ISA Dialog Manager

Release Notes IDM 6

A.06.03.b

Die Release Notes informieren über Fehlerkorrekturen, Änderungen und Erweiterungen in den einzelnen Versionen des ISA Dialog Managers.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Deutschland

Microsoft, Windows, Windows 2000 bzw. NT, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10 und Windows 11 sind eingetragene Warenzeichen von Microsoft Corporation.

UNIX, X Window System, OSF/Motif und Motif sind eingetragene Warenzeichen von The Open Group.

HP-UX ist ein eingetragenes Warenzeichen von Hewlett-Packard Development Company, L.P.

Micro Focus, Net Express, Server Express und Visual COBOL sind Warenzeichen oder eingetragene Warenzeichen von Micro Focus International plc und/oder ihrer Tochterunternehmen in den USA, Großbritannien und anderen Ländern.

Qt ist ein eingetragenes Warenzeichen von The Qt Company Ltd. und/oder ihrer Tochterunternehmen.

Eclipse ist ein eingetragenes Warenzeichen von Eclipse Foundation, Inc.

TextPad ist ein eingetragenes Warenzeichen von Helios Software Solutions.

Alle genannten und ggf. durch Dritte geschützten Marken- und Warenzeichen unterliegen uneingeschränkt den Bestimmungen des jeweils gültigen Kennzeichenrechts und den Besitzrechten der jeweiligen eingetragenen Eigentümer. Allein aufgrund der bloßen Nennung ist nicht der Schluss zu ziehen, dass Markenzeichen nicht durch Rechte Dritter geschützt sind.

© 1987 – 2024; ISA Informationssysteme GmbH, Leinfelden-Echterdingen, Deutschland

Darstellungskonventionen

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

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

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

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

Beschreibungsmodus

Alle Schlüsselwörter sind fett und unterstrichen, z.B.

variable **integer** **function**

Indizierung von Attributen

Syntax für indizierte Attribute:

[I]

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

Identifikatoren

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

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

Die maximale Länge eines Identifikators beträgt *31* Zeichen.

Beschreibung der zugelassenen Identifikatoren in Backus-Naur-Form

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

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

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

<Ziffer> ::= 1 | 2 | 3 | … 9 | 0

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

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

Inhalt

Darstellungskonventionen 3

Inhalt 5

1 A.06.03.b 19

1.1 SVG-Support 19

1.1.1 Qt 19

1.1.2 Windows 19

1.2 Ergänzung zur Dumpstate-Ausgabe 19

1.2.1 WSI-Dumpstate 19

1.2.1.1 Motif / Qt 19

1.2.1.2 Windows 20

1.2.2 EXTERN-Dumpstate 21

1.3 Kompatibilität 21

1.3.1 Windows 21

1.3.2 Qt 21

1.4 Motif 21

1.5 Qt 22

1.6 Windows 22

1.7 Kern 24

1.8 Editor 26

1.9 DDM (Netz) 26

1.10 Debugger 28

1.11 COBOL 28

2 V.06.03.a5 30

2.1 Windows 30

2.2 COBOL 30

3 V.06.03.a4 31

3.1 Hinweis zur WSI Dumpstate-Ausgabe unter Windows 31

3.2 Kern 31

3.3 Windows 32

4 V.06.03.a3 33
4.1 Windows 33
4.2 Kern 33
5 A.06.03.a2 34
5.1 Qt 34
5.2 Kern 34
6 A.06.03.a 35
6.1 HighDPI Unterstützung 35
6.1.1 Layout-Ressourcen 36
6.1.2 Erweiterung an der Tile-Ressource 40
6.1.3 Erweiterung der Font-Ressource 42
6.1.4 Erweiterung am Setup-Objekt 43
6.1.5 Erweiterungen am Image-Objekt 44
6.1.6 Unterstützung von HiDPI Bild-Varianten 44
6.1.7 Startoptionen 45
6.1.8 Installationshinweise 46
6.1.9 Geometrie und Koordinaten 46
6.1.10 Unterstützung hoher Auflösungen unter Motif 47
6.1.11 Unterstützung hoher Auflösungen beim IDM für IDM for Windows 11 48
6.1.12 Unterstützung hoher Auflösungen unter Qt 50
6.1.13 Erweiterungen/Änderungen im IDMED 50
6.1.14 Unterstützung hoher Auflösungen bei der USW-Programmierung 50
6.2 Multi-Monitor Support unter Windows 52
6.3 Kompatibilität 53
6.3.1 Motif 53
6.3.2 Windows 53
6.3.3 Qt 53
6.4 Kern 54
6.5 Windows 55
6.6 Motif 60
6.7 Qt 60
6.8 DDM (Netz) 61
6.9 Debugger 62
6.10 Editor 62
6.11 USW 62
6.12 IFD 63
6.13 Demos 63

6.14 Dokumentation 63

7 A.06.02.o 64

7.1 Kern 64
7.2 Unix 64
7.3 Windows 65
7.4 Qt 66
7.5 DDM (Netz) 67
7.6 COBOL 67
7.7 Dokumentation 68

8 A.06.02.n2 69

8.1 Unix 69
8.2 DDM (Netz) 69
8.3 COBOL 69

9 A.06.02.n 70

9.1 Kern 70
9.2 Unix 71
9.3 Motif 71
9.4 Windows 71
9.5 Qt 72
9.6 Debugger 72
9.7 DDM (Netz) 72
9.8 COBOL 72
9.9 Demos 73
9.10 Dokumentation 73

10 V.06.02.m2 75

10.1 COBOL 75
10.2 DDM (Netz) 75
10.3 Unix 75

11 A.06.02.m 76

11.1 Kern 76
11.2 Debugger 78

12 A.06.02.l 79

12.1 Kompatibilität 79

12.2 Kern ... 79
13 A.06.02.k ... 84
13.1 Kern ... 84
13.2 COBOL ... 87
13.3 Dokumentation ... 88
14 A.06.02.j3 ... 89
14.1 Kern ... 89
15 A.06.02.j2 ... 90
15.1 COBOL ... 90
16 A.06.02.j ... 91
16.1 Windows ... 91
16.2 Kern ... 91
16.3 Dokumentation ... 91
17 A.06.02.i ... 92
17.1 Unix ... 92
17.2 Windows ... 92
17.3 Motif ... 92
17.4 Qt ... 92
17.5 Kern ... 92
17.6 Debugger ... 93
17.7 Dokumentation ... 93
18 A.06.02.h ... 95
18.1 Kompatibilität ... 95
18.2 Windows ... 95
18.3 Kern ... 95
18.3.1 Prüfung der Verwendung lokaler Variablen ... 96
18.3.2 Deaktivierung von Meldungen der Fehleranalyse ... 98
18.4 Debugger ... 98
18.4.1 Teilweises Laden sehr großer Tracedateien im Tracefile Analyzer ... 100
18.5 DDM (Netz) ... 101
18.5.1 Neue Attribute und Optionen für SSL und SSH ... 101
18.5.1.1 Attribut .privatekeyfile ... 101
18.5.1.2 Attribut .publickeyfile ... 102
18.5.1.3 Attribut .options[enum] ... 102

18.5.1.4 Startoption -IDMssl_options ... 103
18.6 COBOL ... 103
19 A.06.02.g ... 105
19.1 Kompatibilität ... 105
19.2 Unix ... 105
19.3 Windows ... 105
19.4 Qt ... 106
19.5 Motif ... 107
19.6 Kern ... 107
19.7 Editor ... 109
19.8 Debugger ... 109
19.9 DDM (Netz) ... 109
19.10 COBOL ... 110
19.11 Fehleranalyse ... 110
19.11.1 Geänderte Ausgabe bei Fehlern und Warnungen in Regeln ... 110
19.11.2 Erweiterte Prüfungen in der Regelsprache ... 111
19.11.3 Steuerung des Schweregrads ... 113
19.11.4 Aufruf von Fehlerhandlern ... 114
19.11.5 Erkennung fehlender return-Anweisungen ... 114
19.12 Neues Werkzeug Tracefile Analyzer ... 114
19.13 Alternativer Import mit use ... 115
19.13.1 Der alternative Import-Mechanismus ... 115
19.13.1.1 Besonderheiten ... 116
19.13.1.2 Groß- und Kleinschreibung in Dateinamen ... 117
19.13.1.3 Empfehlungen ... 117
19.13.2 Sprachbeschreibung und Use-Pfad ... 117
19.13.3 Use-Pfad, Dateinamen und Namensrestriktionen ... 118
19.13.4 Suchpfad für Interface-, Modul-, Dialog- und Binärdateien ... 118
19.13.5 Vergleich zwischen import und use ... 120
19.13.6 Kompilieren von Interface- und Binärdateien ... 120
19.13.7 Dynamisches Laden und Entladen von Modulen ... 122
19.13.8 Umgebung und IDM-Startoptionen ... 123
19.13.9 Startoptionen für idm.exe, pidm.exe und Builder-Prozess ... 124
19.13.10 Ergänzungen zur Funktion DM_ControlEx ... 125
19.13.11 Attribut .searchpath ... 126
19.13.12 Methode :use() ... 127
19.13.13 Methode :unuse() ... 128
19.13.14 Ergänzungen am IDM-Editor ... 130
19.13.14.1 Objekte-Browser ... 130

19.13.14.2 Applikationen in der Werkzeugkiste ... 130
19.13.14.3 Änderung an den Erzeugen-Dialogen ... 130
19.13.14.4 Konfigurationsseite Suchpfad ... 131
19.13.14.5 Spezifische Objekt-Eigenschaften für Dialog und Modul ... 131
19.13.14.6 Spezifische Objekt-Eigenschaften für use (Import-Objekt) ... 132
19.13.15 Ergänzungen am IDM-Eclipse-Plugin ... 133
19.13.15.1 Einstellungen (Preferences und Project-Properties) ... 133
19.13.15.2 Informationsspeicher ... 134
19.13.15.3 IDM-Applikations-Starter ... 134
19.13.15.4 Strukturansicht ... 134
19.14 Reihenfolge beim Initialisieren und Beenden von Modulen ... 135
19.15 Reguläre Ausdrücke ... 135
19.15.1 PCRE-Bibliothek zur Unterstützung Regulärer Ausdrücke ... 135
19.15.2 Neue eingebaute Funktion regex() ... 137
19.15.3 Reguläre Ausdrücke als Format ... 142
19.15.4 Ergänzung der Installation ... 142
19.16 Asynchroner Methodenaufruf ... 142
19.16.1 Eingebaute Funktion sendmethod() ... 143
19.16.2 C-Schnittstellenfunktion DM_SendMethod() ... 143
19.17 Laden und Speichern von Konfigurationsdateien (profiles) ... 144
19.17.1 C-Schnittstellenfunktion DM_SaveProfile() ... 145
19.17.2 Eingebaute Funktion loadprofile() ... 146
19.17.3 Eingebaute Funktion saveprofile() ... 146
19.18 Neue eingebaute Funktion queryhelp() ... 147
19.19 Neue eingebaute Funktionen für Sammlungen und Strings ... 148
19.19.1 append() ... 148
19.19.2 delete() ... 150
19.19.3 exchange() ... 153
19.19.4 insert() ... 155
19.20 Unterstützung von SSH und SSL im DDM ... 159
19.20.1 Schlüssel und Zertifikate ... 159
19.20.2 Erweiterungen am Application-Objekt ... 160
19.20.2.1 Erweiterungen bei vorhandenen Attributen ... 160
19.20.2.1.1 .connect ... 160
19.20.2.1.2 .transport ... 160
19.20.2.1.3 .exec ... 161
19.20.2.2 Neue Attribute ... 162
19.20.2.2.1 .certificatefile ... 162
19.20.2.2.2 .password ... 162
19.20.2.2.3 .username ... 163
19.20.3 Änderungen bei Startoptionen ... 163

20 A.06.02.a – f ... 165
21 A.06.01.h ... 166
21.1 Windows ... 166
21.2 Qt ... 166
21.3 Unix ... 167
21.4 Kern ... 167
21.5 DDM (Netz) ... 168
21.6 Debugger ... 168
22 A.06.01.g ... 169
22.1 Unix ... 169
22.2 Windows ... 169
22.3 Qt ... 170
22.4 Kern ... 171
22.5 Editor ... 171
22.6 Debugger ... 171
22.7 COBOL ... 172
22.7.1 Neue Schnittstellenfunktion DMcob_CreateFromModel ... 172
23 A.06.01.f3 ... 174
23.1 Windows ... 174
23.2 Qt ... 174
23.3 Kern ... 174
23.4 Debugger ... 175
24 A.06.01.f2 ... 176
24.1 COBOL ... 176
25 A.06.01.f ... 177
25.1 Kompatibilität ... 177
25.2 Windows ... 177
25.3 Qt ... 177
25.4 Kern ... 178
25.5 COBOL ... 178
26 A.06.01.e ... 179
26.1 Kompatibilität ... 179
26.2 Windows ... 179

26.3 Qt ... 179
26.4 Kern ... 179
26.5 Unterstützung von Microsoft UI Automation ... 180
26.5.1 Zuordnung von IDM-Objektklassen zu UIA Control Types ... 180
26.5.2 Unterstützte UIA Properties und Control Pattern Methoden ... 183
26.5.2.1 Value Pattern ... 186
26.5.2.2 RangeValue Pattern ... 186
26.5.2.3 Scroll Pattern ... 186
26.5.2.4 ScrollItem Pattern ... 187
26.5.2.5 Transform Pattern ... 187
26.5.2.6 Grid Pattern ... 187
26.5.2.7 GridItem Pattern ... 188
26.5.2.8 Selection Pattern ... 188
26.5.2.9 SelectionItem Pattern ... 188
26.5.2.10 Toggle Pattern ... 189
26.5.2.11 ExpandCollapse Pattern ... 189
26.5.2.12 Window Pattern ... 189
26.5.2.13 Dock Pattern ... 190
26.5.2.14 Table Pattern ... 190
26.5.2.15 TableItem Pattern ... 190
26.5.3 UIA Objektidentifikation ... 190
26.5.4 Besonderheiten ... 191
26.5.4.1 Edittext ... 191
26.5.4.2 Poptext ... 191
26.5.4.3 Formatierte Eingabe ... 191
26.5.4.4 Client-Bereich ... 192
26.5.4.5 Virtuelle Bereiche ... 192
26.5.4.6 Splitbox ... 192
26.5.4.7 Menüs ... 192
26.5.4.8 Messagebox und Filereq ... 192
26.5.4.9 Image-Objekt ... 192
26.5.4.10 Toolbar-Objekt ... 193
26.5.4.11 Treeview-Objekt ... 193
26.5.4.12 Benutzerinteraktion vs. Automatisierung ... 193
26.5.5 Hinweise zu HP UFT 12.52 ... 193
26.5.6 Erweiterung zur C-Schnittstelle ... 196
26.5.7 Attribut .acc_text ... 197
26.5.8 Attribut .acc_label ... 198
26.5.9 Erweiterung der Tile-Ressource ... 199
26.5.10 Erweiterung des Tracing ... 199
27 A.06.01.d ... 200
27.1 Windows ... 200

27.2 Qt ... 200
27.3 Kern ... 200
27.4 COBOL ... 201
27.5 Neue COBOL-Schnittstelle für Micro Focus Visual COBOL ... 201
27.6 Dynamisierte Funktionsaufrufe ... 202
27.6.1 Dynamische Anbindung von Record-Funktionen ... 202
27.6.2 Codepage bei Anwendungsfunktionen ... 203
27.6.3 Siehe auch ... 203

28 A.06.01.c ... 204

28.1 Kompatibilität ... 204
28.2 Windows ... 204
28.3 Motif ... 205
28.4 Qt ... 205
28.5 Kern ... 207
28.6 Editor ... 207
28.7 Debugger ... 208
28.8 Beispiele ... 209
28.9 Locale-Unterstützung und die Auswirkungen auf String- und Sortierfunktionen ... 209
28.9.1 Standardverhalten und sicherer Fallback ... 209
28.9.2 Empfehlungen ... 209
28.9.3 Erzwungene Locale-Benutzung ... 210
28.9.4 Ergänzungen der Funktion DM_Initialize() ... 211
28.9.5 Ergänzungen der Format-Ressource ... 212
28.9.6 Ergänzungen am Escaping von Unicode-Zeichen ... 213
28.10 Dokumentation ... 213
28.10.1 Systemvoraussetzungen ... 213
28.10.2 Bedienung der HTML-Dokumentation ... 213
28.10.3 Kontextsensitive Hilfe ... 215

