- StringBuffer

Buffer for the string, to modify strings longer than *80* characters.

-> StringLength

Length of the buffer (*100* here), must match the definition.

Return value

DM-status of DM-StdArgs

DM-error	Value could not be set. This may be due to an faulty call, an unmanaged or invalid value reference, or an incorrect indexing.
DM-success	The function has been completed successfully, setting the value succeeded or the value had already been set.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Function DMcob_ValueChange

Chapter “Using the anyvalue Data Type”

6.8.59 DMcob_ValueCount

Returns the number of values in a collection (without the default values). It is also possible to return the index type or the highest index value.

The returned value indicates the number of values (without the default values). Thus, in combination with the DMcob_ValueIndex function, loops over all indexed values respectively elements can be implemented easily.

```
77 AnyValue pointer  value  null.
77 Count    pic 9(9) binary value 0.

call "DMcob_ValueCount" using
            DM-StdArgs
            AnyValue
            DM-Value
            Count.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-GetLocalString	This option means that text values (IDs of type *DT-text*) should be returned as strings in the currently set language.
DMF-GetMasterString	This option means that text values (IDs of type *DT-text)* should be returned as a strings in the development language, regardless of which language the user is currently working with.
DMF-DontFreeLastStrings	Strings are usually passed to the application in a temporary buffer, which is retained until the next call to the IDM. If strings in the application shall be valid longer, the option *DMF-DontFreeLastStrings* has to be set. Then the memory will not be released until an IDM function returning a string from the IDM to the applicationis called without this option.

-> AnyValue

Handle of the Managed Value from which the number of values is fetched. It should be a managed value reference or function argument.

<- DM-Value

Here the count value is returned. This may be a managed value reference, however this is not mandatory.

Depending on the *AnyValue* parameter, the following results may occur:

AnyValue Type	Count Return Type	Remark
DT-refvec, *DT-list*, *DT-vector*	*DT-integer*	Highest index value
DT-matrix	*DT-index*	Highest index value
DT-hash	*DT-datatype*	Any index type (*anyvalue*)
otherwise	*DT-void*	Non-indexed value

<- Count

0 … INT_MAX	Number of values (excluding the default values with the indexes *[0]*, *[0,*]* or *[*,0]*).
DM-Value	Highest index value, may be either *void* (scalar value), an *integer* value (one-dimensional array), an *index* value (two-dimensional array), or a data type (associative array).

Return value

DM-status of DM-StdArgs

DM-error	The number of values could not be determined. This may be due to a faulty call or an unmanaged or invalid value reference.
DM-success	Querying the number of values has been successful.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Functions DMcob_ValueChange, DMcob_ValueGet, DMcob_ValueIndex

Chapter “Using the anyvalue Data Type”

Built-in functions countof, itemcount

6.8.60 DMcob_ValueGet

This function allows to retrieve a single element value that belongs to a defined index from collections.

If the given index is of type *DT-void*, the entire value is returned, which usually means copying the value.

If the *DM-Value* parameter is an unmanaged value, it remains unmanaged. When strings are returned, they are only stored in a temporary buffer which may be cleared or overwritten the next time a DM function without the *DMF-DontFreeLastStrings* option is called.

```
77 AnyValue pointer value null.

call "DMcob_ValueGet" using
            DM-StdArgs
            AnyValue
            DM-ValueIndex
            DM-Value.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-GetLocalString	This option means that text values (IDs of type *DT-text*) should be returned as strings in the currently set language.
DMF-GetMasterString	This option means that text values (IDs of type *DT-text)* should be returned as a strings in the development language, regardless of which language the user is currently working with.
DMF-DontFreeLastStrings	Strings are usually passed to the application in a temporary buffer, which is retained until the next call to the IDM. If strings in the application shall be valid longer, the option *DMF-DontFreeLastStrings* has to be set. Then the memory will not be released until an IDM function returning a string from the IDM to the applicationis called without this option.

-> AnyValue

Handle of the Managed Value. It should be a managed value reference or function argument.

-> DM-ValueIndex

This parameter sets the index for which the element value is retrieved. This parameter does not need to be a managed value. If *DM-idx-datatype* is set to the value *DT-void*, then the *DM-Value* parameter is returned.

<- DM-Value

This parameter defines the value to be set. It may be a managed or an unmanaged value reference.

Return value

DM-status of DM-StdArgs

DM-error	The value could not be retrieved. This may be due to an faulty call, an unmanaged or invalid value reference, or an incorrect indexing.
DM-success	Getting the value has been successful.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Functions DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGetBuffer, DMcob_ValueIndex

Chapter “Using the anyvalue Data Type”

6.8.61 DMcob_ValueGetBuffer

This function allows to retrieve a single element value that belongs to a defined index from collections. Unlike DMcob_ValueGet, this function provides a buffer for string return values that is larger than the standard buffer.

If the given index is of type *DT-void*, the entire value is returned, which usually means copying the value.