29 A.06.01.b3 ... 216

29.1 Kern ... 216

30 A.06.01.b2 ... 217

30.1 Windows ... 217
30.2 Motif ... 218
30.3 Kern ... 218
30.4 USW ... 218

31 A.06.01.b ... 220
31.1 Windows ... 220
31.1.1 Installation ... 221
31.1.1.1 Registrierung von Dateierweiterungen ... 221
31.1.1.2 Merge-Module für die Redistribution entfallen ... 221
31.2 Unix ... 222
31.3 Motif ... 222
31.4 Qt ... 222
31.5 Kern ... 223
31.6 Debugger ... 224
31.6.1 Starten des Debuggers ... 225
31.6.2 Objektansicht (Object View) ... 226
31.6.2.1 Objekte suchen ... 227
31.6.2.2 Objekte und ihre Attribute überwachen ... 227
31.6.2.3 Attribute filtern ... 229
31.6.2.4 Attributänderungen ... 230
31.7 USW ... 231
31.8 COBOL ... 233
31.9 Neue Stringfunktionen strreplace(), concat() und split() ... 234
31.9.1 strreplace() ... 234
31.9.2 concat() ... 237
31.9.3 split() ... 239
31.10 Verwendung von XML mit dem Datenmodell ... 240
31.10.1 Beispiel ... 241
31.10.2 Neuer Indexwert dopt_cache_data des Attributs dataoptions ... 244
31.10.3 Neue Attribute des Doccursors ... 244
31.10.3.1 .dataselect ... 244
31.10.3.2 .dataselectattr ... 246
31.10.3.3 .dataselecttype ... 247
31.10.3.4 .dataselectcount ... 248
31.11 Laufzeitbegrenzung von Testlizenzen ... 249
32 A.06.01.a2 ... 250
32.1 Kern ... 250
33 A.06.01.a ... 251
33.1 Qt-Unterstützung ... 251
33.1.1 Unterstützte Plattformen ... 251
33.1.2 Fehler- und Warnmeldungen ... 251
33.1.3 Besonderheiten von Qt ... 251
33.1.4 Unterschiede zu anderen IDM-Versionen (Kompatibilität) ... 251

33.1.4.1 Objekte ... 252
33.1.4.1.1 Canvas ... 252
33.1.4.1.2 Edittext ... 253
33.1.4.1.3 Filereq ... 253
33.1.4.1.4 Image ... 254
33.1.4.1.5 Menubox, Menuitem, Menusep ... 254
33.1.4.1.6 Messagebox ... 255
33.1.4.1.7 Poptext ... 255
33.1.4.1.8 Progressbar ... 255
33.1.4.1.9 Scrollbar ... 256
33.1.4.1.10 Setup ... 256
33.1.4.1.11 Spinbox ... 256
33.1.4.1.12 Statusbar ... 256
33.1.4.1.13 Tablefield ... 256
33.1.4.1.14 Toolbar ... 257
33.1.4.1.14.1 Verhalten ... 260
33.1.4.1.15 Window ... 261
33.1.4.2 Ressourcen ... 263
33.1.4.2.1 Mnemonics ... 263
33.1.4.2.2 Color ... 263
33.1.4.2.2.1 Farbbezeichner und Null-Farbe ... 263
33.1.4.2.2.2 Farbverläufe (Gradienten) ... 265
33.1.4.2.3 Cursor ... 267
33.1.4.2.4 Font ... 268
33.1.4.3 Attribute ... 269
33.1.4.3.1 colorname[integer] ... 269
33.1.4.3.2 cursorname[integer] ... 269
33.1.4.3.3 fontname[integer] ... 270
33.1.4.3.4 picture und picture_hilite ... 270
33.1.4.3.5 statushelp ... 270
33.1.4.4 Schnittstellenfunktion DM_PictureHandler ... 270
33.1.5 Bekannte Probleme und Einschränkungen dieser Version ... 271
33.1.5.1 Qt 4.6 ... 271
33.1.5.2 Rahmen ... 271
33.1.5.3 Ereignisse ... 272
33.1.5.4 Drag-and-Drop ... 272
33.1.5.5 Startoption -IDMserver ... 272
33.2 Border-Attribute ... 272
33.2.1 Neuerungen ... 272
33.2.2 Rahmengestaltung ... 273
33.2.2.1 Kompatibilität bzgl. Gestaltung ... 274
33.2.2.2 Motif-Option .options[opt_xmborder_compat] ... 275
33.2.2.3 WSI-Einschränkungen und -Besonderheiten ... 275
33.2.2.3.1 Motif ... 275
33.2.2.3.2 Windows ... 275
33.2.2.3.3 Qt ... 276
33.2.3 Geometrie ... 276
33.2.3.1 Kompatibilität bzgl. Positionierung ... 278

33.2.3.2 Startoption -IDMborder5_compat ... 278
33.2.4 Attribut .borderstyle ... 279
33.2.5 Ergänzungen im Editor ... 281
33.3 Neue Objekte datetime und listview ... 282
33.3.1 datetime ... 282
33.3.1.1 Definition ... 284
33.3.1.2 Beschreibung der Ereignisse ... 285
33.3.1.2.1 deselect ... 285
33.3.1.2.2 select ... 285
33.3.1.3 Attributübersicht ... 286
33.3.1.4 Beschreibung der Attribute ... 287
33.3.1.4.1 .allowundefined ... 287
33.3.1.4.2 .calendaralignment ... 287
33.3.1.4.3 .format ... 288
33.3.1.4.4 .maxvalue ... 290
33.3.1.4.5 .minvalue ... 291
33.3.1.4.6 .real_modfied ... 292
33.3.1.4.7 .shortdaynames ... 292
33.3.1.4.8 .style ... 293
33.3.1.4.9 .today ... 294
33.3.1.4.10 .todaymarker ... 294
33.3.1.4.11 .trailingdates ... 295
33.3.1.4.12 .value ... 296
33.3.1.4.13 .weeknumbers ... 297
33.3.2 listview ... 298
33.3.2.1 Definition ... 301
33.3.2.2 Beschreibung der Ereignisse ... 302
33.3.2.2.1 activate ... 302
33.3.2.2.2 dbselect ... 302
33.3.2.2.3 deactivate ... 302
33.3.2.2.4 resize ... 302
33.3.2.2.5 select ... 302
33.3.2.3 Attributübersicht ... 303
33.3.2.4 Beschreibung der Attribute ... 304
33.3.2.4.1 .colcount ... 304
33.3.2.4.2 .coltitle[integer] ... 305
33.3.2.4.3 .colwidth[integer] ... 305
33.3.2.4.4 .content[index] ... 306
33.3.2.4.5 .mincolwidth[integer] ... 307
33.3.2.4.6 .picheight ... 308
33.3.2.4.7 .picture[integer] ... 308
33.3.2.4.8 .picwidth ... 309
33.3.2.4.9 .rowcount ... 310
33.3.2.4.10 .selected[integer] ... 311
33.3.2.4.11 .smallpicheight ... 311
33.3.2.4.12 .smallpicture[integer] ... 312
33.3.2.4.13 .smallpicwidth ... 313
33.3.2.4.14 .style ... 314

33.4 Erweiterung der USW-Schnittstelle ... 315
33.4.1 UC_InqName ... 315
33.4.2 UC_API ... 315
33.4.3 UC_InqArg ... 316
33.4.4 UC_Replace ... 316
33.4.5 getToolkitDataEx ... 316
33.4.6 UC_Inquire ... 317
33.5 Datentypen für Sammlungen ... 317
33.5.1 Erweiterung der Datentypen ... 317
33.5.2 Erweiterung der Ausdrücke ... 318
33.5.3 Listengröße und Standardwerte ... 319
33.5.4 Zugriff und Zuweisungen von Sammlungen ... 319
33.5.5 Automatische Konvertierung ... 320
33.5.6 Performanz ... 320
33.5.7 Lokale und globale Variablen ... 321
33.5.8 Vergleichsoperatoren ... 321
33.5.9 Uninitialisierte Variablen ... 322
33.5.10 Neue und geänderte eingebaute Funktionen ... 323
33.5.10.1 itemcount() ... 323
33.5.10.2 countof() ... 324
33.5.10.3 indexat() ... 325
33.5.10.4 valueat() ... 326
33.5.10.5 keys() ... 327
33.5.10.6 values() ... 328
33.5.10.7 join() ... 330
33.5.10.8 find() ... 332
33.5.10.9 sort() ... 334
33.5.10.10 min() ... 336
33.5.10.11 max() ... 337
33.5.10.12 getvector() ... 338
33.5.10.13 setvector() ... 340
33.5.11 Schleifenkonstrukt foreach ... 342
33.5.12 C-Schnittstellenerweiterung ... 343
33.5.12.1 Ergänzung der DM_Value-Struktur um Sammlungsdatentypen ... 343
33.5.13 Neue und geänderte C-Schnittstellenfunktionen ... 345
33.5.13.1 DM_ValueInit ... 345
33.5.13.2 DM_ValueChange ... 349
33.5.13.3 DM_ValueGet ... 352
33.5.13.4 DM_ValueCount ... 354
33.5.13.5 DM_ValueIndex ... 357
33.5.13.6 DM_ValueReturn ... 358
33.5.13.7 DM_StringInit ... 360
33.5.13.8 DM_StringChange ... 362

33.5.13.9 DM_StringReturn ... 364
33.5.13.10 DM_IndexReturn ... 365
33.6 Datenmodell ... 367
33.6.1 Motivation ... 367
33.6.2 Einführung ... 367
33.6.3 Kopplung zwischen Model und View ... 371
33.6.4 Reihenfolge und Werteaggregation ... 375
33.6.5 Synchronisierung zwischen Model und View ... 383
33.6.6 Konvertierung und Konvertierungsmethoden ... 386
33.6.7 Aktionen ... 388
33.6.8 Ablaufverfolgung ... 389
33.6.9 Restriktionen ... 389
33.6.10 Attributklassifikation und Datenmodellattribute ... 389
33.6.10.1 .datamodel ... 389
33.6.10.2 .dataget ... 390
33.6.10.3 .dataset ... 391
33.6.10.4 .dataindex ... 391
33.6.10.5 .datamap ... 392
33.6.10.6 .dataoptions ... 392
33.6.11 Datenmodellmethoden ... 393
33.6.11.1 :apply() ... 393
33.6.11.2 :collect() ... 395
33.6.11.3 :propagate() ... 397
33.6.11.4 :represent() ... 400
33.6.11.5 :retrieve() ... 404
33.6.11.6 :calldata() ... 405
33.6.12 Datenfunktion ... 407
33.6.12.1 Regelsprache ... 407
33.6.12.2 C-Schnittstelle ... 408
33.6.12.3 Beispiel ... 408
33.6.12.4 Datenfunktions-Struktur DM_DataArgs ... 413
33.6.13 Applikationsfunktion DM_DataChanged ... 414
33.7 Kompatibilität ... 415
33.8 Unix ... 415
33.9 Motif ... 416
33.10 Windows ... 416
33.10.1 Änderung von .toolkit_version unter Microsoft Windows ... 416
33.10.2 Anmerkung zum Window-Objekt ... 417
33.11 Kern ... 417
33.12 Editor ... 419

Index ... 421

1 A.06.03.b

1.1 SVG-Support

Unter MICROSOFT WINDOWS und QT unterstützt die **tile-Ressource** nun auch Vektor-Bilder im SVG-Format. Siehe hierzu Kapitel SVG-Support im „Ressourcenreferenz“-Handbuch.

1.1.1 Qt

Zwar konnte der IDM für Qt bereits SVG-Bilder anzeigen, allerdings nur über ein pixelbasiertes Stellvertreterbild das entsprechend skaliert wurde. Die Dateiendung .svg wird jetzt als SVG-Bildformat erkannt und bei der Darstellung in den diversen Objekten ein vektorbasiertes Zeichnen angewandt. Aus Gründen der Performanz wird beim Zeichnen von Hintergrund-Tiles in Gruppierungsobjekten im Stil *tilestyle_tiled* das pixelbasierte Stellvertreterbild verwendet.

1.1.2 Windows

MICROSOFT DIRECT2D wird verwendet um die Unterstützung der Grafikformate zu gewährleisten. Die SVG-Unterstützung von MICROSOFT DIRECT2D ist abhängig von der Microsoft Windows Version. Ab dem WINDOWS 10 CREATORS UPDATE ist die Unterstützung in eingeschränkter Form gegeben.

1.2 Ergänzung zur Dumpstate-Ausgabe

1.2.1 WSI-Dumpstate

Je nach Fenstersystem (WINDOWS/MOTIF/QT) unterscheidet sich die Ausgabe. In den meisten Fällen wird eine Auflistung der Bildschirme mit Auflösung, DPI-Zahl und Skalierung angezeigt.
Die Anzeige erfolgt nur für eine ungekürzte Ausgabe (**dump_full** oder **dump_wsi**) oder bei einem Fehlerverdacht der WSI-Komponente.

1.2.1.1 MOTIF / QT

Unter MOTIF/QT wird der Default-Screen mit "*" markiert. Farbtyp, Farbanzahl sowie die zur Verfügung stehende Bildschirmgröße wird angezeigt. Darüber hinaus werden die DPI-Werte und die Skalierungsstufe in Prozent angegeben.
Bitte beachten: Der IDM für MOTIF zeigt nur die Screens des benutzten Displays an.

BeispielMOTIF

```
WSI:

 screen coltype  colors   width  height    xdpi   ydpi   scale
------- ------- ------- ------- ------- ------- ------- -------
*     0   color     256    1864    1053      96      96     200
```

1.2.1.2 Windows

Unter MICROSOFT WINDOWS werden im WSI-Bereich der Dumpstate-Ausgabe folgende Werte ausgegeben:

- Version der *comctl32* und Status der Visual Styles.
- Informationen zur DPI-Awareness:
  - DPI-Awareness (on/off) des IDM und der aktuelle DPI-Awareness-Wert der Anwendung.
  - System DPI-Wert, den der IDM erhält wenn er den Wert abfragt sowie den reale Wert.
  - DPI-Wert des primären Monitors, den der IDM bei Abfrage erhält sowie den realen Wert.
- Verschiedene Zähler:
  - Anzahl der verwendeten GDI-Objekte und deren Höchstwert.
  - Anzahl der verwendeten Oberflächenobjekte und deren Höchstwert.
  - Anzahl der verwendeten Process-Handles.
- Informationen zu den jeweiligen Monitoren:
  - IDM-Koordinaten des Monitors.
  - System-Koordinaten des Monitors.
  - Skalierungsfaktor des Monitors.

Der primäre Monitor ist mit "*" gekennzeichnet. Außerdem werden die Werte für den virtuellen Bildschirm, der über alle Monitore gelegt wird, angegeben.

Beispiel

```
WSI:
comctl32 6.16.10586.3672, platformid=2, visual styles enabled
dpi awareness: on (real: on), system dpi = 144 (real: 144), primary dpi = 144
(real: 144)

     counter    used    peak
------------ ------- -------
 gdi objects      78      91
user objects      94      98
     handles     286

monitor   idm_x   idm_y   idm_w   idm_h  real_x  real_y  real_w  real_
h     dpi
```

```
------- ------- ------- ------- ------- ------- ------- ------- ------- -----
--
*     0       0       0    2560    1392       0       0    3840    2088     1
44
virtual       0       0    2560    1392       0       0    3840    2088     1
44
```

1.2.2 EXTERN-Dumpstate

Informationen die eine externe dump-Bibliothek, die bei Bedarf vom IDM-Support bereitgestellt wird, ausgibt.

Diese Zusatzinformation ist nicht vorgegeben. Die Anzeige erfolgt nur für eine ungekürzte Ausgabe (**dump_full** oder **dump_extern**) und benötigt eine dynamische Zusatzbibliothek. Sie dient dazu um zusätzlich dynamische Dumpstate-Informationen möglich zu machen.

Je nach System ist dazu eine .dll/.sl/.so-Datei mit Basisnamen "idmxdumpstate" notwendig. Diese Datei muss auf der selben Ebene wie das Applikationsverzeichnis liegen. Außerdem benötigt die Datei eine zugängliche C-Funktion **char* getDumpStateInfo()** welche die Information als ASCII oder UTF8-String zurück liefert.

1.3 Kompatibilität

1.3.1 Windows

Hinweise zum Bau einer Anwendung - Binden (link)

Um Scalable Vector Graphics zu unterstützen wird Microsoft Direct2D verwendet. Deshalb müssen beim Binden der Anwendung die Bibliotheken "**d2d1.lib**" und "**shlwapi.lib**" nach den IDM-Bibliotheken angegeben werden (siehe IDM-Beispiele).

1.3.2 Qt

Installationshinweis: Das Qt-Paket **qt5-qtsvg** muss bereits installiert sein um die entsprechende SVG-Unterstützungs-Bibliothek verfügbar zu machen und die Startfähigkeit einer IDM-Anwendung zu gewährleisten.

1.4 Motif

IDM-13249: Das interaktive Resizing von **Tablefield**-Zellen in Breite/Höhe berücksichtigt nun den Vergrößerungsfaktor des Bildschirms korrekt.

IDM-13266: Beim Schließen eines Fenster mit Toolbar bleiben die Abmessungen (.width/.height) nun bestehen.

IDM-13267: Diverse Positionierungsfehler beim interaktiven Verschieben von eingedockten Toolbars bereinigt. Beim Beenden des Verschiebevorgangs wird nun die Mausposition herangezogen anstatt des Zentrums des Toolbarrahmens.

1.5 Qt

Behoben: Die generierte MakeDefs-Datei für Qt enthält nun auch die Qt5Svg-Bibliothek für das Linken.

Neu: Die **Tile-Ressource** unterstützt nun auch Vektor-Bilder im SVG-Format, s. hierzu weiter oben **SVG-Support**.

1.6 Windows

IDM-13277: Es konnte einen Absturz geben wenn die UI Automation-Schnittstelle eines poptext-Objekts durch das Öffnen der Poptextliste angelegt wurde. Beobachtet wurde das bei eingeschaltetem Textcursor-Indikator. Der Fokus durfte nicht vorher auf dem poptext-Objekt liegen, denn dann wären die UI Automation-Schnittstellen bereits angelegt.

Behoben: Der IDM konnte sofort mit einem Speicherfehler abstürzen, wenn mehrere Monitore verwendet wurden.

Behoben: Es konnte vorkommen, dass die Tabgrößen eines Notebook-Objekts nicht in der Größe angepasst wurden, wenn das Fenster auf einen anderen Monitor mit unterschiedlichem Vergrößerungsfaktor verschoben wurde (nur IDM für Windows 11).

IDM-13253: Eine Dialogbox wurde mit falscher Größe geöffnet, wenn das zugeordnete Hauptfenster auf einem anderen Monitor dargestellt wurde. Das Problem trat mit dem IDM für Windows 11 auf, wenn beide Monitore unterschiedliche Vergrößerungen (DPI-Werte) besaßen.

IDM-13166: Es gab eine Fehlermeldung wenn *.hsb_visible* an einem einzeiligen **edittext**-Objekt auf den Wert *false* gesetzt wurde. Beim Setzen von Attributen werden nur noch dann Fehler in die Logdatei geschrieben, wenn der Wert nicht gesetzt werden kann. Bei Attributen, die nicht unterstützt werden, führt jetzt ein Setzen auf den internen Default zu keiner Fehlermeldung.

IDM-12444 Das Verhalten der Installation war verwirrend, wenn man im Options-Dialog das Installationsverzeichnis einer Vorgängerversion ausgewählt hat. Dann waren die **checkbox**-Objekte für das Ersetzen *aktiv* und *insensitiv*. Man hätte im Options-Dialog ein anderes Verzeichnis wählen müssen, um diese Objekte wieder *sensitiv* zu setzen. Jetzt bleiben die erwähnten **checkbox**-Objekte *sensitiv* und das Installationsverzeichnis wird implizit geändert, wenn diese umgeschaltet werden.

IDM-12442: Die Vorgängerversion des IDM wurde unbeabsichtigt gelöscht, wenn man bei der Installation der neuen IDM-Version im "Installationsart auswählen" Dialog die Checkbox "Ersetze ISA Dialog Manager ..." zuerst aktiviert und anschließend wieder deaktiviert hatte.

IDM-12439: Startmenüeinträge wurden bei der Deinstallation des IDM nur teilweise entfernt. Das Problem trat auf, wenn mehrere IDM-Versionen installiert waren.

IDM-13251: Es konnte vorkommen, dass nicht alle Kindobjekte eines **notepage**-Objekts angezeigt wurden, nachdem das Fenster auf einen anderen Monitor verschoben wurde. Das Problem trat auf, wenn nach dem Verschieben ein inaktives **notepage**-Objekt aktiviert wurde und auch nur dann wenn die Monitore unterschiedliche Skalierungsfaktoren besaßen und die HiDPI-Unterstützung aktiv war (nur IDM für WINDOWS 11).

IDM-13250: Wenn die Unterstützung der UI Automation aktiv war, konnte es zu Abstürzen kommen. Zum Beispiel wenn das *.text* Attribut eines **statictext**-Objekts geändert wurde. Im beobachteten Fall war das **statictext**-Objekt ein Kind eines **statusbar**-Objekts und wurde kurz unsichtbar und wieder sichtbar geschaltet. Außerdem werden die "*UI property not registered*" Fehlermeldungen für **statictext**-Objekte in einem **statusbar**-Objekt nicht mehr angezeigt.

Behoben: Die Debugversion des bodysim Projekts ließ sich wegen einer Kompatibilitätsprüfung beim Binden nicht mehr bauen.

Mögliche Probleme beim Erstellen von Debugversionen mit MICROSOFT VISUAL C/C++ 2022.
Beim Bau einer Debugversion kann es zu einem Linkerfehler LNK2038 kommen:

```
error LNK2038: Konflikt ermittelt für "_ITERATOR_DEBUG_LEVEL": Der Wert "0"
stimmt nicht mit dem Wert "2" in <Modul> überein.
error LNK2038: Konflikt ermittelt für "RuntimeLibrary": Der Wert "MD_
DynamicRelease" stimmt nicht mit dem Wert "MDd_DynamicDebug" in <Modul>
überein.
```

Das liegt daran, dass der MICROSOFT Compiler bei einem Debug-Bau implizit Einstellungen wählt, die inkompatibel zu einem Release-Bau sind. Um die beiden oben erwähnten Fehler zu umgehen, kann mit den IDM DLL's gelinkt werden.
Alternativ können folgende Präprozessorsymbole vor dem ersten *#include*-Ausdruck definiert werden:

```
#if defined(_DEBUG)
# define _ITERATOR_DEBUG_LEVEL 0
# define _ALLOW_RUNTIME_LIBRARY_MISMATCH
#endif
```

IDM-13235: Die **tile-Ressource** unterstützt nun auch die Bildformate "JPEG FILE INTERCHANGE FORMAT" (*.jpg), "PORTABLE NETWORK GRAPHICS" (*.png) und "SCALABLE VECTOR GRAPHICS" (*.svg). Hierzu wurde das Zeichnen auf DIRECT2D umgestellt. Da MICROSOFT DIRECT2D andere Skalierungsmethoden als das Windows GDI verwendet, können Muster, die zur Anzeige skaliert werden müssen, anders aussehen als in den Vorgängerversionen des IDM. Insbesondere gilt dies für Muster die als Fenstericon oder als Bild in den Objekten **treeview** und **notepage** verwendet werden, da MICROSOFT WINDOWS hier den Icondatentyp verlangt (s. auch Kapitel "SVG-Support" weiter oben).
Best Practice: Muster sollten bereits in der richtigen bzw. verwendeten Größe vorliegen.
Die Funktionen DM_GetToolkitData und DM_GetToolkitDataEx liefern aus Kompatibilitätsgründen weiterhin die GDI Datentypen zurück, wenn an der tile Ressource die Attribute AT_Tile, AT_XTile oder AT_wsidata abgefragt werden. Die hierfür nötigen GDI-Objekte werden

bei der Abfrage zusätzliche angelegt. Um die neuen MICROSOFT DIRECT2D-Datentypen zu erhalten muss die Funktion DM_GetToolkitDataEx verwendet werden. Siehe hierzu DM_GetToolkitDataEx.

Behoben: Wenn eine **tile-Ressource** skaliert werden musste und deswegen nach einer skalierten Datei (*@2x.*) gesucht wurde, dann wurde die Originaldatei nicht mehr geladen.

Behoben: Es wurde beobachtet, dass IDM Anwendungen als Administrator ausgeführt wurden. Um das zu verhindern wurde die Manifest-Datei um einen "trustinfo" Eintrag erweitert (siehe auch IDM-Beispiele):

```
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
        <security>
                <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
                        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
                </requestedPrivileges>
        </security>
</trustInfo>
```

IDM-12386: Wenn bei der Installation des IDM ausgewählt wurde, dass die Vorgängerversion deinstalliert bzw. überschrieben werden sollte, dann funktionierte die Installation nicht richtig.
In früheren Versionen blieb die alte Version erhalten, in aktuelleren wurde die Online-Doku in ein falsches Verzeichnis installiert. Das wurde korrigiert. Außerdem erscheint eine Warnung jedes Mal wenn man ein nicht unterstütztes Überschreiben aktiviert. Dafür erscheint diese Warnung dann nicht nochmal wenn die Installation ausgeführt wird.

IDM-11753: Multimonitor-Support besser an WINDOWS angepasst. Der IDM für WINDOWS 11 übernimmt nun die Monitorkoordinaten die WINDOWS einer Anwendung mitteilt, die nicht an hohe Auflösungen angepasst ist. Hierdurch erscheinen IDM-Fenster im IDM für WINDOWS 10 und 11 in den meisten Fällen an denselben Positionen.

IDM-13194: Nach einer Drag und Drop Aktion erschien die Fehlermeldung "WSI error processing clipboard.value". Voraussetzung war, dass eine *:setclip* Methode existierte, die das Clipboard änderte.

1.7 Kern

Neu: Die Plattformen RHEL8.1 und RHEL9 (QT & MOTIF) unterstützen nun die IDM Plugins für ECLIPSE (ECLIPSE4.24, LINUX-5). Die Sprachdefinitionsdatei ***idmlangdef.xml***, die Sucheunterstützung ***IDMParser.jar*** wie auch das ***IDM Help Plugin*** werden jetzt in der Distribution mit ausgeliefert.

Die jeweiligen Ablageorte im Installationsbaum:

- lib/IDM/idmlangdef.xml
- lib/IDM/IDMParser.jar
- lib/(doc-de|doc-en)/eclipse/de.isa.idmdoc.eclipseplugin_*.jar

Das IDM Help Plugin (Datei: *lib/(doc-de|doc-en)/eclipse/de.isa.idmdoc.eclipseplugin_*.jar*) muss in das *dropins*-Verzeichnis der entsprechenden ECLIPSE-Version kopiert werden. Weitere Informationen finden sich im Kapitel „Kontextsensitive Hilfe“ (Pfad: *Hilfe -> Hilfeinhalt -> IDM-Plugin Benutzerdokumentation -> Referenz -> Editor -> Kontextsensitive Hilfe*) des IDM-Eclipse-Plugins.

Neu: Erweiterung der Dumpstate-Ausgabe unter MOTIF, QT und WINDOWS, siehe weiter unten.

Behoben: Readme(s) hinsichtlich Sprachfehler (Englisch) und Darstellung angepasst.

IDM-12752: Zuweisung von text-Ressourcen auf String-Variablen. Die Zuweisung von text-Ressourcen auf String-Variablen wird nun als String gespeichert und nicht wie zuvor fälschlicherweise als text-ID.

Achtung

Durch diese Korrektur kann es unter Umständen zu Verhaltensänderungen kommen. Das folgenden Beispiel skizziert eine mögliche Verhaltensänderung:

```
 ...
text TxHello "Hello";
 ...
record Rec {
        string Assoc[anyvalue];
        .Assoc[Tx] := "World";

        rule Print() {
                variable string Idx := Tx;

                print Idx + " " + Rec.Assoc[Idx];
        }
}
```

```
Der Aufruf von "Print" führt jetzt zu einer Fehlermeldung:

       ***ERROR IN EVAL: cannot get value
```

```
Während in Vorgängerversionen (fälschlicherweise) "Hello World" ausgegeben
wurde.
```

IDM-13261: Behoben sind Abstürze im Umfeld von dynamisierten COBOL-Applikationsfunktionen oder fehlerhaften Übertragungen von Record-Inhalten im Zusammenhang mit verschachtelten Records.

Behoben: Eine Änderung des **.name** Attributs an der tile Ressource erhält nun den Zustand von *.scale* oder *.scalestyle*.

IDM-13242: Die Ereignisse *select*, *activate* und *deactivate* des Checkbox-Objekts werden nun vom Datenmodell als Änderung der Attribute *.state* und *.active* propagiert.

Objekte die im Datenmodellkonzept als Model fungieren, propagieren nun Änderungen, die durch Dialogereignisse verursacht werden, an die *Views*. Dies geschieht analog zu der Propagierung wegen einem *Setvalue*.

IDM-13244: Eine unter Umständen auftretende doppelte Codepage-Konvertierung von Argumenten eines **DM_QueueExtevent()**- oder **DM_SendEvent()**-Aufrufs wird nun vermieden.

IDM-13247: Die Verwendung eines indizierten Zugriffs auf ein Listenelement einer lokalen Variable oder eines Argumentparameters sollte nun als *input/output*-Argument korrekt möglich sein, inklusive Typanpassung and Zurückschreiben auf das indizierte Element.

PLUGIN-12417: Die Sprachbeschreibung *idmlangdef.xml* enthält nun bei Attributen einen Type mit Scope in [] (eckige Klammern) falls vorhanden. Dadurch wird die Vervollständigung des IDM-Eclipse-Plugin unterstützt und verbessert, sodass für vordefinierte Attribute die Zuordnung zu einem speziellen Objekttyp möglich ist (Bsp: .fgc => object[color]) und die Vervollständigungsliste klassenspezifisch erfolgen kann.

Behoben: Die vom Datenmodell durchgeführte Propagierung von vordefinierten Attributen bei einem Dialogereignis hatte fälschlicherweise einen Index erhalten.
Zum Beispiel bei der Propagierung von *.activeitem* einer **listbox**, wenn eine Auswahl durch einen Mausklick erfolgte.

IDM-13248: Die Dateisuche über "~:<Dateipfad>", die beim IDM für das Auffinden von Dialogen, Modulen, Interface-Dateien oder anderer Dateiarten verwendet wird, wurde ergänzt.
Typischerweise dient das "~"-Symbol als Platzhalter für das Verzeichnis in dem sich die Applikation oder das geladene Modul befindet.
Nun wird zusätzlich noch das Dialogverzeichnis herangezogen. Das vereinfacht das Auffinden von Interfaces und Modulen in modularisierten Dialogen.
Auch der Standard-Suchpfad bei Import über "use" erlaubt das Auffinden von Sub-Modulen die unterhalb des Dialog-Verzeichnisses liegen.

IDM-13203: Bei einem Poptext-Objekt mit Format geht nun eine *.activeitem*-Umsetzung die im unsichtbaren Zustand durchgeführt wird, nicht mehr verloren.

1.8 Editor

IDM-13248: Um die Simulation von modularisierten Dialogen zu vereinfachen, wird der konfigurierte Suchpfad sowie das Verzeichnis des simulierten Dialogs dem Simulationsprogramm als Suchpfad weitergegeben. Dadurch wird die Ladbarkeit von Sub-Modulen gewährleistet.

Behoben: Die Bildschirmpositionsprüfung beim Wiederherstellen der Fensterposition des IDM Editor-Hauptfensters berücksichtigt nun auch Multiscreen Set-ups.

1.9 DDM (Netz)

IDM-13212: Es konnte keine SSL-Verbindung aufgebaut werden, wenn die Anwendungsseite mit der Startoption "-IDMssl_option opt_verify_peer;opt_cert_required" gestartet wurde. Jetzt funktioniert die

Authentifizierung der Darstellungsseite, hierzu benötigt sie ihren privaten Schlüssel.