If the *DM-Value* parameter is an unmanaged value, it remains unmanaged. When strings are returned, they are only stored in a temporary buffer which may be cleared or overwritten the next time a DM function without the *DMF-DontFreeLastStrings* option is called.

```
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.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-GetLocalString	This option means that text values (IDs of type *DT-text*) should be returned as strings in the currently set language.
DMF-GetMasterString	This option means that text values (IDs of type *DT-text)* should be returned as a strings in the development language, regardless of which language the user is currently working with.
DMF-DontFreeLastStrings	Strings are usually passed to the application in a temporary buffer, which is retained until the next call to the IDM. If strings in the application shall be valid longer, the option *DMF-DontFreeLastStrings* has to be set. Then the memory will not be released until an IDM function returning a string from the IDM to the applicationis called without this option.

-> AnyValue

Handle of the Managed Value. It should be a managed value reference or function argument.

-> DM-ValueIndex

This parameter sets the index for which the element value is retrieved. This parameter does not need to be a managed value. If *DM-idx-datatype* is set to the value *DT-void*, then the *DM-Value* parameter is returned.

<- DM-Value

This parameter defines the value to be set. It may be a managed or an unmanaged value reference.

<- StringBuffer

Buffer for the string, to obtain strings longer than *80* characters.

-> StringLength

Length of the buffer (*100* here), must match the definition.

Return value

DM-status of DM-StdArgs

DM-error	The value could not be retrieved. This may be due to an faulty call, an unmanaged or invalid value reference, or an incorrect indexing.
DM-success	Getting the value has been successful.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Functions DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGet, DMcob_ValueIndex

Chapter “Using the anyvalue Data Type”

6.8.62 DMcob_ValueGetType

This function queries the data type of a value managed by the IDM.

```
77 AnyValue pointer  value  null.
77 Type     pic 9(4) binary value 0.

call "DMcob_ValueType" using
            DM-StdArgs
            AnyValue
            Type.
```

Parameters

<-> DM-options of DM-StdArgs

Unused, has to be *0*.

-> AnyValue

Handle of the Managed Value.

<- Type

Data type of the Managed Value.

Return value

DM-status of DM-StdArgs

DM-error	The data type could not be determined. This may be due to a faulty call or an unmanaged or invalid value reference.
DM-success	Querying the data type has been successful.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Chapter “Using the anyvalue Data Type”

6.8.63 DMcob_ValueIndex

This function can be used to determine the corresponding index for a position in a collection. This is especially important for values of type *DT-hash* and *DT-matrix* in order to access all respective indexes the easiest way. The function only allows access to indexes that do not belong to default values.

The returned index can be stored in a managed *DM-Value* parameter value or in an unmanaged one. In the latter case, the string value is allocated in the temporary buffer if necessary.

```
77 AnyValue pointer  value  null.
77 IndexPos pic 9(9) binary value 0.

call “DMcob_ValueIndex” using
            DM-StdArgs
            AnyValue
            IndexPos
            DM-Value.
```

Parameters

<-> DM-options of DM-StdArgs

These are the options available:

Option	Meaning
DMF-GetLocalString	This option means that text values (IDs of type *DT-text*) should be returned as strings in the currently set language.
DMF-GetMasterString	This option means that text values (IDs of type *DT-text)* should be returned as a strings in the development language, regardless of which language the user is currently working with.
DMF-DontFreeLastStrings	Strings are usually passed to the application in a temporary buffer, which is retained until the next call to the IDM. If strings in the application shall be valid longer, the option *DMF-DontFreeLastStrings* has to be set. Then the memory will not be released until an IDM function returning a string from the IDM to the applicationis called without this option.

-> AnyValue

Handle of the Managed Value from which the index for the position will be retrieved.

-> IndexPos

This parameter defines the position of the index and should be in the range *0* < *IndexPos* <= *DMcob_ValueCount()*.

<- DM-Value

The respective index is stored here. This may be a managed or an unmanaged value reference.

Return value

DM-status of DM-StdArgs

DM-error	The index could not be determined. This may be due to an faulty call, an unmanaged or invalid value reference, or an incorrect position.
DM-success	The value has an index at this position; the obtained index can afterward be found in *DM-Value* if *DM-Value != NULL*.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Functions DMcob_ValueChange, DMcob_ValueCount, DMcob_ValueGet

Chapter "Using the anyvalue Data Type"

Method index

Built-in functions countof, itemcount

6.8.64 DMcob_ValueInit

With this function a value can be converted into a local or global value reference managed by the IDM. This allows the further manipulation of the value by **DMcob_Value*** functions and its transfer as parameter or return value.

The value reference is initialized with the appropriate type. The collection data types *DT-list*, *DT-vector*, *DT-hash*, *DT-matrix* and *DT-refvec* are also permitted.

If the value reference is initialized as static or global via the *DMF-StaticValue*option, access is also possible outside the function call. Value lists and strings are not released at the end of the function. The initialization of arguments as static or global managed value references is not allowed.

Collections are created without element values. All other value types are also initialized with a *0* value.

The function DMcob_ValueChange can be used to add or change values or part values respectively elements.

```
77 AnyValue pointer  value  null.
77 Type     pic 9(4) binary value 0.

call "DMcob_ValueInit" using
            DM-StdArgs
            AnyValue
            Type
            DM-Value.
```

Parameters

<-> DM-options of DM-StdArgs

Option	Meaning
0	The value reference will be initialized as a local value.
DMF-StaticValue	The value reference will be initialized as a global, static value.

<- AnyValue

Handle of the newly created Managed Value.

-> Type

This parameter specifies the requested initial type.

-> DM-indexcount of DM-Value

In this parameter the initial size of collections like *list* or *matrix* can be specified or the appropriate value type for *vector* values.

Return value

DM-status of DM-StdArgs

DM-error	The value reference could not be initialized.
DM-success	The function has been completed successfully so the value reference is initialized.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

See also

Chapter “Using the anyvalue Data Type”

6.8.65 DMmfviscob_BindThruLoader

With this function functions can be addressed via the COBOL runtime system. Thus, they do not have to be entered in the functions table, but are hold as separate files in the current directory or in a directory that can be addressed via the environment variable `COBLIB`. If the function name and the file name correspond, the COBOL runtime system can find the function and call it. To initialize this entire process, you have to call **DMmfviscob_BindThruLoader**.

```
call "DMmfviscob_BindThruLoader" using
        DM-StdArgs
        DM-dialogid.
```

Parameter

-> DM-Options of DM-StdArgs

- *DMF-Silent*
  No error messages shall be sent to the user.
  Otherwise error messages are subdivided into “superfluous functions” and “missing functions”.
- *DMF-Verbose*
  Normal error messages shall be sent to the user.
  Otherwise error messages are divided into “superfluous functions” and “missing functions”.

-> DM-dialogid

This is the identifier of the dialog or the application.

You have received the dialog identifier as return value from DMcob_LoadDialog.

You have received the application identifier as return value from the parameter *DM-APPL-ID* of the function COBOLMAIN.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

This function is available on UNIX and MICROSOFT WINDOWS.

6.8.66 DMufcob_BindThruLoader

With this function functions can be addressed via the COBOL runtime system. Thus, they do not have to be entered in the functions table, but are hold as separate files in the current directory or in a directory that can be addressed via the environment variable `COBLIB`. If the function name and the file name correspond, the COBOL runtime system can find the function and call it. To initialize this entire process, you have to call **DMufcob_BindThruLoader**.

```
call "DMufcob_BindThruLoader" using
        DM-StdArgs
        DM-dialogid.
```

Parameter

-> DM-Options of DM-StdArgs

- *DMF-Silent*
  No error messages shall be sent to the user.
  Otherwise error messages are subdivided into “superfluous functions” and “missing functions”.
- *DMF-Verbose*
  Normal error messages shall be sent to the user.
  Otherwise error messages are divided into “superfluous functions” and “missing functions”.

-> DM-dialogid

This is the identifier of the dialog or the application.

You have received the dialog identifier as return value from DMcob_LoadDialog.

You have received the application identifier as return value from the parameter *DM-APPL-ID* of the function COBOLMAIN.

Availability

Only MICRO FOCUS COBOL on UNIX

6.9 Options of the Interface Functions

In the following table all options are listed which can be specified in the parameter *options* of the DM interface functions. Usually they can be connected by a logical “or” so that a function with more than one valid option can be called.

Option	Function	Description
DMF-AcceptChild	DMcob_GetValue, DMcob_SetValue	Is valid for attribute *AT-options* of canvas under Motif. A canvas is marked which can have children.

Option	Function	Description
DMF-AppendValue	DMcob_ValueChange DMcob_ ValueChangeBuffer	Append a data value to a collection if *Index = NULL*.
DMF-BindForLoader	DMcob_Control	Enables the BindThruLoader functionality that allows the COBOL runtime system to call application functions.
DMF-CreateInvisible	DMcob_CreateFromModel DMcob_CreateObject	The newly generated object shall be generated invisibly; independently of how the attribute *AT-visible* is set in the copy model.
DMF-CreateModel	DMcob_CreateFromModel DMcob_CreateObject	The object to be generated anew shall be generated as a model.
DMF-DontFreeLastStrings	DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	Temporary buffer for strings is preserved beyond the next call of the IDM.
DMF-DontTrace	DMcob_QueueExtEvent	Function call shall not be logged in the tracefile.
DMF-DontWait	DMcob_EventLoop	The event processing shall not be carried out “blockingly”, i.e. if there is an event, it shall be processed; if there is no event, the calling function shall be entered again.
DMF-ForceDestroy	DMcob_Destroy DMcob_StopDialog	This option shows that the indicated object shall be deleted. On DMcob_Destroy this option has to be chosen, if the object shall really be deleted.

Option	Function	Description
DMF-GetLocalString	DMcob_GetValue DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	The text shall be returned as a string in the currently specified language.
DMF-GetMasterString	DMcob_GetValue DMcob_ValueCount DMcob_ValueGet DMcob_ValueGetBuffer DMcob_ValueIndex	The text shall be returned as a string in the original language.
DMF-GetTextID	DMcob_GetValue	The text shall be returned as tex-tid.
DMF-IncludeIdent	DMcob_ErrMsgText	The name of the part which has produced the error shall be con-tained in the error message. This part may be either the operation system, the window system or the DM.
DMF-IncludeModule	DMcob_ErrMsgText	The name of the module in which the error has occurred shall be included in the error message.
DMF-IncludeSeverity	DMcob_ErrMsgText	The degree of the error (warning, error, fatal error) shall be con-tained in the error text.
DMF-IncludeText	DMcob_ErrMsgText	The actual error text shall be con-tained in the error message.
DMF-InheritFromModel	DMcob_CreateObject	The object including the children indicated at the model shall be generated.
DMF-Inhibit	DMcob_SetValue DMcob_SetVector	For setting the attribute value no event shall be created, thus no rule *on object attribute changed* shall be triggered.
DMF-InhibitTag	DMcob_TraceMessage	On tracing the header *“[UM]”* shall not be printed.

Option	Function	Description
DMF-LogFile	DMcob_TraceMessage	Output is not in tracefile, but in logfile.
DMF-OmitActive	DMcob_GetVector DMcob_SetVector	The attribute *AT-active* shall not be transferred.
DMF-OmitSensitive	DMcob_GetVector DMcob_SetVector	The attribute *AT-sensitive* shall not be transferred.
DMF-OmitStrings	DMcob_GetVector DMcob_SetVector	The strings shall not be trans-ferred.
DMF-OmitUserData	DMcob_GetVector	The attribute *AT-userdata* shall not be allocated.
DMF-SetCodePage	DMcob_Control	The indicated code page shall be used in the application and all texts shall be converted into this code page.
DMF-ShipEvent	DMcob_SetValue DMcob_SetVectorValue	For setting the attribute an event shall be created and the possibly existing rule *on .attribute changed* shall be triggered.
DMF-SortBinary	DMcob_ValueChange DMcob_ ValueChangeBuffer	Binary sorting of a collection.
DMF-SortLinguistic	DMcob_ValueChange DMcob_ ValueChangeBuffer	Sorting a collection based on lin-guistic rules.
DMF-SortReverse	DMcob_ValueChange DMcob_ ValueChangeBuffer	Sorting a collection in reverse order.
DMF-StaticValue	DMcob_ValueInit	Creating a static or global man-aged value reference.
DMF-Synchronous	DMcob_QueueExtEvent	Can be set to optimize the internal processes, if DMcob_ QueueExtEvent is not called from a "Signal Handler" function.

Option	Function	Description
DMF-UpdateScreen	DMcob_Control	All actions carried out internally shall be made visible on the screen.
DMF-UseUserData	DMcob_SetContent	In the DM-Content vector the userdata shall be considered and allocated to the object.
DMF-Verbose	DMcob_DataChanged	Activates tracing of the function.
DMF-XlateString	DMcob_SetValue	The indicated string shall be translated into the currently specified language before it is allocated to the object. This does only work if the text is present already internally and if a translation exists.

7 Attributes and Definitions

In order to make accesses from the application to the DM possible, a number of symbolic names are made available in the file **IDMcobws.cob**. These names are explained in the following.

7.1 Attribute Definitions and their Data Types

A number of definitions for the access functions DMcob_GetValue, DMcob_SetValue and DMcob_ResetValue are available to the application.

The definitions can be found in the “Attribute Reference”.

7.2 Class Definition

The attribute AT-class is available to request the class of an attribute. Calling **DMcob_GetValue** can result in the following possibilities (as defined in **IDMcobws.cob**).

Class Identifier	Meaning
DM-Class-Accel	Accelerator
DM-Class-Application	Application
DM-Class-Canvas	Canvas
DM-Class-Check	Checkbox
DM-Class-Color	Color
DM-Class-Cursor	Cursor
DM-Class-Dialog	Dialog
DM-Class-Editext	Edittext
DM-Class-Font	Font
DM-Class-Func	Function
DM-Class-Groupbox	Groupbox
DM-Class-Image	Image
DM-Class-Import	Import
DM-Class-Listbox	Listbox

Class Identifier	Meaning
DM-Class-Menubox	Menubox
DM-Class-Menuitem	Menuitem
DM-Class-Menusep	Menuseparator
DM-Class-Messagebox	Messagebox
DM-Class-Module	Module
DM-Class-Notebook	Notebook
DM-Class-Notepage	Notepage
DM-Class-Poptext	Poptext
DM-Class-Push	Pushbutton
DM-Class-Radio	Radiobutton
DM-Class-Rect	Rectangle
DM-Class-Rule	Rule
DM-Class-Scroll	Scrollbar
DM-Class-Statext	Statictext
DM-Class-Tablefield	Tablefield
DM-Class-Text	Text
DM-Class-Tile	Tile
DM-Class-Timer	Timer
DM-Class-Var	Variable
DM-Class-Window	Window

8 COBOL in Distributed Environment

The following conditions have to be noticed when using COBOL together with the distributed Dialog Manager.

- Usually the function call via the net is expensive, whereby the transmitted contents is only insignificantly decisive. Thus when calling via the parameter interface, the COBOL functions should be supplied as far as possible with the information necessary for them. This procedure charges the computer less than querying each missing piece of information by DMcob_GetValue individually. Thus records are especially suitable as parameters since they can be configured by the dialog designer himself.
- If contents of listboxes or tablefields are to be processed, this contents should first be copied or created respectively in a temporary memory: copy with DMcob_GetVector, create with DMcob_InitVector. Then the contents can be inquired individually but locally with DMcob_GetVector and can be set individually withDMcob_SetVectorValue. The assignment via network follows a function call with DMcob_SetVector. For the use in a network environment, this solution is much more efficient than setting each listbox or tablefield contents individually via the network.

Example

In the following example, a listbox is filled with any number of items defined by the user. The filling is first done in the temporary memory which is then assigned to the object.

The contents is queried individually after it was copied via the network in a second function.

Dialog file