Damit der IDM die SSL-Verschlüsselung unterstützt, wird mindestens die Version 1.1.1 der OpenSSL-Bibliothek vorausgesetzt. Aus Sicherheitsgründen sollte jedoch eine aktuellere Version verwendet werden.
LINUX: Auf LINUX-Systemen findet man oft eine "libssl3.so" Bibliothek, hierbei handelt es sich nicht um die Version 3.0.0 von OpenSSL.

Behoben: Wenn es zu einem Netzwerkfehler kam, während die Netzwerkprotokollversion synchronisiert wurde, wurde fälschlicherweise die Fehlermeldung ausgegeben, dass die Netzwerkprotokollversion nicht übereinstimmt. Jetzt wird nur noch der eigentliche Fehler protokolliert.

IDM-12970: Wenn entweder die Zertifikatsdatei oder die private Schlüsseldatei nicht geladen werden konnte, erhielt man nur die Fehlermeldung "no shared cipher". Jetzt wird zusätzlich eine Fehlermeldung ausgegeben, wenn eine der Dateien nicht geöffnet werden konnte.

IDM-13227: Klarstellung der Suche nach der SSL DLL.

Der IDM versucht die SSL DLL des Systems dynamisch zu laden. Hierzu wird immer ein versionsunabhängiger Name verwendet. Bei richtiger Installation, wird die aktuellste Version der SSL-Unterstützung über den Namen herangezogen. Um spezielle Versionen verwenden zu können, sucht der IDM zunächst in Verzeichnissen, die vom Installationsverzeichnis der Anwendung abhängig sind.
Siehe hier drunter.

Für UNIX:
1. Im Installationsverzeichnis der Anwendung "."
2. Im Unterverzeichnis "./lib"
3. In "../lib" (wenn die Anwendung direkt in einem "bin" Verzeichnis installiert ist)

In diesen Verzeichnissen wird versucht folgende DLL zu laden: "**libssl**"
Falls notwendig wird zusätzlich versucht die "**libcrypto**" DLL zu laden.
Wird eine bestimmte Version dieser DLLs gewünscht, kann in entsprechender "*symbolic link*" in einem der erwähnten Verzeichnisse eingerichtet werden.

Für MICROSOFT WINDOWS:
Im Unterverzeichnis ".\dll"
Leider gibt es unter WINDOWS keine einheitliche, versionsunabhängige Bezeichnung, deshalb wird versucht folgende DLL's zu laden:
1. "libssl-3-x64" (falls notwendig "libcrypto-3-x64")
2. "libssl-1_1-x64" (falls notwendig "libcrypto-1_1-x64")
3. "libssl32" (falls notwendig "libeay32")
4. "ssleay32" (falls notwendig "libeay32")
5. "ssl"

Wird eine bestimmte Version dieser DLLs gewünscht, dann kann die gewünschte DLL unter einem der genannten Namen in das "**.\dll**" Verzeichnis kopiert werden.
Wird in den Anwendungsverzeichnissen nichts gefunden, werden die angegebenen DLLs im Systemsuchpfad gesucht.

Behoben: Die Serverlizenz führte keine Lizenzprüfung durch, damit war eine Testlizenz der Anwendungsseite auch nach Ablauf einsetzbar.

1.10 Debugger

Neu: Die Liste der im Regel-Browser angezeigten Regeln kann durch ein Filtermuster eingegrenzt werden. Außerdem fanden kleinere Verbesserungen an der Regelsuche statt.
Unter MOTIF gibt es Einschränkungen hinsichtlich der Anzahl anzeigbarer Zeilen im **Treeview**, weshalb die Darstellung unter MOTIF nun über eine listbox erfolgt. Bis zu 65.000 Regeln können somit dargestellt werden.

Um den Umgang beim Debuggen von Dialogen mit vielen Regeln zu verbessern, wurde der Regel-Browser durch einen Filter ergänzt. Wird ein Filtermuster angegeben erfolgt die Reduzierung der Anzeige auf die Regeln welche dem Filtermuster entsprechen.

Zusätzlich sind kleine Verbesserungen/Veränderungen an der Regelsuche vorgenommen worden. Zu Beginn erfolgt die Suche in der gefilterten Regelliste. Im Suchfenster wird die Trefferanzahl dargestellt. Mit der **F3-Taste** springt man zum nächsten Treffer. Der Regelname des Treffers wird nun ebenfalls im Suchfenster angezeigt.

Zusätzlich zur vorgebbaren Suchrichtung, ist nun ein Umbrechen der Suche möglich (standardmäßig AN).

Neben dem bekannten **F3-Accelerator** gibt es nun den **Ctrl-F3 Accelerator**, der die Suchrichtung umkehrt.

Das Regelsuch-Fenster ist nun außerdem einfach über **ESC** schließbar.

1.11 COBOL

Erweitert: Cobol-Protokollnummern eingeführt um Diskrepanzen zwischen verwendeter DM-Copy-Strecke und DM Cobol-Anbindung zu erkennen.

Die **DM-StdArgs**-Struktur ist deshalb um die folgenden Informationen zu der IDM-Version erweitert worden:

```
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.
...
```

Behoben: Es wurde bei einem Absturz kein Dumpstate ausgegeben, wenn die Micro Focus Cobol-Schnittstelle unter Windows benutzt wurde. Das lag daran, dass der Exceptionhandler des IDM

nicht aufgerufen wurde, da die Exception bereits von Cobol abgefangen wurde. Ein zusätzliches Microsoft structered exception handling eingefügt um einen Dumpstate zu erhalten.

Behoben: Mit dem Skript idmcob.bat konnte die Cobol-Anwendung nicht mit den IDM DLLs gebunden werden. Außerdem wurden die IDM-Optionen ndx und ole nicht unterstützt.

2 V.06.03.a5

2.1 Windows

IDM-13277: Es konnte einen Absturz geben wenn die **UIAutomation**-Schnittstelle eines poptext-Objekts durch das Öffnen der Poptextliste angelegt wurde. Beobachtet wurde das bei eingeschaltetem Textcursor-Indikator. Der Fokus durfte nicht vorher auf dem poptext-Objekt liegen, denn dann wären die **UIAutomation**-Schnittstellen bereits angelegt.

2.2 COBOL

Behoben: Bei einem Absturz wurde kein Dumpstate ausgegeben, wenn die MICRO FOCUS COBOL-Schnittstelle unter WINDOWS benutzt wurde.

3 V.06.03.a4

3.1 Hinweis zur WSI Dumpstate-Ausgabe unter WINDOWS

Unter MICROSOFT WINDOWS werden im WSI-Bereich der Dumpstate-Ausgabe folgende Werte ausgegeben:

» Version der *comctl32* und Status der Visual Styles.

» Informationen zur DPI-Awareness:
- DPI-Awareness (on/off) des IDM und der aktuelle DPI-Awareness-Wert der Anwendung.
- System DPI-Wert, den der IDM erhält wenn er den Wert abfragt sowie den reale Wert.
- DPI-Wert des primären Monitors, den der IDM bei Abfrage erhält sowie den realen Wert.

» Verschiedene Zähler:
- Anzahl der verwendeten GDI-Objekte und deren Höchstwert.
- Anzahl der verwendeten Oberflächenobjekte und deren Höchstwert.
- Anzahl der verwendeten Process-Handles.

Beispiel

```
WSI:
comct132 6.16.10586.3672, platformid=2, visual styles enabled
dpi awareness: on (real: on), system dpi = 144 (real: 144), primary dpi = 144
(real: 144)

     counter    used    peak
------------ ------- -------
 gdi objects      78      91
user objects      94      98
     handles     286
```

3.2 Kern

Neu: Erweiterung der Dumpstate-Ausgabe um folgende Bereiche:
- WSI: Informationen des Fenstersystems (MOTIF/QT/WINDOWS).
- EXTERN: Informationen die eine externe Dump-Bibliothek findet (wird typischerweise bei Bedarf vom IDM-Support bereitgestellt).
Diese zusätzlichen Dumpstate-Informationen werden nur bei *dump_full*/*dump_wsi*/*dump_extern* über das Builtin-Dumpstate(***<enum>***) oder die Optionen ***-IDMdumpstate=<enum>*** oder ***-IDMobjdump_dumpstate=<enum>*** ausgebeben.

3.3 Windows

IDM-13264: Es wurde ein GDI-Objekt-Leak beobachtet, das zu einer ständigen Verlangsamung der Anwendung und zu Zeichenproblemen führte.

Das Problem trat in folgenden Fällen auf:
- Ein Bild aus einer Datei geladen wurde und skaliert werden musste.
- Ein Bild, das eine Farbpalette besitzt, beim Zeichnen skaliert werden musste (Farbpaletten werden vor allem von GIF-Dateien verwendet).

4 V.06.03.a3

4.1 Windows

IDM-13251: Es konnte vorkommen, dass nicht alle Kindobjekte eines **notepage**-Objekts angezeigt wurden, nachdem das Fenster auf einen anderen Monitor verschoben wurde. Das Problem trat auf, wenn nach dem Verschieben ein inaktives **notepage**-Objekt aktiviert wurde und auch nur dann wenn die Monitore unterschiedliche Skalierungsfaktoren besaßen und die HiDPI-Unterstützung aktiv war (nur IDM für WINDOWS 11).

IDM-13250: Wenn die Unterstützung der **UI Automation** aktiv war, konnte es zu Abstürzen kommen; zum Beispiel wenn das *.text* Attribut eines **statictext**-Objekts geändert wurde. Im beobachteten Fall war das **statictext**-Objekt ein Kind eines **statusbar**-Objekts und wurde kurz unsichtbar und wieder sichtbar geschaltet. Außerdem werden die "*UI property not registered*" Fehlermeldungen für **statictext**-Objekte in einem **statusbar**-Objekt nicht mehr angezeigt.

Behoben: Wenn eine **Tile-Ressource** skaliert werden musste und deswegen nach einer skalierten Datei (*@2x.*) gesucht wurde, dann wurde die Originaldatei nicht mehr geladen.

4.2 Kern

IDM-13261: Behoben sind Abstürze im Umfeld von dynamisierten Cobol-Applikationsfunktionen oder fehlerhaften Übertragungen von Record-Inhalten im Zusammenhang mit verschachtelten Records.

5 A.06.03.a2

5.1 Qt

IDM-13223: Das Performanz-Problem bei der Anzeige von mehrzeiligen Edittexten mit großem Inhalt ist behoben. Es lag an der Unterstützung des .alignment-Attributes. Der IDM für Qt unterstützt das .alignment-Attribut am Edittext. Dabei ist zu beachten, dass ein Alignment von 0,-1 (zentriert, rechtsbündig) bei größeren Textmengen zu deutlichen Zeitverzögerungen bei der Anzeige führt. HTML-Inhalte die über die Option **.option[opt_html] true;** aktiviert werden, können nicht in ihrer horizontalen Ausrichtung geändert werden, da das Layouting über HTML erfolgt.

5.2 Kern

IDM-12752: Die Zuweisung von text-Ressourcen auf String-Variablen wird nun als String gespeichert und nicht wie zuvor fälschlicherweise als text-ID.

6 A.06.03.a

6.1 HighDPI Unterstützung

Die Auflösung moderner Bildschirme wächst stetig. Damit ändern sich auch die Anforderungen an Design und Qualität der Bedienelemente und Oberflächen. Probleme sind dabei häufig, dass Anwendungen viel zu klein erscheinen und Text und Grafiken unscharf oder pixelig dargestellt werden. Die im System eingestellte Auflösung, DPI und Skalierung sowie Fonts und Größe und Qualität der Bilder beeinflussen das Darstellungsbild einer Anwendung. Damit bestehende Anwendungen auch auf hochauflösenden Bildschirmen detailreich und scharf dargestellt werden, müssen diese auf HighDPI vorbereitet werden.

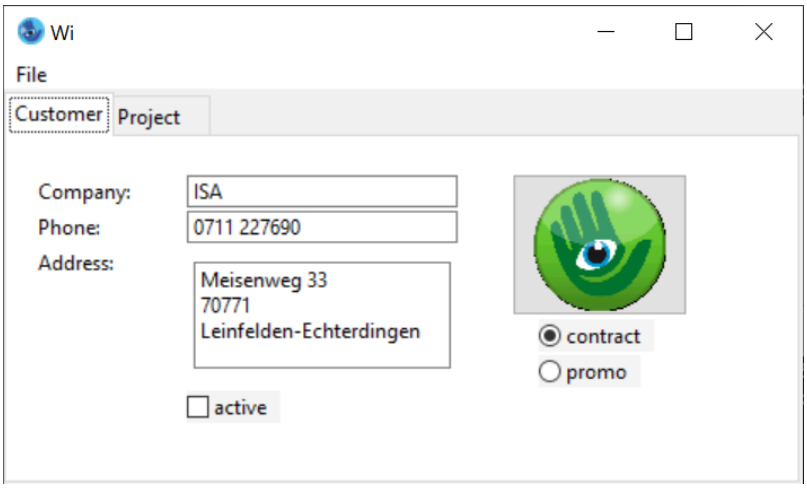


Abbildung 1: *ohne HighDPI Unterstützung*

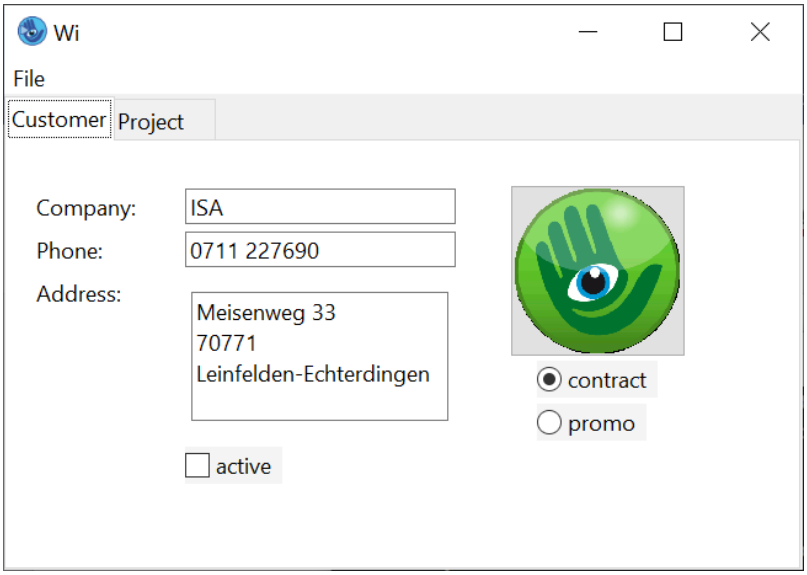


Abbildung 2: *mit HighDPI Unterstützung*

Der IDM für MOTIF, QT und MICROSOFT WINDOWS ab Version A.06.03.a (unter MICROSOFT WINDOWS nur der IDM FÜR WINDOWS 11) bietet nun die geeigneten Werkzeuge um hochauflösende Bildschirme zu unterstützen und Anwendungen zukunftssicher zu gestalten. Außerdem wurde das Arbeiten mit mehreren Bildschirmen verbessert.
IDM-Anwendungen beachten nun die DPI-Einstellungen und Skalierungsfaktoren des Systems. Bisherige Probleme wie pixelige, matschige oder zu klein angezeigte Anwendungen oder verringerte Schriftgrößen bei hohen Auflösungen und geringer Bildschirmgröße (die Effekte können je nach System variieren) treten somit nicht mehr auf. Der IDM passt Geometrie, Layout, Schriftgrößen, Grafiken sowie Cursor nun automatisch an die entsprechende Auflösung und vom System bestimmte DPI an. Der Skalierungsfaktor basiert dabei auf den Systemeinstellungen der jeweiligen Desktop-Umgebung. Es steht dem Anwender aber frei durch verschiedene neue Attribute, Optionen und Mechanismen andere Vorgaben zu machen bzw. gestalterisch einzugreifen.

6.1.1 Layout-Ressourcen

Bisher waren die verfügbaren, vordefinierten Ressourcen beim IDM immer WSI- bzw. systemabhängig und bedurften bei Übertragung auf andere Systeme immer aufwändiger Anpassungen. Der IDM A.06.03.a bietet nun erstmals WSI-unabhängige und HighDPI-konforme, vordefinierte Layout-Ressourcen an. Dazu gehören folgende Ressourcen, die auf allen Systemen in ähnlicher Ausprägung bzw. Bedeutung verfügbar sind: font (Zeichensatz), color (Farbe), cursor.
Das bietet den enormen Mehrwert, dass Dialoge, die komplett auf die neuen UI-Ressourcen umgestellt wurden, nun einfach und ohne Aufwand auf andere Systeme übertragen werden können.
Erkennbar sind diese Ressourcen über das Namensschema ***UI*_FONT***, ***UI*_COLOR*** bzw. ***UI*_CURSOR***.

Beispiel:

```
module Resc

font FontNormal "UI_FONT";
font FontBig "UI_FONT", 16;
font FontFixed "UI_FIXED_FONT";

cursor CurStop "UI_STOP_CURSOR";

!! Background color for group objects
color ColGrp "UI_BG_COLOR";
!! Background color for input objects
color ColInp "UI_INPUTBG_COLOR";
```

Die UI_-Ressourcennamen sind ebenfalls – wie alle anderen vordefinierten Ressourcennamen - am setup-Objekt über die Attribute .colorname[integer], .fontname[integer], .cursorname[integer] abfragbar.

Cursor

Die einheitlichen Cursor-Ressourcen haben das Namensschema ***UI*_CURSOR*** und sind wie folgt definiert:

UI_CURSOR	Standardcursor meist Pfeil
UI_IBEAM_CURSOR	Edittext-Einfügemarke
UI_WAIT_CURSOR	Anzeige, dass die Anwendung beschäftig ist, „Eieruhr“
UI_CROSS_CURSOR	Kreuz
UI_UP_CURSOR	Pfeil nach oben
UI_SIZEDIAGF_CURSOR	Sizing-Pfeil von links oben nach rechts unten
UI_SIZEDIAGB_CURSOR	Sizing-Pfeil von links unten nach rechts oben
UI_SIZEVER_CURSOR	Sizing-Pfeil von oben nach unten
UI_SIZEHOR_CURSOR	Sizing-Pfeil von links nach rechts
UI_MOVE_CURSOR	Verschiebe-Pfeil in alle Richtungen
UI_STOP_CURSOR	Verbotszeichen (durchgestrichener Kreis)
UI_HAND_CURSOR	Handsymbol (Zeigefinger dient zum Zeigen)
UI_HELP_CURSOR	Pfeil mit Fragezeichen (für Kontexthilfe-Modus)

Fonts

Die einheitlichen Font-Ressourcen haben das Namensschema ***UI*_FONT*** und sind wie folgt definiert:

UI_FONT	Schriftart, die standardmäßig für Oberflächenelemente verwendet wird (Default oder Dialog Font).
UI_FIXED_ FONT	Schriftart mit einer festen Buchstabenbreite.
UI_MENU_ FONT	Schriftart, die für Menüs verwendet wird.
UI_TITLE_ FONT	Schriftart, die in der Fensterüberschrift verwendet wird.
UI_ SMALLTITLE_ FONT	Schriftart, die in einer kleinen/niedrigen Fensterüberschrift verwendet wird.

UI_ STATUSBAR_ FONT	Schriftart, die in der Statusbar verwendet wird.
UI_NULL_FONT	Verhalten als wenn keine Font-Vorgabe gemacht würde (null). Das WSI benutzt den für das Oberflächenelement vorgesehenen Standard-Systemfont oder einen Systemfallback-Font.

Color

Die Farb-Ressourcen haben das Namensschema ***UI*_COLOR*** und sind wie folgt definiert:

UI_BG_ COLOR	Farbe, die standardmäßig für allgemeine Hintergründe von (meist Gruppierungs-) Objekten verwendet wird. Verwendung bei Objektklassen mit Attribut .bgc/.bgc[], z.B. Fenster, Groupbox, Notepage, Statictext, Pushbutton...
UI_FG_COLOR	Farbe, die standardmäßig für allgemeine Vordergründe/Textfarbe von (meist Gruppierungs-) Objekten verwendet wird. Verwendung bei Objektklassen mit Attribut .fgc/.fgc [], z.B. Fenster, Groupbox, Notepage, Statictext, Pushbutton...
UI_INPUTBG_ COLOR	Farbe, die standardmäßig bei Hintergründen von Eingabe-/Auswahl-Objekten verwendet wird. Verwendung z.B. bei Edittext, Listen, Tabellen, Poptext, Treeview…(z.B. Objektklassen mit Attributen .bgc/.textbgc)
UI_INPUTFG_ COLOR	Farbe, die standardmäßig bei Vordergründen von Eingabe-/Auswahl-Objekten verwendet wird, wie z.B. Edittext, Listen, Tabellen, Poptext, Treeview (z.B. Objektklassen mit Attributen .fgc/.textfgc)
UI_ BUTTONBG_ COLOR	Farbe, die standardmäßig beim Hintergrund von Button-artigen Objekten, z.B. Pushbutton oder Image, verwendet wird (z.B. Objektklassen mit Attribut .bgc/.imagebgc).
UI_ BUTTONFG_ COLOR	Farbe, die standardmäßig beim Vordergrund von Button-artigen Objekten, z.B. Pushbutton oder Image, verwendet wird (z.B. Objektklassen mit Attribut. fgc/.imagefgc).
UI_BORDER_ COLOR	Farbe, die standardmäßig als Rahmen, Rand oder Begrenzung verwendet wird (z.B. an .bordercolor).
UI_ ACTIVEBG_ COLOR	Farbe, die standardmäßig als Hintergrundfarbe des aktiven Elements z.B. bei Listen, Poptext, Tabellen, ggf. Progressbar, Markierungen etc. verwendet wird. Meist Invertierung der normalen Vorder- und Hintergrundfarben des Widgets (z.B. Objektklassen mit Attribut .bgc).

UI_ ACTIVEFG_ COLOR	Farbe, die standardmäßig als Vordergrund-/Textfarbe des aktiven Elements z.B. bei Listen, Poptext, Tabellen, ggf. Progressbar, Markierungen etc. verwendet wird. Meist Invertierung der normalen Vorder- und Hintergrundfarben des Objekts (z.B. Objektklassen mit Attribut .fgc).
UI_TITLEBG_ COLOR	Farbe, die standardmäßig als Hintergrundfarbe bei Titelleisten oder Umrandungen von Fenstern/Dialogen verwendet wird, sofern unterstützt (z.B. an Attribut .titlebgc).
UI_TITLEFG_ COLOR	Farbe, die standardmäßig als Vordergrund-/Textfarbe bei Titelleisten oder Umrandungen von Fenstern/Dialogen verwendet wird, sofern unterstützt (z.B. an Attribut .titlefgc).
UI_MENUBG_ COLOR	Farbe, die standardmäßig als Hintergrundfarbe bei Menüs verwendet wird, sofern unterstützt (z.B. an Attribut .titlebgc).
UI_MENUFG_ COLOR	Farbe, die standardmäßig als Vordergrund-/Textfarbe bei Menüs verwendet wird, sofern unterstützt (z.B. an Attribut .titlefgc).
UI_ HEADERBG_ COLOR	Farbe, die standardmäßig als Hintergrundfarbe bei Tabellenheadern – sofern unterstützt – verwendet wird.
UI_ HEADERFG_ COLOR	Farbe, die standardmäßig als Vordergrund-/Textfarbe bei Tabellenheadern – sofern unterstützt – verwendet wird. Verwendung bei Tablefield (z.B. an Attribut .rowheadfgc, .cloheadfgc).
UI_NULL_ COLOR	Verhalten, als wenn keine Farbe (null) gesetzt wäre. WSI zeichnet nach seiner Standardpalette.

Es kann vorkommen, dass sich nicht alle UI*COLORS voneinander unterscheiden. Das liegt daran, dass die meisten Desktopfarben aus wenigen Basisfarben "berechnet" werden.

Hinweise Motif:

Neben den ***UI*_COLOR*** Farben wurden die Systemnamen für Farb-Ressourcen vervollständigt. Neu hinzugekommen sind:

- INACTIVE_BACKGROUND
- INACTIVE_FOREGROUND
- INACTIVE_TOP_SHADOW
- INACTIVE_BOTTOM_SHADOW
- INACTIVE_SELECT
- SECONDARY_BACKGROUND

- SECONDARY_FOREGROUND
- SECONDARY_TOP_SHADOW
- SECONDARY_BOTTOM_SHADOW
- SECONDARY_SELECT

Hinweise Windows:

Die ***UI*_COLOR*** Farben greifen auf die Windows Systemfarben zu. Hierbei ist zum einen zu beachten, dass mit Einführung der Visual Styles von den einzelnen Objekten nicht mehr die Systemfarben, sondern die durch das Theme definierten Farben verwendet werden. Je nach ausgewähltem Theme kann es somit starke Diskrepanzen geben. Deshalb sollten unter Windows nach Möglichkeit keine Farben (oder besser die ***UI_NULL_COLOR***) gesetzt werden.
Zum anderen ist zu beachten, dass mit Windows 10 die Farbvielfalt verringert wurde. Die Farben ***UI_HEADERBG_COLOR***, ***UI_HEADERFG_COLOR***, ***UI_MENUBG_COLOR, UI_MENUFG_COLOR***, ***UI_TITLEBG_COLOR*** und ***UI_TITLEFG_COLOR*** werden zum Beispiel nicht mehr unterstützt. Das heißt, dass diese Farben für den Anwender zwar noch verfügbar und nutzbar sind, sie aber aktuell von keinem Objekt defaultmäßig verwendet werden.

Hinweise Qt:

Die ***UI*_COLOR*** Farben greifen auf die von Qt angebotenen Farben der Standard-Palette zu. Für die Farbausprägung wird dabei die ColorGroup Normal bzw. Active zugrunde gelegt. Die Farben der Standard-Palette wiederum werden durch den aktuell eingestellten Systemstyle vorgegeben.

Neben den ***UI*_COLOR*** Farben wurden die vordefinierten Farben um weitere vom Qt-Toolkit angebotenen ColorRoles vervollständigt. Neu hinzugekommen (in allen Ausprägungen der ColorGroups) sind:

- BRIGHTTEXT
- ALTERNATEBASE
- TOOLTIPBASE
- TOOLTIPTEXT
- LINK
- LINKVISITED
- PLACEHOLDERTEXT

6.1.2 Erweiterung an der Tile-Ressource

Die bestehenden Skalierungsoptionen an der Tile-Ressource wurden erweitert. Es kann nun ein .scalestyle angegeben werden, der bestimmt wie ein Tile skaliert werden soll. So können Muster (tiles) nun nach folgenden Merkmalen skaliert werden:

- automatisch
- um einen bestimmten Faktor
- proportional

» in alle Richtungen frei

» an den DPI-Wert anpassen und keine Skalierung

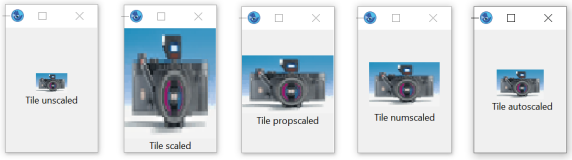


Abbildung 3: *das gleiche Bild mit verschiedenen Ausprägungen des Attributs .scalestyle bei einer Systemskalierung von 150%*

In der Regelsprache stehen dabei folgende Werte zur Auswahl:

Definition am Tile	Dynamische Setzung	Bedeutung
noscale	scalestyle_ none	Das Muster bzw. Bild wird nicht skaliert., setup.tiledpi hat keine Auswirkungen.
scale	scalestyle_any	Höhe und Breite des Musters bzw. Bildes werden voll auf die verfügbare Fläche vergrößert.
propscale	scalestyle_ prop	Höhe und Breite des Musters bzw. Bildes werden auf die verfügbare Fläche vergrößert, wobei Höhen- und Breitenproportionen des Musters bzw. Bildes auf jeden Fall erhalten bleiben. D.h. es können oben und unten bzw. links und rechts Freiflächen entstehen.
numscale	scalestyle_ num	Die Skalierung des Musters bzw. Bildes wird durch einen numerischen Skalierungsteiler vorgenommen. Dabei wird die Skalierung in Viertel-Schritten vorgenommen, also 1.25-fach, 1.5-fach, 1.75-fach, 2-fach, 2.25-fach, 2.5-fach, usw. Eine Verkleinerung finded bis maximal 0.25-fach statt.
dpi	scalestyle_dpi	Das Muster bzw. Bild wird immer entsprechend der eingestellten Bildschirmskalierung skaliert.
Keine Angabe	scalestyle_ auto	Das Muster bzw. Bild wird entsprechend der eingestellten Bildschirmskalierung skaliert. Eine zur Vorgängerversion kompatible Skalierung findet statt. **Defaultwert**

Der ***scalestyle*** kann dabei entweder direkt bei der **Definition des Tiles** oder aber als **dynamisches Attribut** angegeben werden.

Tile-Definition:

```
tileSpec ::= <tileBitmap> | "<Dateipfad>" | "<Grafikressource>" { scale |
noscale | numscale | propscale | dpiscale} ;
```

Dynamisches Attribut:

	Bezeichner		Datentyp
Regelsprache	**.scalestyle**		enum
C	AT_scalestyle		DT_enum
COBOL	AT-scalestyle		DT-enum
Klassifizierung	Objektspezifisches Attribut		
Objekte	tile		
Zugriff	get, set	Standardwert	scalestyle_auto
changed-Ereignis	nein	Vererbung	–

Beispiel:

```
dialog D
tile Ti_Rect_Pattern 5,5,
  "#####",
  "# #",
  "# #",
  "# #",
  "#####" noscale;          // entspricht scalestyle_none
tile Ti_BG „bg.gif“ scale;  // entspricht scalestyle_any
tile Ti_Logo „logo.gif“;    // Default: scalestyle_auto
...
on dialog start {
  if setup.scale > 100 then
  Ti_Logo.scalestyle := scalestyle_prop;   // dynamisch
endif
}
```

Siehe auch Kapitel „tile (Muster)“ im „Ressourcenreferenz“-Handbuch.

6.1.3 Erweiterung der Font-Ressource

Die Font-Ressource hat eine neue Eigenschaft .propscale. Dies steuert, ob das horizontale und vertikale Raster proportional auf den maximalen Wert gesetzt werden soll, welcher durch die Werteberechnung von xraster und yraster des Font-Rasters ermittelt wird.

Die ***propscale***-Eigenschaft kann dabei entweder direkt bei der **Definition des Fonts** oder aber als **dynamisches Attribut** angegeben werden.

Der Divisor dient der Verfeinerung des Rasters.

Font-Definition:

```
fontspec ::=
...
...
{ x:= * <xscale> % + <xoffset> | / <xdivider>} { y:= * <yscale> %
+ <yoffset> | / <ydivider>}
{ propscale }
{ r:= "<RefString >" } ;
```

Dynamisc hes Attribut:

	Bezeichner		Datentyp
Regelsprache	**.propscale**		boolean
C	AT_propscale		DT_boolean
COBOL	AT-propscale		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	font		
Zugriff	get, set	Standardwert	false
changed-Ereignis	nein	Vererbung	–

Siehe auch Kapitel „font (Zeichensatz)“ im „Ressourcenreferenz“-Handbuch.

6.1.4 Erweiterung am Setup-Objekt

	Bezeichner		Datentyp
Regelsprache	**.tiledpi**		integer
C	AT_tiledpi		DT_integer
COBOL	AT-tiledpi		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	setup		
Zugriff	get, set	Standardwert	96
changed-Ereignis	nein	Vererbung	–

Dieses Attribut bestimmt für welche Auflösung die Bilder/Muster entworfen wurden. Die Größe eines Bildes/Musters wird ausgehend von diesem Wert auf den aktuell geltenden DPI-Wert umgerechnet.

Siehe auch Kapitel „setup“ im „Objektreferenz“-Handbuch.

6.1.5 Erweiterungen am Image-Objekt

Für eine bessere Ausrichtung und erweiterte Layoutmöglichkeiten können nun über die folgenden Attribute am Image-Objekt die Ränder bzw. Abstände gesteuert werden:

	Bezeichner		Datentyp
Regelsprache	**.xmargin**		integer
C	AT_xmargin		DT_integer
COBOL	AT-ymargin		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	image		
Zugriff	get, set	Standardwert	0
changed-Ereignis	nein	Vererbung	ja

Dieses Attribut bestimmt den horizontalen Abstand zwischen Rand und Darstellungsbereich.