```
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  */

/* dialog start rule */
```

```
on dialog start
{
    TestAppl.active := true;
    if (not TestAppl.active) then
    TestAppl.local := true;
        TestAppl.active := true;
    endif
}

/* this rules 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();
}
```

Corresponding COBOL program

```
    *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(4) binary.
77    DLG-STRING PIC X(80).
77    DLG-STR-LEN PIC 9(9) binary.

*This COBOL function creates a temporary memory in DM
*and then assigns it to a listbox

PROCEDURE DIVISION USING DM-COMMON-DATA DLG-OBJECT DLG-COUNT
    DLG-STRING DLG-STR-LEN.
ORGANIZE-IN SECTION.
    MOVE DLG-COUNT TO ICOUNT.
*Initialization of memory in DM
    CALL "DMcob_InitVector" USING DM-StdArgs DM-POINTER DT-String
    ICOUNT.
*Initialization of DM-Value structure
    MOVE DT-STRING TO DM-DATATYPE.
    MOVE DLG-STRING TO DM-VALUE-STRING.

*Setting of individual contents
    PERFORM VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER = DLG-COUNT
    MOVE COUNTER TO DM-INDEX
    MOVE DLG-STRING TO STR-TAB
*Preparation of a changed string for display
    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.
```

```
*Transfer of the stored values to the display
    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.

*Freeing memory
    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.