	Bezeichner		Datentyp
Regelsprache	**.ymargin**		integer
C	AT_ymargin		DT_integer
COBOL	AT-ymargin		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	image		
Zugriff	get, set	Standardwert	0
changed-Ereignis	nein	Vererbung	ja

Dieses Attribut bestimmt den vertikalen Abstand zwischen Rand und Darstellungsbereich.

Siehe auch Kapitel „image (Bild)“ im „Objektreferenz“-Handbuch.

6.1.6 Unterstützung von HiDPI Bild-Varianten

Für eine optimale Gestaltung der Oberfläche bzgl. Bilder und Applikations-Icons durch Ausnutzung der höheren Auflösung sind für die Skalierungsstufe angepasste Bilder sinnvoll.
Bisher gab es hier nur für die Fenstersysteme WINDOWS und QT eingeschränkte Möglichkeiten. Windows bietet hier bislang zur Unterstützung den ICON/BITMAP-Mechanismus und Qt den Icon-Ressource-Mechanismus. Beide Mechanismen ermöglichen das Vorhalten mehrerer Auflösungsstufen,

sind jedoch begrenzt was Bildtyp oder Verwendung angeht. Einen solchen Mechanismus gibt es für MOTIF nicht.
Daher wurde mit dem IDM A.06.03.A eine Möglichkeit zum Laden mehrerer Auflösungsstufen geschaffen, die gleichermaßen für WINDOWS, QT und MOTIF angewandt werden kann und somit auch systemunabhängig ist.
Die Tile-Verwaltung vom IDM war bisher nur in der Lage eine Bilddatei zu laden (bspw.: tile Ti „smiley.gif“) und konnte nicht mehrere Auflösungsstufen der Datei vorhalten (bis auf o.g. Ausnahmen). Liegt nun eine Skalierung der Anwendung vor, versucht der neue Lademechanismus für Tile-Dateien nun zuerst die Bilddaten für die entsprechende Skalierung zu laden. Dazu wird in Anlehnung an den QT-Mechanismus die Endung ***@nx*** an den Dateinamen und vor dem Suffix angehängt. Dabei steht ***n*** für die Skalierungsstufe zwischen *2* und *9*. Die Zahl *2* entspricht dabei einer Skalierung von *150%-249%*, *3* von *250%-349%*, usw.
Sinnvollerweise sollten die Größenänderungen der Varianten Bilder der Skalierungsstufe entsprechen, wobei eine Überprüfung durch den IDM nicht erfolgt. Also z.B. Orginalbild „smiley.gif“ (32x32 Pixel), smiley@2x.gif (64x64 Pixel), smiley@3x.gif (96x96 Pixel), smiley@4x.gif (128x128 Pixel).

Beispiel:

```
tile Ti „smiley.gif“;  // bei einer Skalierung von 200% wird – sofern
                       // vorhanden – vom IDM automatisch das Bild
                       // smiley@2x.gif geladen (Windows, Motif, Qt)

                       // bei einer Skalierung von 300% wird – sofern
                       // vorhanden – vom IDM automatisch das Bild
                       // smiley@3x.gif geladen (Windows, Motif, Qt)

tile Ti2 „smiley.ico“  // nur Windows! ICON-Mechanismus von Window nimmt
                       // das für die Auflösungsstufe am besten Bild aus
                       // der .ico-Datei

tile Ti3 „:/smiley“    // nur Qt! Der Resource-Mechanismus von Qt nimmt das
                       // für die Auflösungsstufe am besten passende Bild
                       // mit dem hier angegebenen Alias aus der Ressource-
                       // Datei
```

6.1.7 Startoptionen

-IDMscale <integer>

Über die Startoption **-IDMscale** kann die Skalierung durch den IDM aktiviert oder deaktiviert werden. Unter QT und MOTIF gibt der Wert die Skalierung in % an.

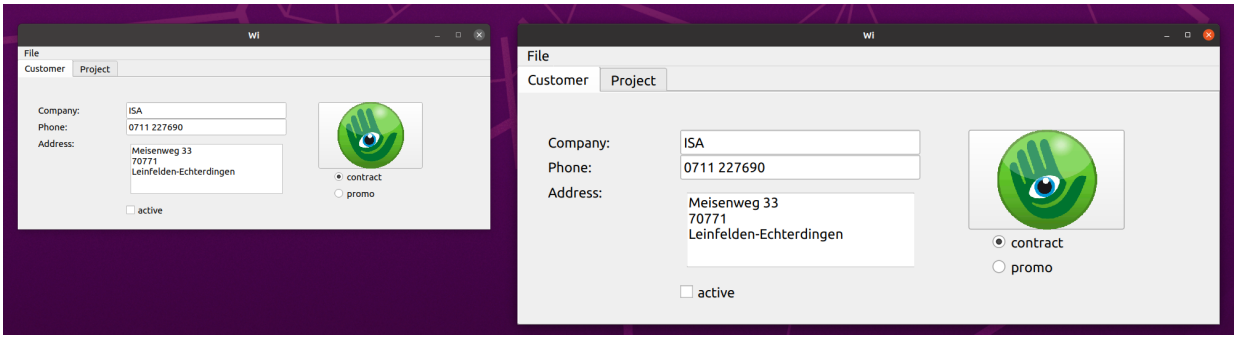


Abbildung 4: *links normal gestartet (Systemskalierung 100%), rechts mit 150% Skalierung gestartet*

Hinweis

Diese Option sollte unter MICROSOFT WINDOWS nicht verwendet werden. Die DPI-Awareness ist eine Eigenschaft der Anwendung und sollte in einer Manifest-Datei spezifiziert werden. Für Testzwecke kann die DPI-Awareness eingeschaltet (Wert: *1*) oder ausgeschaltet (Wert: *0*) werden.

Achtung:

Es wird empfohlen eine Skalierung > 0 und < 100% nicht zu verwenden, da es hier zu Einschränkung in der Darstellung und Bedienung von Objekten kommen kann.

-IDMtiledpi <integer>

Grafiken werden automatisch entsprechend des eingestellten Vergrößerungsfaktor skaliert. Es wird hierbei davon ausgegangen, dass die Bilder für einen DPI-Wert von *96* entworfen wurden. Sind die Bilder der Anwendung für eine höhere Auflösung entworfen worden, kann diese über die Startoption -IDMtiledpi eingestellt werden. Die Größe eines Bildes wird dann ausgehend von diesem Wert auf den aktuell geltenden DPI-Wert umgerechnet und dementsprechend skaliert.

Ebenso kann diese Einstellung direkt am setup-Objekt mit dem Attribut .tiledpi (siehe Erweiterung am Setup-Objekt) vorgenommen werden.

6.1.8 Installationshinweise

Um eine Anwendung für unterschiedliche Auflösungen vorzubereiten, kann es notwendig sein, die Grafiken für die verschiedenen Skalierungsstufen vorzuhalten. Wer das in Anspruch nehmen möchte und nicht den ICON-Mechanismus von WINDOWS oder den Resource-Mechanismus von QT nutzt, der sollte darauf achten die Bilderdateien entsprechend der @-Namensgebung (siehe Kapitel „Unterstützung von HiDPI Bild-VariantenSupport of HiDPI image variants“) mit seiner Anwendung mit auszuliefern bzw. diese ggf. in die Installationspakete mit aufzunehmen.

6.1.9 Geometrie und Koordinaten

Koordinaten und Größenangaben werden vom IDM passend zum Skalierungsfaktor des Systems transformiert um eine zu den Systemeinstellungen konsistente Darstellung zu gewährleisten.

Die Geometrieeinheiten werden dabei wie bisher in **IDM-Pixelwerten** oder *Raster* gesetzt.

Die **IDM-Pixelwerte** werden intern entsprechend dem Skalierungsfaktor in **reale Pixelwerte** hochgerechnet und umgesetzt. Dadurch wird die Anwendung als Ganzes skaliert und Verhältnisse und Proportionen der Objekte zu- und untereinander beibehalten.

Diese Konvertierung geschieht in beide Richtungen. Beim Abfragen von Größe und Position von Objekten wie auch bei der Event-Verarbeitung werden die **realen Pixelwerte** vom IDM ebenso wieder bereinigt in **IDM-Pixelwerten** angegeben. Der Vergrößerungsfaktor ist dann wieder herausgerechnet. Dies betrifft alle pixelorientierten Geometrieattribute.

Rasterwerte sind entweder an den verwendeten *(HighDPI-fähigen) Zeichensatz* geknüpft oder werden ebenfalls in **IDM-Pixelwerten** gesetzt und dann hoch skaliert. Die *Rastereinheiten* können basierend auf der zugrundeliegenden Schriftart ebenfalls zu **IDM-Pixelwerten** ermittelt werden.

6.1.10 Unterstützung hoher Auflösungen unter Motif

Das Motif-Toolkit selbst enthält zwar keine besondere Technologie zur Unterstützung hoher Auflösungen, wenn aber das verwendete X-Display die Xrender-Extension implementiert hat und eine Unterstützung von XFT-Fonts mit den entsprechenden Schriftarten vorliegt, können IDM-Anwendungen für Motif auf HiDPI-Bildschirmen, unter Berücksichtigung des Skalierungsfaktors des Desktops, betrieben werden.

Wie kann geprüft werden, ob auf dem verwendeten X-Display/Desktop eine Unterstützung vorliegt? Dazu kann die Serverinformation des X-Displays über die Option **-IDMserver:** erfragt werden.

```
$ idm -IDMserver
ServerVendor The X.Org Foundation
VendorRelease 12013000
Motif at compiletime @(#)Motif Version 2.3.8
Motif at runtime 2003
XRenderVersion 11 (active)
XFT Support yes (active)
Scaling 200% (active)
Screen dpi 96 (tile dpi: 96, scaled dpi: 192)
```

Die Version der Xrender-Extension und deren Nutzbarkeit (active) wird darin ebenso aufgelistet wie das Vorhandensein von XFT und seiner Nutzbarkeit. Die „Scaling“-Zeile zeigt den Skalierungsfaktor an, der in den Desktop-/Anzeigeeinstellungen vom Anwender gesetzt wurde.

Die DPI-Information welche X liefert, ist zumeist der Standardwert von 96 und spiegelt damit nicht unbedingt den real am Monitor vorhandene DPI-Wert wider. Die optional ausgegebenen „tile dpi“ und „scaled dpi“ dienen zur Kontrolle der für die Bilder und Bedienelemente zugrunde gelegten DPI-Werte.

Ein dynamisches Wechseln der Skalierung ist für Motif-Anwendungen nicht vorgesehen. Ebenso kann eine Umsetzung der Skalierung im Desktop nicht erkannt werden.

Unterstützung von XFT / Font-Handhabung / Motif-Font

Die Basis für eine GUI-Anwendung für unterschiedliche Desktop-Skalierungen stellen die verwendeten Schriften dar. Im Gegensatz zu den bisher unter Motif/X verwendbaren Schriftarten, die meist über einen XLFD (X Logical Font Descriptor) angeben wurden, werden XFT Fonts entsprechend der gesetzten Skalierung des Desktop mitskaliert.

Die bisher unter dem IDM FÜR MOTIF verwendbaren Schriftarten hatten sich nicht automatisch an unterschiedliche Desktop-Skalierungen angepasst.

Die XFT-Unterstützung ist im IDM nur auf Linux-Plattformen verfügbar. Da die meisten Desktops auf Linux-Plattformen den Standard-Font für Motif-Widgets nur unzureichend setzen, benutzt der IDM ***Liberation Sans*** *als Standard-Schrift und* ***Liberation Mono*** *als Festbreitenschriftart, wenn kein Font im Dialog gesetzt wurde. Dadurch skalieren sich Texte in IDM-Dialogen auch ohne gesetzte Schriftart.*
Durch explizites Ausschalten des Scaling (über die Startoption -IDMscale=0), kann man die alten Default-Schriftarten erhalten.

Zum setzen eines XFT Fonts muss wie bei bisherigen Fontdefinitionen im IDM nur der Fontname und ggf. weitere Modifier gesetzt werden. Ist der Font als XFT Font auf dem System vorhanden, dann wird dieser geladen. Die Liste der auf dem System verfügbaren XFT Fonts kann im IDM EDITOR eingesehen werden.

Unter CDE-konformen Plattformen wie AIX und HP-UX verwendet der IDM als Standard-Schrift „-dt-interface_system-medium-r-normal-*“ und „-dt-interface_user-medium-r-normal-*“ als Festbreitenschrift, sofern keine Schrift gesetzt wurde. Je nach Skalierungsfaktor werden dabei unterschiedliche Größenausprägungen des Fonts herangezogen. Zwischen *1%...75%* wird **xs** (extra small) verwendet, zwischen *75%-150%* wird **m** (medium) verwendet, zwischen *150%-250%* wird **l** (large) verwendet und ab *250%* wird **xxl** (extra extra large) verwendet. Dadurch kann für solche Anwendungen eine Skalierbarkeit erreicht werden auch wenn keine XFT-Fonts verfügbar sind.

6.1.11 Unterstützung hoher Auflösungen beim IDM für IDM for Windows 11

Es gibt jetzt einen ***IDM für Windows 11*** und einen für ***Windows 10***. Der IDM FÜR WINDOWS 11 unterstützt DPI-Awareness und High DPI.

Der IDM FÜR WINDOWS 11 unterstützt hohe Auflösungen des Modells „PerMonitor V2“. Da die Unterstützung hoher Auflösungen eine Eigenschaft der Anwendung ist, muss diese im Anwendungsmanifest entsprechend gekennzeichnet werden. Entsprechende Setzung können in den IDM-Beispielen entnommen werden. Der IDM erkennt dies dann selbständig und rechnet die Koordinaten entsprechend dem eingestellten Vergrößerungsfaktor um. Mit anderen Worten, die Pixelkoordinaten des IDM sind virtuelle Pixel, die entsprechend dem eingestellten Vergrößerungsfaktor umgerechnet werden.

Die Manifest-Dateien der IDM-Beispiele kennzeichnen die gebauten Anwendungen als „High DPI-Aware“. Um eine Anwendung zu bauen, die nicht „DPI-Aware“ ist, muss der gesamte Block `<asmv3:application>` aus der Manifest-Datei entfernt werden.

Hinweise IDM für Windows 11

Alle Funktionen, die Microsoft Windows Nachrichten verarbeiten: Wenn die Anwendung hohe Auflösungen unterstützt, dann sind alle Koordinaten, die über Microsoft Windows Nachrichten erhalten werden, die realen hochauflösenden Koordinaten. Der IDM benutzt dagegen um den Vergrößerungsfaktor bereinigte Koordinaten. Es muss also berücksichtigt werden, dass im Falle der Unterstützung hoher Auflösungen, die IDM-Koordinaten im Allgemeinen nicht mehr identisch zu den Microsoft Windows Koordinaten sind. Betroffen hiervon sind zum Beispiel Inputhandler-, Canvas-, Monitor- und Subclassing-Funktionen, sowie auch die USW-Implementierungen.

Hinweise IDM für Windows 10 und 11:

Die Unterstützung hoher Auflösungen erfordert eine andere Behandlung von Windowsnachrichten. Dadurch ist es leider nicht mehr möglich Fenster während des Verschiebens oder Vergrößerns an Rahmenbedingungen wie zum Beispiel Rasterkoordinaten anzupassen. Erst nach dem Loslassen der Maustaste wird das Fenster auf die nächste Rasterkoordinate angepasst.

Intern definierte Cursor werden jetzt auf die vom System geforderte Größe vergrößert bzw. verkleinert, um diese automatisch auf den eingestellten Vergrößerungsfaktor anzupassen. Hierdurch werden die Cursor richtig dargestellt, wenn die Anwendung hohe Auflösungen unterstützt. Bisher wurden die Cursor entweder mit durchsichtigen Bereichen aufgefüllt oder einfach abgeschnitten, wenn die Größe nicht passte.

Bilder (tile Ressource) werden automatisch entsprechend des eingestellten Vergrößerungsfaktor vergrößert. Es wird hierbei davon ausgegangen, dass die Bilder für einen DPI-Wert von *96* entworfen wurden. Sind die Bilder der Anwendung für eine höhere Auflösung gestaltet, dann kann dies am setup Objekt mit dem Attribute .tiledpi eingestellt werden. Eine automatische Anpassung gilt nur für die tile-Ressource. Bilder die direkt über den Dateinamen am .picture Attribute des image-Objektes angegeben werden, werden nicht automatisch angepasst.

Als Standardschriftart wird nun die in den Systemparameters definierte Dialog- bzw. Message-Schriftart verwendet, da diese sich an die Auflösung anpasst. Hierdurch kann es zu Änderungen der Darstellung von Objekten kommen, die keine Schriftart gesetzt haben. Die erwähnte Schriftart kann über Windows Systemeinstellungen konfiguriert werden.

Die alten Systemschriftarten sind nicht DPI-Aware. Nur die neuen ***UI*_FONT*** Schriftarten passen sich der eingestellten Auflösung an.
Die alten Windows-Schriftarten „ANSI_FIXED_FONT“, „ANSI_VAR_FONT“, „DEVICE_ DEFAULT_FONT“ und „SYSTEM_FONT“ lassen sich nicht an verschiedene Auflösungen anpassen, daher wird von der Verwendung abgeraten. Von der Verwendung von Schriftarten ohne Größenangabe wird abgeraten, da in einem solchen Fall vom Windows Font Mapper eine schriftartspezifische Standardgröße gewählt wird, die sich nicht an unterschiedliche Auflösungen anpasst.

6.1.12 Unterstützung hoher Auflösungen unter Qt

Für die Unterstützung hoher Auflösungen aktiviert der IDM das Qt-seitige HighDpi-Scaling sowie das mögliche Rendern von Bildern über ihre eigentlich angeforderte Größe hinaus. D.h. die Skalierung wird auf Basis der Pixeldichte des Monitors vorgenommen.

Unter Linux Desktop-Umgebungen ist die Unterstützung für HighDPI leider sehr unterschiedlich ausgeprägt und fortgeschritten. Da die Werte für logische und tatsächliche Pixeldichte sowie Skalierungsfaktoren vom Qt-Framework bezogen werden, kann es sein, dass weniger populäre Desktop-Umgebungen hier unzureichende Werte liefern.

Hinweise zu HighDPI Einstellungen

Qt bietet noch einige Umgebungsvariablen an, mit denen die Darstellung bei hohen Auflösungen gesteuert werden kann. Bei Einsatz solcher Umgebungsvariablen sollte jedoch bewusst sein, dass somit das Verhalten des IDM überschrieben wird und es dadurch zu ungewollten Darstellungseffekten kommen kann.

Anmerkungen zur Skalierung von Tiles

Objekte werden standardmäßig mit den vom Desktop-Style vorgegebenen Werten durch das WSI/Toolkit dargestellt. Es kann daher in bestimmten Situationen vorkommen, dass die gewünschten Einstellungen bezüglich Größe und Skalierung von Tiles nicht umgesetzt werden. Dies betrifft die Verwendung von Tiles bei Objekten mit Tabs wie dem Notebook bzw. den Notepages sowie dem Menu mit Menuitems. Der IDM setzt zwar die gewünschten Darstellungsoptionen, jedoch unterliegt die finale Anzeige der Darstellungsrichtlinie der jeweiligen DrawingEngine des gesetzten UI-Styles. Das bedeutet, dass die Anzeige von den gesetzten Optionen Style-abhängig abweichen kann oder Optionen vollständig ignoriert werden.

6.1.13 Erweiterungen/Änderungen im IDMED

Eine beliebige Auswahl der Schriftart für die Bedienoberflächen oder Regelcode ist nicht mehr möglich. Dafür kann die Schriftgröße zwischen Klein (Small) – Normal – Groß (Large) – Extra Groß (Extra Large) ausgewählt werden. Diese hat dann natürlich Einfluss auf die Fenstergröße.

Im Schriften- bzw. Font-Einstellungsbereich werden unter Motif auch XFT-Fonts aufgelistet. Außerdem kann die Liste der Auswählbaren Schriften zum Zwecke der Übersichtlichkeit auf UI-Fonts, X-Fonts und XFT-Fonts reduziert werden.

Der IDMED, TracefileAnalyzer sowie der Debugger setzen nun UI-Ressourcen ein.

6.1.14 Unterstützung hoher Auflösungen bei der USW-Programmierung

Die Funktion DM_GetToolkitDataEx wurde zur Unterstützung von HighDPI verbessert. Die Toolkit datena-S strukcture DM_ToolkitDataArgs wurde zu diesem Zweck erweitert und liefert nun detaillierte Informationen zu DPI, Skalierungsfaktor und Bild. Diese können über die Attribute *AT_DPI* und *AT_XWidget* erfragt werden.

Hinweis Motif

Der USW-Programmierer muss die Koordinaten zwischen dem IDM (dies betrifft Attribute sowie Events) und den Motif-Werten/Strukturen/Funktionen (*X…/Xt…/Xm…*) umrechnen. Die für eine Umrechnung benötigten Daten können über Funktion DM_GetToolkitDataEx mittels AT_DPI und die Toolkit datena-S strukcture DM_ToolkitDataArgs-Struktur (*tile.scale.factor*) erfragt werden. Je nach Bildart muss u.U. auch eine Skalierung vorgenommen werden.

Hinweis für Windows 11

Der IDM FÜR WINDOWS 11 unterstützt hohe Auflösungen. Wenn eine Anwendung für hohe Auflösungen freigegeben wird, dann müssen auch die von ihr verwendeten USW-Objekte hohe Auflösungen unterstützen. Das heißt dann, dass die USW-Objekte von Windows reale Koordinaten erhalten (um den Vergrößerungsfaktor vergrößert), während der IDM mit virtuellen Koordinaten (nicht vergrößert) arbeitet. Praktisch wird sich nicht viel ändern, da der IDM die Koordinatenberechnungen für das USW Objekt ausführt. Aber es ist zu berücksichtigen, dass der **control** Aufruf für „UC_C_PrefSize“ reale Koordinaten erwartet, da diese im Normalfall aus Toolkitwerten berechnet werden. Sind hier fixe Zahlen wie im *ucarrow* Beispiel eingesetzt, dann müssen diese um den aktuellen DPI-Wert korrigiert werden (siehe ucarrow Beispiel).

Wenn die Anwendung als DPI-Aware gestartet wurde, werden nun von der Task „UC_I_clientarea“ der Funktion **UC_Inquire** auch die hochauflösenden Werte erwartet.

Ergänzungen der C-Schnittstelle für DM_GetToolkitData() und DM_GetToolkitDataEx()

Bei den C-Schnittstellen Funktionen "DM_GetToolkitData()", "DM_GetToolkitDataEx()" und Toolkitdaten-Struktur "DM_ToolkitDataArgs" gibt es zur Unterstützung von hohen Auflösungen die folgenden Erweiterungen:

Attribute AT_DPI

Gibt den DPI-Wert des Systems oder des angegebenen Objektes zurück. Die Funktion "DM_GetToolkitDataEx()" muss mit einem Zeiger auf eine Toolkitdaten-Struktur "DM_ToolkitDataArgs" aufgerufen werden. In dieser Struktur werden weitere DPI-Informationen zurückgeliefert. Sie ist hierzu um das Datenfeld "dpi" und die Unterdatenstruktur "scale" erweitert.

Attribute "AT_Tile" bzw. "AT_XTile"

Dieses Attribut gab es schon für die "tile" Ressource. Jetzt kann die Funktion "DM_GetToolkitDataEx()" mit einem Zeiger auf eine Toolkitdaten-Struktur "DM_ToolkitDataArgs" aufgerufen werden, die um die Unterdatenstruktur "tile" erweitert wurde. Diese enthält unter anderem den DPI-Wert, für den die "tile" Ressource entworfen wurde.

Attribute "AT_IsNull"

Liefert einen Wert ungleich "0" zurück, wenn die angegebene "color" oder "font" Ressource auf "UI_NULL_FONT" bzw. "UI_NULL_COLOR" definiert war.

Siehe auch Kapitel „Toolkit datena-S strukcture DM_ToolkitDataArgs“ im „C-Schnittstelle - Grundlagen“-Handbuch.

Siehe auch Kapitel „DM_GetToolkitDataEx“ im „C-Schnittstelle - Funktionen“-Handbuch.

Siehe auch Kapitel „DM_GetToolkitData“ im „C-Schnittstelle - Funktionen“-Handbuch.

6.2 Multi-Monitor Support unter Windows

Unter Windows ist es jetzt möglich, die Anzahl der Monitore (.screencount) und die Koordinaten der Monitore (.screen_x[integer], .screen_y[integer], .screen_width[integer] und .screen_height[integer]) über das setup-Objekt abzufragen. Die erfragten Werte sind dabei dynamisch, da man jederzeit die Skalierung eines Monitors ändern oder auch Monitore hinzufügen oder wegnehmen kann.

Werden mehrere Monitore verwendet, werden diese in einem virtuellen Desktop positioniert. Die Koordinaten dieses virtuellen Desktop sind mit .vscreen_x, .vscreen_y, .vscreen_width, .vscreen_height abfragbar. Es werden immer die Koordinaten des Arbeitsbereichs, also ohne die Taskleiste, geliefert. Zur Berechnung des virtuellen Desktops werden auch die Arbeitsbereiche herangezogen.

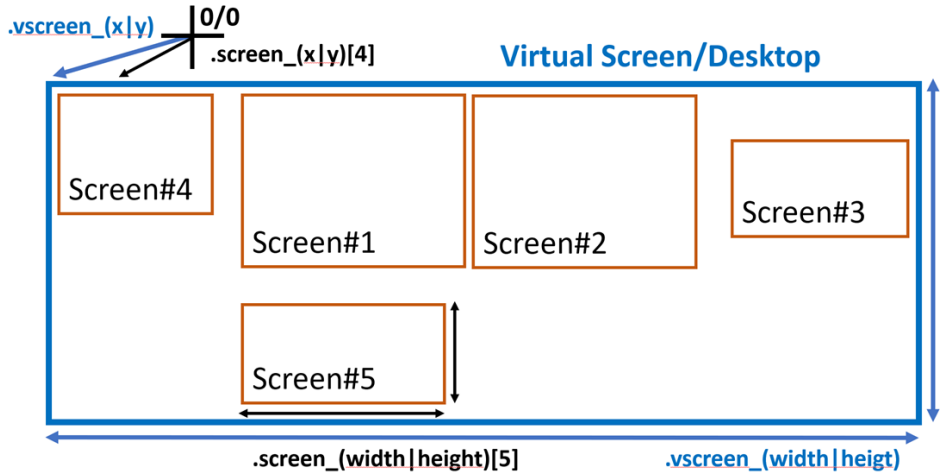


Abbildung 5: *Relation der .screen_*[] und .vscreen_* Attribute*

Achtung:

Bei unterschiedlichen Vergrößerungseinstellungen der Monitore ist zu beachten:

IDM für Windows 11

Die Werte erscheinen nicht konsistent. Ausgehend von der für Fenster empfohlenen Verwendung von *.xauto = 1* und *.yauto = 1* wird die Position mit dem Systemvergrößerungsfaktor umgerechnet, während die Größe mit dem Monitorvergrößerungsfaktor umgerechnet wird. Hierdurch lässt sich ein Fenster auf eine Position und Größe setzen, die den gesamten Monitor ausfüllt.
Die Position und die Größe des virtuellen Desktops werden mit dem Systemvergrößerungsfaktor umgerechnet. Zu welchem Monitor ein Fenster, das sich über mehrere Monitore erstreckt, zugeordnet wird, kann von dem Monitor abhängen, auf dem sich das Fenster gerade befindet. Für ein definiertes Verhalten sollten deshalb Koordinaten nur im unsichtbaren Zustand geändert werden.

IDM für Windows 10

Microsoft Windows skaliert Anwendungen, die DPI-Unaware sind, intern. Hierbei führen die unterschiedlichen Vergrößerungsfaktoren zu Falschdarstellungen. Um dies zu vermeiden sollten entweder alle Fenster nur auf dem primären Monitor geöffnet werden, oder alle Monitore denselben Vergrößerungsfaktor besitzen.

Anmerkung zu Windows 10 und 11 mit mehreren Monitoren:

Unter WINDOWS kann es nach dem Hinzufügen/Entfernen eines Monitors oder dem Ändern der Anzeigeeinstellungen zu folgenden Problemen kommen:

- Es erscheint kein Verschieberahmen beim Verschieben eines **toolbar** Objektes. Es kann auch vorkommen, dass ein Rechteck in der Größe des **toolbar** Verschieberahmens mit einem falschen Hintergrund auf einem anderen Monitor dargestellt wird.
- Ausgedockte **toolbar** Objekte passen sich nicht an Änderungen der DPI Wertes an. Sowohl beim Verschieben auf einen anderen Monitor oder auch beim Ändern der Vergrößerung.

Das Problem liegt an WINDOWS, das zum einen die Koordinaten beim Zeichnen auf den Desktop falsch umsetzt und zum anderen wichtige Nachrichten (*WM_DPICHANGED*) nicht an sogenannte Tool-Fenster versendet und den Window DPI Wert nicht ändert.

Nachdem der Bildschirmschoner aktiv war oder man sich neu angemeldet hat (und nach einem Neustart), tritt das Problem nicht mehr auf.

6.3 Kompatibilität

6.3.1 Motif

Der IDM FÜR MOTIF stellt eine Anwendung vergrößert dar, wenn eine Systemskalierung aktiv ist. Mittels der Startoption `-IDMscale=0` kann dieses Verhalten ausgeschaltet werden kann.

6.3.2 Windows

Wenn eine Bildschirmvergrößerung aktiv ist, kann das Raster des IDM FÜR WINDOWS 11 nicht proportional zur Vergrößerung sein, da die Schriftgröße in logischen Points angegeben wird. Je mehr Pixel zur Verfügung stehen umso besser entspricht der logische Wert dem realen.

6.3.3 Qt

Beim IDM FÜR QT ist nun HighDPI-Skalierung eingeschaltet. D.h. bei aktiver Systemskalierung wird die Anwendung nun unweigerlich und automatisch angepasst, was nun nicht mehr nur Fonts sondern auch Geometrie betrifft.

6.4 Kern

IDM-13192: Ein *finish*-Ereignis von Applikations-Objekten hatte Einfluss auf die Behandlung von finish-Ereignissen von Dialog/Module-Objekten. Dies führte zur doppelten Ausführung von `finish`-Regeln am Dialog/Modul und konnte das Verlassen, bzw. nicht-Verlassen der Ereignisschleife beeinflussen. Dieses Fehlverhalten ist nun behoben.

IDM-13191: Es kommt nun nicht mehr zu ungültigen Speicherzugriffen im Regel-Resolver bei der Auflösung von Pfaden in Werte-Ausdrücken. Auch ein u.U. weggelassener Attributszugriff sollte nicht mehr passieren.

IDM-13072: Das „Sharing“ von importieren Modulen, welches mittels Umgebungsvariable **DM_SHARED_MODULES** aktiviert werden kann, um Imports mit gleichem Bezeichner vom Vorhandensein des gleichen Imports im Supermodule zu befreien, ist eingeschränkt um die Konsistenz bzgl. `Start/Stop/:init/:finish`, der Dialogzugehörigkeit und dem Sichtbarkeitszustand von Objekten zu gewährleisten. Es können nur Module im gleichen Dialog geshared werden!

Achtung:

Das Sharing von Modulen über **DM_SHARED_MODULES** ist kein Standard-Feature des IDM. Sinnvoller ist die Verwendung von „use“ um Module zu importieren und so das mehrfache Laden von Modulen in komplexen Dialogen sicher zu unterbinden.

Neu: Attribut .hinttext an den Objekten ***edittext*** und ***poptext*** eingefügt, mit dessen Hilfe es möglich ist einen Platzhalter- oder Hinweistext anzeigen zu lassen, wenn das Objekt noch keinen Inhalt gesetzt hat. Am ***poptext***-Objekt muss muss dabei der Style *edittext* gesetzt sein..

	Bezeichner		Datentyp
Regelsprache	**.hinttext**		string
C	AT_hinttext		DT_string
COBOL	AT-hinttext		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	edittext, poptext		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Achtung:

Bei einem Leerzeichen als Inhalt wird der Platzhalter - oder Hinweis-Text nicht angezeigt, dies betrifft ebenso einige Formate.