*Inquiring individual contents
    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.
*Freeing memory
    CALL "DMcob_FreeVector" USING DM-StdArgs DM-POINTER.
    GOBACK.
```

9 Compiling and Linking Dialog Manager Programs

This chapter describes how programs developed with the Dialog Manager have to be compiled and linked.

9.1 Copy Files

All source files of the application which imply any reference to the DM have to include the files **IDM-cobws.cob** or **IDMcobls.cob** provided by the DM. Depending on where these files were installed, the include path for the compiler has to be set.

» **IDMcobws.cob**
This file contains all definitions of the Dialog Manager and must be included once into the working storage section of the application. COBOL allocates memory for the defined structures in this file so that the application can access the data inside of it.
This file has to be copied in a module exactly once in each application.

» **IDMcobls.cob**
This file contains all definitions of the Dialog Manager without the values. This file can be included into the "linkage storage section" of all other COBOL subprograms. This file makes the definitions of the Dialog Manager available without allocating memory.
You can use this file as often as you want in an application.

» **IDMcobls.cob**
This file contains all definitions of the Dialog Manager without values. It can be copied into the "link-age storage section" of all other COBOL subprograms. With this file you can conserve the Dialog Manager definitions without allocating memory capacity. You can therefore use this file in the application as often as you want.

» **IDMcoboc.cob**
This file must be included into those COBOL programs which are using the COBOL callback func-tion of the Dialog Manager. Since this file does not contain value definitions but only a structure definition, this file has to be copied into the "linkage storage section".

If, in the application, COBOL functions are included which are called directly by the DM and which have a record as parameter, the COBOL file generated of the DM via the option **+writetrampolin** has to be copied into the respective module in the "linkage storage section", too. It is the only way you can access this structure in the COBOL program.

Apart from these COBOL copy files the following C Include files are necessary to create a runnable COBOL-DM program.

- **IDMuser.h**
  This file contains all definitions of the DM for the programming language C. It is needed for compiling the C modules which have been generated with the program **gencobfx** or the option **+writetrampolin**. This file is included there.
- **IDMcobol.h**
  This include file is only needed when records are to be transferred to COBOL functions. The module generated with the option **+writetrampolin** includes this file.

9.2 Compiling the Generated C Files

To compile the C modules necessary for the COBOL program, the C compiler has to be called with the definitions for the operating system and the hardware included in the file "MakeDefs".

Please take these definitions from the file installed on your machine.

Example

PC with Microsoft Windows

```
cl -DDOS -DMSW -DMSC
```

Apart from these symbols, when compiling the module generated with the option **+writetrampolin**, you also have to define for which kind of COBOL the module has to be compiled:

The following flags have to be set:

Compiler	Switch for compiling C sources
Micro Focus COBOL	-DUFCOB

Example

- 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 Compiling the COBOL Module

The COBOL modules have to be compiled depending on the used compiler that the modules can work together with the Dialog Manager. In addition, the options to be set in the COBOL modules depend on the COBOL compiler.

9.3.1 Micro Focus COBOL

There are no special settings necessary in the source file. The DM definitions are made available with the command "COPY".

Settings	none
Copy files	`COPY <FILENAME>`
Flags	**-u** for interpretable files **-x** for object files
Compilation	`cob -c -x <file>`

Note

In a Microsoft Windows environment the switch **/LITLINK** has to be set in addition when compiling, otherwise the COBOL program cannot call the Dialog Manager.

The environment symbol `COBDIR` has to include the COBOL copy files as include-directory.

9.4 Linking

To link the COBOL application together, the created C source files have to be compiled by means of the C compiler first. After having successfully compiled the COBOL programs, they have to be linked with the corresponding COBOL libraries as well as with the DM library and the corresponding COBOL module.

Environment	Compiler	Use	Files
PC Windows	cob	local	ufcob.obj dm.lib +
PC Windows	vcob	Client	ufcob.obj dm.lib dmndx.lib +
UNIX	cob	local/Motif	ufcob.obj libIDM.a +
UNIX	cob	Server	ufcobnet.o libIDMnet.

In the table "+" means that the respective libraries of the window system have to be linked additionally. The libraries libIDMinit.a or dminit.lib have also to be linked to all COBOL programs.

For linking purposes system-specific shell scripts, jobs or batch files are available; they are responsible for the linking process of the application. These files are provided together with the examples.

The name of the files are:

UNIX and MICRO FOCUS COBOL:	ufdmlink*
PC and MICRO FOCUS COBOL:	see example makefile

*) If the switch "-net" is given here when calling, the program is linked as server process without display part.

9.5 Examples for Makefiles

In the following chapter the various environments are introduced in form of listed makefiles. These makefiles are part of the distribution and therefore you do not need to type them in. In the chapter "Peculiarities" all special features of the system are mentioned, in "General Part of Declarations" the beginning part of a makefile is shown, in the chapter "Compiling" you will find a description of a command line and of how the COBOL source can be compiled, in the chapter "Compiling the C Source" you will learn how to compile a generated C file and finally in the chapter "Linking" you will learn how to link the application together. For illustrating purposes, there is always an example attached to the text.

9.5.1 Micro Focus COBOL on Microsoft Windows

9.5.1.1 Peculiarities

When translating the COBOL sources the switch **/LITLINK** has to be set, when translating the C sources the switches **-DUFCOB** and **-DUFCOB_LINK** have to be set.

For the COBOL programs being able to call up the Dialog Manager, the module **mfc6intw.obj** of the COBOL compiler has to be linked to them. If the module **coboljmp.obj** is linked to them, the interface functions in the Dialog Manager can be called up without a prefixed underline "_".

9.5.1.2 General Part of Declarations

```
##############################################################

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 Compiling a COBOL Source

```
obj\table.obj: src\table.cob
    @echo [cobol table]
    @$(Cobol) src\table.cob, obj\table.obj,,,/LITLINK
```

9.5.1.4 Compiling the Generated C Source

```
obj\tablec.obj : src\tablec.c
```

```
    $(CC) -c $(CFLAGS) -DUFCOB -DUFCOB_LINK src\tablec.c
```

9.5.1.5 Linking the Application

```
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)
```

mkdef.bat is the script supplied by the Dialog Manager; it looks as follows:

```
@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 on Unix

9.5.2.1 Peculiarities

On the UNIX systems the dynamic reloading of the COBOL modules can be used by the COBOL runtime system. To do so, the names of the functions defined in the dialog have to correspond with the functions included in it. If this behavior shall be used, the COBOL sources have to be compiled in an intermediate format ".gnt" or ".int" and do not need to be linked to the application. At the actual runtime the intermediate files have to be found by the COBOL runtime system. To do so, the environment variable `COBLIB` has to be set accordingly. You also have the possibility to compile the COBOL sources to object files and to link them to a normal application.

To compile the generated C file, the switch **-DUFCOB** has to be set. If a program shall be linked without the COBOL runtime system, the switch **-DUFCOB_LINK** has to be set additionally when translating the generated C file.

9.5.2.2 General Part of Declarations

```
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 Compiling a COBOL Source

```
table.o:    table.cbl
        @echo [Compiling table]
        $(NOECHO) $(COB) $(COBDEFS) table.cbl
```

9.5.2.4 Compiling a Generated C Source

```
tablec.o:    tablec.c
        $(CC) $(COPTS) $(CDEFS) $(IDM_INC) -c tablec.c
```

9.5.2.5 Linking the Application

If a server stub is to be linked together, the switch **-net** has to be set on calling the script **ufdmlink**.

```
Tablenet:    table.o tablec.o table_.o
        @echo [Linking tablenet]
        $(NOECHO) $(VCDMLINK) -net -o tablenet table.o
            tablec.o table_.o
```

If a local application is to be linked together, the switch **-dvl** has to be set on calling the script **ufdm-link**.

```
Table:    table.o tablec.o table_.o
        @echo [Linking table]
        $(NOECHO) $(VCDMLINK) -dvl -o table table.o tablec.o
                 table_.o
```

The shell script **ufdmlink** looks as follows:

```
#!/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

A

Accelerator 219

access

  attributes 65

  DM identifier 86

  functions 86

  listbox 88

  object attribute 86

  tablefield 88

anyvalue 46

API 11

application 213-214

Application 219

application programming interface 11

AT-active 192, 217

AT-content 192

AT-label 66

AT-sensitive 192, 217

AT-userdata 192, 217

attribute

  changing 47

  inquiring 47

B

basic data types 40

BindFuncs 26, 68, 73-74, 77

BindThruLoader 75

C

C module 77

callback function 59

Canvas 219

canvas function 72

CFR-LOAD 61

Checkbox 219

COBDIR 233

COBLIB 237

COBOL

  communication 21

  environment variable COBDIR 235

COBOL object callback 60

COBOL program

  definition 23

  distributed 68

  local 68

COBOLAPPFINISH 25, 27, 82, 92-93

COBOLAPPINIT 25, 27, 68, 82, 92, 213-214

coboljmp.obj 234

COBOLMAIN 25, 68, 82, 91, 194

cobRecInit 77

CobRecMInit 26, 28

cobw.lib 235

code page 105-106

collection 14

collection data type 47

  return 47

collection data types 41

collections 41

Color 219

compiler 11, 77

  UNIX 25, 27

compiling 39

  DM program 231

constant 98

COPY 62, 76, 232

copy file 13, 231

CP-acp 106

CP-ascii 105

CP-cp1252 106

CP-cp437 105

CP-cp850 105

CP-dec169 105

CP-hp15 106

CP-iso6937 105

CP-iso8859 105

CP-jap15 106

CP-prc15 106

CP-roc15 106

CP-roman8 105

CP-utf16 106

CP-utf16b 106

CP-utf16l 106

CP-utf8 106

CP-winansi 105

creating objects 86

Cursor 219

D

data function 71

data function structure 63

data type 59

  collection 14

  list 14

data types 40

Datamodel 63

default object 59

destroying objects 86

dialog 213-214

  definition 16

  finish 85

Dialog 219

dialog data types

  mapping 23

distributed applications 27

DM-attribute 49, 66

DM-Class- Notebook 220

DM-Class- Notepage 220

DM-Class-Accel 219

DM-Class-Application 111, 219

DM-Class-Canvas 111, 219

DM-Class-Check 111, 219

DM-Class-Color 219

DM-Class-Cursor 219

DM-Class-Dialog 219

DM-Class-Doccursor 111

DM-Class-Document 111

DM-Class-Editext 111, 219

DM-Class-Filereq 111

DM-Class-Font 219
DM-Class-Func 219
DM-Class-Groupbox 111, 219
DM-Class-Image 111, 219
DM-Class-Import 219
DM-Class-Layoutbox 111
DM-Class-Listbox 111, 219
DM-Class-Mapping 111
DM-Class-Menubox 111, 220
DM-Class-Menuitem 111, 220
DM-Class-Menusep 111, 220
DM-Class-Messagebox 111, 220
DM-Class-Module 220
DM-Class-Notebook 111
DM-Class-Notepage 111
DM-Class-Poptext 111, 220
DM-Class-Progressbar 111
DM-Class-Push 111, 220
DM-Class-Radio 111, 220
DM-Class-Record 111
DM-Class-Rect 111, 220
DM-Class-Rule 220
DM-Class-Scroll 111, 220
DM-Class-Spinbox 111
DM-Class-Splitbox 111
DM-Class-Statext 111, 220
DM-Class-Statusbar 111
DM-Class-Tablefield 111, 220
DM-Class-Text 220
DM-Class-Tile 220
DM-Class-Timer 111, 220
DM-Class-Toolbar 111
DM-Class-Transformer 111
DM-Class-Treeview 112
DM-Class-Var 220
DM-Class-Window 112, 220
DM-CO-FIRSTLOAD 61
DM-CO-LASTLOAD 61
DM-CO-OBJECT 61
DM-CO-REASON 61
DM-CO-VISFIRST 61
DM-CO-VISLAST 61
DM-COMMON-DATA 60
DM-Content-Data 34
DM-count 98
DM-COUNT 61
DM-Datafunc-Data 63
DM-datatype 50, 57, 66
DM-error 67
DM-error-code 90
DM-error-package 89
DM-error-severity 89
DM-ErrorData 181
DM-ErrorDetail 118
DM-getsep 24, 42, 44-45
DM-getsep-u 44
DM-HEADER 61
DM-index 50
DM-indexcount 50
DM-long-first 50
DM-long-second 50
DM-method 98
DM-ModuleIDM 90
DM-ModuleUnix 90

DM-object 49
DM-ObjectCallback-Data 59-60
DM-objectID 98
DM-ocb-evbits 59
DM-ocb-index 59
DM-ocb-object 59
DM-ocb-status 59
DM-options 24, 42, 44, 214
DM-rescode 24, 42, 44
DM-second 50
DM-setsep 24, 42, 44-45
DM-setsep-u 44
DM-SeverityError 89
DM-SeverityFatal 89
DM-SeverityProgErr 89
DM-SeveritySuccess 89
DM-SeverityWarning 89
DM-status 24, 42-43, 99
DM-StdArgs 24, 42, 44, 67
DM-success 43, 67
DM-truncspaces 24, 43-45
DM-usercodepage 24, 43-44
DM-usercodepage-u 44
DM-va-attribute 54
DM-va-datatype 55, 57, 99
DM-va-index 54
DM-va-indexcount 54
DM-va-object 54
DM-va-second 55
DM-va-value-attribute 55
DM-va-value-boolean 55
DM-va-value-cardinal 55
DM-va-value-classid 55
DM-va-value-first 55
DM-va-value-integer 55
DM-va-value-long-first 55
DM-va-value-long-second 55
DM-va-value-method 55
DM-va-value-object 55
DM-va-value-pointer 55
DM-va-value-second 55
DM-va-value-string 56
DM-va-value-string-getlen 56
DM-va-value-string-putlen 55
DM-va-value-string-size 56
DM-va-value-string-u 56
DM-Value 46, 48, 98
DM-value-attribute 51
DM-value-boolean 50
DM-value-cardinal 50
DM-value-classid 50
DM-value-count 54
DM-value-first 50
DM-value-integer 50
DM-value-long-first 51
DM-value-long-second 51
DM-value-method 51
DM-value-object 50
DM-value-pointer 46, 51
  life time 47
DM-value-second 50
DM-value-size 54
DM-value-string 51, 66
DM-value-string-getlen 51

DM-value-string-putlen 51

DM-value-string-size 51

DM-value-string-u 51

DM-ValueArray 51, 98

DM-ValueIndex 15, 52

DM data type 40

DM identifier

access 86

DM program

compiling 231

linking 231

DM_FuncMap 72

DMcob_CallFunction 82, 87, 96

DMcob_CallMethod 82, 87, 98

DMcob_CallRule 82, 87, 101

DMcob_Control 82, 87, 104

DMcob_CreateFromModel 82, 87, 108

DMcob_CreateObject 82, 87, 110

DMcob_DataChanged 82, 86, 112

DMcob_DeleteArgString 82, 90, 114

DMcob_Destroy 82, 87, 115

DMcob_DialogPathToID 83, 86, 116

DMcob_ErrMsgText 83, 89, 118

DMcob_EventLoop 25, 68, 83, 85, 119

DMcob_ExecRule 83, 87, 101, 120

DMcob_FatalAppError 83, 89, 122

DMcob_FreeVector 30, 33, 83, 88, 123

DMcob_GetArgCount 83, 90, 124

DMcob_GetArgString 83, 90, 125

DMcob_GetValue 65-66, 83, 86, 128, 221

DMcob_GetValueBuff 83, 86, 130

DMcob_GetVector 83, 88, 132, 221

DMcob_GetVectorValue 83, 88, 136, 151, 153

DMcob_Initialize 25-27, 45, 61, 68, 83, 85, 138

DMcob_InitVector 29, 31, 35, 83, 88, 139, 155, 221

DMcob_LGetValueBuff 141, 144

DMcob_LGetVector 147

DMcob_LoadDialog 25-26, 68, 83, 85, 157, 213-214

DMcob_LoadProfile 83, 85, 159

DMcob_LSetValueBuff 160, 162

DMcob_OpenBox 83, 87, 171

DMcob_OpenBoxBuff 83, 87, 173

DMcob_PathToID 83, 86, 176

DMcob_PutArgString 83, 90, 178

DMcob_QueryBox 83, 87, 179

DMcob_QueryError 83, 89, 181

DMcob_QueueExtEvent 83, 87, 182

DMcob_ResetValue 83, 86, 184

DMcob_SetValue 65, 84, 86, 129, 131, 142, 145, 185

DMcob_SetValueBuff 84, 86, 187

DMcob_SetVector 30, 35, 84, 88, 164, 189, 221

DMcob_SetVectorValue 32, 84, 88, 168, 170, 192, 221

DMcob_ShutDown 84-85, 194

DMcob_StartDialog 25-26, 68, 84-85, 195

DMcob_StopDialog 84-85, 196

DMcob_TraceMessage 84, 87, 197

DMcob_ValueChange 15, 84, 88, 198

DMcob_ValueChangeBuffer 15, 84, 88, 200

DMcob_ValueCount 15, 84, 88, 202
DMcob_ValueGet 15, 84, 88, 204
DMcob_ValueGetBuffer 15, 84, 88, 206
DMcob_ValueGetType 15, 84, 88, 208
DMcob_ValueIndex 15, 84, 89, 209
DMcob_ValueInit 15, 84, 89, 211
DMF-AcceptChild 214
DMF-AppendValue 198, 200, 215
DMF-BindForLoader 215
DMF-CreateInvisible 108, 110, 215
DMF-CreateModel 108, 110, 215
DMF-DontFreeLastStrings 202, 204, 206, 209, 215
DMF-DontTrace 215
DMF-DontWait 119, 215
DMF-ForceDestroy 115, 196, 215
DMF-GetLocalString 129, 131, 142, 145, 202, 204, 206, 209, 216
DMF-GetMasterString 129, 131, 142, 145, 202, 204, 206, 209, 216
DMF-GetTextID 129, 131, 142, 145, 216
DMF-IncludeIdent 216
DMF-IncludeModule 118, 216
DMF-IncludeSeverity 118, 216
DMF-IncludeText 118, 216
DMF-InheritFromModel 110, 216
DMF-Inhibit 161, 163, 184, 186, 188, 216
DMF-InhibitTag 216
DMF-LogFile 217
DMF-OmitActive 165, 190, 217
DMF-OmitSensitive 165, 190, 217
DMF-OmitStrings 165, 190, 217
DMF-OmitUserData 217
DMF-SetCodePage 105, 217
DMF-SetFormatCodePage 105
DMF-ShipEvent 161, 163, 184, 186, 188, 217
DMF-SignalMode 104
DMF-Silent 213-214
DMF-SortBinary 198, 200, 217
DMF-SortLinguistic 198, 200, 217
DMF-SortReverse 198, 200, 217
DMF-StaticValue 211, 217
DMF-Synchronous 217
DMF-UpdateScreen 104, 218
DMF-UseUserData 165, 218
DMF-Verbose 112, 213-214, 218
DMF-XlateString 218
DMmfviscob_BindThruLoader 84-85, 213
DMufcob_BindThruLoader 28, 84-85, 214
DT-accel 57
DT-anyvalue 14, 46, 57
DT-application 57
DT-attribute 57
DT-boolean 57, 99
DT-class 57
DT-color 57
DT-cursor 57
DT-enum 57
DT-event 57
DT-font 57
DT-hash 57
DT-index 57
DT-instance 58
DT-integer 58, 99

DT-list 58

DT-matrix 58

DT-object 58

DT-refvec 58

DT-rule 58

DT-scope 58

DT-string 58, 66

DT-text 58

DT-tile 58

DT-timer 58

DT-undefined 58

DT-var 58

DT-vector 58

DT-void 58

DUFCOB 39

DUFCOB_LINK 39

DVISCOB 39

E

Edittext 219

EM-activate 59

EM-charinput 59

EM-close 59

EM-dbselect 59

EM-deactivate 60

EM-deiconify 60

EM-deselect 60

EM-deselect-enter 60

EM-finish 60

EM-focus 60

EM-help 60

EM-hscroll 60

EM-iconify 60

EM-key 60

EM-modified 60

EM-move 60

EM-resize 60

EM-scroll 60

EM-select 60

EM-start 60

EM-vscroll 60

ENTRY 68

error

  handling 89

  string 118

executing a dialog 85

F

FILLTAB 21

finish 85

Font 219

function

  record 37

Function 219

function address

  transfer 72

function definition file 73

functions 82

  records 72, 76

G

gencobfx 72-73

gencobfx.exe 74

GETADDR 21

Groupbox 219

H

hash 14, 42

I

identifier 3

IDM

finish 85

IDMcobls.cob 65, 98, 139, 231

IDMcoboc.cob 62, 231

IDMcobws.cob 40, 59, 65, 90, 98, 139, 219, 231

Image 219

Import 219

initializing a dialog 85

input and output parameter 82

input parameter 82

instance 59

ISO 8859-1 105

L

linkage storage

section 231

linking 39, 68, 233

DM program 231

list 14, 42

Listbox 219

M

main program 194

MakeDefs 232

Managed DM-Value 14

Managed Value 47

matrix 14, 42

Menubox 220

Menuitem 220

Menuseparator 220

Messagebox 220

method 98

mfc6intw.obj 234

-mfviscob 13

-mfviscob-u 13

Micro Focus Compiler 11

Micro Focus Visual COBOL 12

model 59

Module 220

MT-clear 98-99

MT-delete 98-99

MT-insert 98-99

N

National Character 12, 51, 56

network 221

Notebook 220

Notepage 220

NULL object 65

O

object

access attribute 86

callback 59-60

creating 86

destroying 86

object callback 71

options 214

output parameter 82

P

parameter 68

  options 214

  record 37

perform 87

Poptext 220

Pushbutton 220

PUTADDR 21

R

Radiobutton 220

record 37, 41

record parameter 13, 80

records 221

  parameters 72, 76

Rectangle 220

refvec 14, 42

reloading function 61

return

  values 67

Rule 220

S

Scrollbar 220

simulation 76

Speicherbereich 147

standard argument 42

Statictext 220

string parameters 45

summary of functions 82

T

TABFUNC 21

tablefield

  reloading 100

Tablefield 220

terminate 85

Text 220

Tile 220

Timer 220

tracefile 87

trampoline module 76

U

ufcob.obj 235

ufdmlink 39, 239

Unicode 12, 51, 56

  activation 12

UNIX 25

utilities 87

V

Variable 220

vcdmlink 39

vector 14, 42

Visual COBOL 12

W

Window 220

working storage section 231

+writeheader 80

+/-writetrampolin 13, 29, 39, 41