Besonderheiten:

Dieses Attribut wird nur von WINDOWS und QT unterstützt.

Anmerkung für Windows:

Unter MICROSOFT WINDOWS wird das Attribut im Moment an folgenden Objekten unterstützt:

» ***edittext***-Objekt, wenn die Attribute .multiline und .options[opt_rtf] den Wert *false* besitzen

» ***poptext***-Objekt, wenn das Attribut .style den Wert *edittext* oder *listbox* besitzt

Der Hinttext wird angezeigt, wenn das Objekt den Fokus nicht besitzt und der Inhalt leer ist.

Anmerkung für Qt:

Wenn beim ***edittext***-Objekt das Attribut .multiline den Wert *true* besitzt und Layoutattribute (z.B. .alignment, .xmargin, .ymargin etc.) gesetzt sind, so werden diese bei der Anzeige des Hinweis- oder Platzhaltertexts nicht beachtet. Der Hinweis- oder Platzhaltertext ist immer links oben angebunden.

6.5 Windows

Neu: ***Achtung:*** Es gibt jetzt einen IDM FÜR WINDOWS 11 und einen IDM FÜR WINDOWS 10. Der IDM für Windows 11 unterstützt ***DPI Awareness*** und ***High DPI***.

Neu: Mit dem IDM FÜR WINDOWS 11 wird nun auch MICROSOFT VISUAL C++ 2022 unterstützt.

IDM-13194: Nach einer Drag und Drop Aktion erschien die Fehlermeldung "WSI error processing clipboard.value". Voraussetzung war, dass eine :setclip Methode existierte, die das Clipboard änderte.

IDM-13215: Unter Windows wurden die Geometrieattribute nicht vererbt, wenn diese interaktiv am Modell geändert wurden.

IDM-12999: Es trat eine Exception in einer **UIAutomation** Testanwendung auf, wenn an einem poptext Objekt die Option **opt_hscroll** gesetzt war. Dies wird jetzt vermieden, indem das **UIAutomation** Objekt für die Poptextliste erst wieder freigegeben wird, wenn das Poptext **UIAutomation** Objekt nicht mehr gebraucht wird. Seither wurde das **UIAutomation** Objekt für die Poptextliste freigegeben, wenn diese nicht mehr gebraucht wurde.

IDM-12967: Es konnte vorkommen, dass die erste Tracemeldung der **UIAutomation** leer war. Das Problem trat dann auf, wenn das erste Objekt erst nach dem Einschalten des **UIAutomation** Tracings sichtbar geschalten wurde.

IDM-13188: Wenn ein Fenster aus dem minimierten Zustand heraus maximiert wurde, dann wurden unten angebundene Objekte falsch positioniert, sie erschienen zu weit unten.

IDM-11065: Wenn die Anwendung unter Terminal Services lief, ertönte kein Warnton. Jetzt wird unter Terminal Services die WINDOWS-Funktion **Beep** verwendet. Achtung, die Änderung betrifft nur die Töne, die der IDM erzeugt.

IDM-10294: **Mnemonics** werden nun nicht mehr auch an Objekten auf inaktiven notepage Seiten ausgelöst.

Neu: Die Attributsnamen *AT_Font* und *AT_XFONT* wurden als Alternative zu *AT_wsidata* für die **ToolkitData**-Funktionen an der font Ressouce eingeführt. Die Verwendung ist identisch zu *AT_wsidata*.

Neu: Die Attributsnamen *AT_Color*, *AT_Tile* und *AT_Widget* als Alternative zu *AT_XColor*, *AT_XTile* und *AT_Widget* für die **ToolkitData**-Funktionen eingeführt. Die Verwendung ist identisch zu dem entsprechenden Attribut mit dem „X“.

IDM-12697: Bei einer Installation auf ein Netzwerklaufwerk kam es zu Problemen. Achtung, die Installaltion auf ein Netzwerklaufwerk ist nicht vorgesehen. In der Regel wird der Versuch zu einem Abbruch führen, nachdem bereits alle Dateien entpackt sind. Falls kein Abbruch geschieht, findet das Entsperren der Dateien statt. Dies kann nur in der letzten Phase der Windows Installation geschehen. Diese Phase wird vom Windows Installer unter dem "SYSTEM" Konto ausgeführt und hat in der Regel keinen Zugriff auf Netzwerk-Resourcen.

Erweitert: Für den IDM EDITOR werden jetzt mehr Informationen zu den vorhandenen Schriftarten angezeigt, mögliche Optionen werden verdeutlicht. Außerdem wird bei der Selektion jetzt der *type* Parameter berücksichtigt.

Neu: Die Option **opt_show_ticks** wird nun vom Scrollbar-Objekt unterstützt.

Behoben: Das setup Objekt wurde im WSI nicht angelegt/initialisiert. Hierdurch wurde ein initial gesetzter .overridecursor nicht übernommen.

Neu: Die font Ressource um die neuen Optionen für *black*, *demibold* und *light* erweitert. Außerdem wird jetzt auch die Fontfamilie *roman* unterstützt.

IDM-13057: Der Verschieberahmen eines toolbar Objektes erschien nicht über dem toolbar Objekt, wenn ein Vergrößerungsfaktor eingestellt war. Dieses Problem ist jetzt behoben.

Anmerkung Windows 10 und 11 mit mehreren Monitoren:

Unter WINDOWS kann es nach dem Hinzufügen/Entfernen eines Monitors oder dem Ändern der Anzeigeeinstellungen zu folgenden Problemen kommen:

- Es erscheint kein Verschieberahmen beim Verschieben eines **toolbar** Objektes. Es kann auch vorkommen, dass ein Rechteck in der Größe des **toolbar** Verschieberahmens mit einem falschen Hintergrund auf einem anderen Monitor dargestellt wird.
- Ausgedockte **toolbar** Objekte passen sich nicht an Änderungen der DPI Wertes an. Sowohl beim Verschieben auf einen anderen Monitor oder auch beim Ändern der Vergrößerung.

Das Problem liegt an WINDOWS, das zum einen die Koordinaten beim Zeichnen auf den Desktop falsch umsetzt und zum anderen wichtige Nachrichten (*WM_DPICHANGED*) nicht an sogenannte Tool-Fenster versendet und den Window DPI Wert nicht ändert.

Nachdem der Bildschirmschoner aktiv war oder man sich neu angemeldet hat (und nach einem Neustart), tritt das Problem nicht mehr auf.

Neu, IDM-11904, IDM-11753: Der IDM FÜR WINDOWS bietet nun Multi-Monitor Support. Siehe Kapitel 6.2 „Multi-Monitor Support unter Windows“

Damit ein Fenster in einer Multi Monitor Konfiguration richtig maximiert wird, muss die Berechnung der maximalen Größe Microsoft Windows überlassen werden. Hierdurch können Fenster von der Taskbar überdeckt werden.

Behoben: **Achtung:** Änderung der Berechnungsmethode für die Rasterbreite.

Mit der IDM-Version A.06.03.a wurde die Berechnung der Rasterbreite grundlegend umgestellt. Wenn kein Referenzstring angegeben ist, wird die Rasterbreite jetzt aus einem internen Referenzstring („M“) berechnet, um ein übermäßiges Breitenwachstum durch überbreite Buchstaben innerhalb einer Schriftart zu vermeiden.

Aus Kompatibilitätsgründen kann jedoch durch die Option *opt_fontraster_compat*, die Startoption **-IDMfontraster_compat** oder die Umgebungsvariable `IDM_FONTRASTER_COMPAT` vorübergehend die veraltete Berechnung der Rasterbreite wieder reaktivieren werden (verbunden mit dem Nachteil, dass es wieder zu übermäßiges Breitenwachstum kommen kann).
Bei Verwendung der Option *opt_fontraster_compat* basiert die Größenberechnung zum Teil auf dem Systemfont, der nicht High DPI fähig ist, daher sollten High DPI fähige Anwendungen, die mit IDM FÜR WINDOWS 11 erstellt wurden, dies Option nicht verwenden.

Hinweis: Die DM_GetToolkitData/DM_GetToolkitDataEx Attribute *AT_Tile*, *AT_XTile* und *AT_wsidata* wurden um Toolkit datena-S strukcture DM_ToolkitDataArgs erweitert und sind wie folgt belegt:

- `tile.gfxtype`
  Wird auf den *GFX Type* gesetzt, der am besten passt.
- `tile.datatype`
  Wird auf den Wert gesetzt, der bisher über *AT_DataType* abgefragt werden musste.
- `tile.data`
  Wird auf die Daten gesetzt, der genaue Typ ist abhängig von `tile.gfxtype` bzw. `tile.datatype`:
  HICON: wenn *DM_GFX_ICO* bzw. *DMF_TlkDataIsIcon*
  HBITMAP: wenn *DM_GFX_BMP* bzw. *0*
  HMETAFILE: wenn *DM_GFX_WMF* bzw. *DMF_TlkDataIsWMF*
  HENHMETAFILE: wenn *DM_GFX_EMF* bzw. *DMF_TlkDataIsEMF*
- `tile.mask`
  Wird auf die monochrome Maske gesetzt, falls eine vorhanden ist. Dies kann nur bei *DM_GFX_BMP* vorkommen.
- `tile.palette`
  Wird auf die Farbpalette gesetzt, falls eine vorhanden ist.
- `tile.dpi`
  Der DPI Wert für den die vom IDM geladenen Bilder entworfen wurden.
- `rectangle`
  Wird auf die Größe gesetzt, die zum angeforderten DPI Wert passt. Siehe Bemerkung.
- `dpi`, `scale.dpi` und `scale.factor`
  Werden passend zum angeforderten DPI Wert gesetzt. Siehe Bemerkung und Hinweise zu *AT_DPI*.

Bemerkung zum angeforderten DPI Wert:

Der angeforderte DPI Wert wird in der angegebenen Reihenfolge aus folgenden Angaben ermittelt:

- Ist in der `argmask` das *DM_TKAM_handle* Bit gesetzt, dann wird der DPI Wert des Microsoft Windows Control ermittelt dessen Windows Handle (HWND) im `handle` Element steht.
- Ist in der `argmask` das *DM_TKAM_scaledpi* Bit gesetzt, dann wird der System DPI Wert ermittelt (entspricht dem Vergrößerungsfaktor des primären Monitors).
- Sonst wird die originale Bildgröße, also der DPI Wert für den die Bilder entworfen wurden, ermittelt.

Achtung: Dies bedeutet nicht, dass die Bilddaten die entsprechende Größe haben, diese müssen eventuell skaliert werden.

Der Rückgabewert bleibt wie gehabt, also entspricht `tile.data`.

Hinweis: Die Elemente der Toolkit datena-S strukcture DM_ToolkitDataArgs werden beim Abfragen des Attributs *AT_DPI* in DM_GetToolkitData/DM_GetToolkitDataEx folgendermaßen belegt:

- `dpi`
  Wird auf den Standard DPI Wert gesetzt, den Anwendungen die nicht DPI Aware sind verwenden. Dieser Wert wird von MICROSOFT WINDOWS als Konstante *USER_DEFAULT_SCREEN_DPI* (Wert: *96*) definiert. Diesen Wert verwendet der IDM für seine Pixelkoordinaten.
- `scale.dpi`
  Wird auf den aktuellen DPI Wert des Objektes gesetzt, er entspricht dem Rückgabewert von *AT_GetDPI*. Dieser Wert ist beim IDM FÜR WINDOWS 11 dynamisch, wenn die Anwendung DPI Aware ist. Sonst, DPI Unaware oder IDM FÜR WINDOWS 10, ist der Wert fest auf *96* definiert.
- `scale.factor`
  Ganzzahliger Prozentwert, der aus `scale.dpi` und `dpi` berechnet wird. Der Rückgabewert (auf int gewandelt) entspricht `scale.dpi`.
  Um den DPI Wert für ein Microsoft Windows Control zu erhalten, kann in der Toolkit datena-S strukcture DM_ToolkitDataArgs Struktur das `handle` Element auf den Windows Handle (HWND) des Controls gesetzt werden. In der `argmask` muss dann das Bit *DM_TKAM_handle* gesetzt sein und als Objekt muss *0* oder das setup Objekt im DM_GetToolkitDataEx Aufruf angegeben werden.

Neu: Mit DM_GetToolkitData bzw. DM_GetToolkitDataEx kann unter MICROSOFT WINDOWS der aktuelle DPI Wert mit *AT_GetDPI* abgefragt werden. Abhängig vom *objectID* Parameter werden folgende Werte geliefert:

- ID eines sichtbaren Objektes:
  Der DPI Wert, der für dieses Objekt gilt. Wert ist abhängig vom Monitor auf dem das Objekt erscheint.

» NULL-ID oder ID des setup Objektes:
Der System DPI Wert oder, wenn im *data* Parameter der Microsoft Windows Window Handle (HWND) übergeben wird, der DPI Wert dieses Objektes.

» sonst:
Entweder ein zufälliger alter Wert (wenn Objekt nicht sichtbar) oder 0 und eine Fehlermeldung.

Der Rückgabewert muss auf "int" gewandelt werden. Achtung, alle Werte sind dynamisch und können durch den Anwender über die Systemsteuerung geändert werden. Ist die Anwendung nicht DPI Aware (zum Beispiel IDM FÜR WINDOWS 10) dann wird immer der Standard DPI Wert von *96* geliefert.

Änderung: Linkeroption zu Subsystemversion entfernt, so dass die Anwendung nicht mehr auf XP-Komatibilität gesetzt wird.

Behoben: Das scrollbar Objekt hatte die .real_width und .real_height Attribute nicht aktualisiert, wenn keine Größe gesetzt war und das Attribut .arrows den Wert false besaß.

Änderung: Ergänzungen zur Einstellung der fontraster-Kompatibität für das Windows-WSI. Diese ist normalerweise auf false/inactive. Diese Kompatibitätseinstellung kann über folgende Wege gesetzt/aktiviert werden:

» Startoption **-IDMfontraster_compat**

» Umgebungsvariable `IDM_FONTRASTER_COMPAT` ist gesetzt

» Option *setup.options[opt_fontraster_compat]*

Für Hinweise bzgl. Rasterberechnung siehe entsprechende Kapitel im Windows WSI.

Neu: DM_GetToolkitData bzw. DM_GetToolkitDataEx unterstützt unter MICROSOFT WINDOWS am setup Objekt die Attribute *AT_Font* und *AT_XFont*. Es wird der Font-Handle der verwendeten Standardschriftart zurückgeliefert (der Rückgabewert muss auf *HFONT* gewandlet werden). Die Schriftart ist für den System-DPI-Wert skaliert. Im *data* Parameter kann ein anderer DPI-Wert als Integer angegeben werden (Wandlung: (DM_Pointer) (size_t) dpi).

Neu: DM_GetToolkitData bzw. DM_GetToolkitDataEx unterstützt unter MICROSOFT WINDOWS an den IDM Objekten die Attribute *AT_Font* und *AT_XFont*. Es wird der Font-Handle der verwendeten Schriftart zurückgeliefert (Rückgabewert muss auf *HFONT* gewandelt werden). Der Font ist für den aktuellen DPI-Wert des Objektes skaliert.

Neu: DM_GetToolkitData bzw. DM_GetToolkitDataEx unterstützt unter MICROSOFT WINDOWS an der font und color Resource das Attribute *AT_IsNull*. Hiermit kann abgefragt werden, ob die Resource auf *NULL* definiert wurde (*UI_NULL_FONT* bzw. *UI_NULL_COLOR*). Die Resource wurde auf NULL definiert, wenn der Rückgabewert nicht *0* ist.

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

» *AT_wsidata*, *AT_Font*, *AT_XFont*:
Der Rückgabewert ist bei *UI_NULL_FONT* dann *(HFONT) 0*.

» *AT_Color*, *AT_XColor*:
Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(COLORREF) -1L*.

» *AT_Tile*, *AT_XTile*:
Der Rückgabewert ist bei *UI_NULL_COLOR* dann *(HBRUSH) 0*.

Neu: Das .scalestyle Attribut an der tile Ressource wird jetzt auch von den Objekten treeview, notebook/notepage, menuitem und window beachtet.

Neu: Die beiden Attribute .real_x und .real_y entsprechend .real_width und .real_height eingeführt.

Änderung: An der tile Ressource wurden die vordefinierten Namen *OLD_UPARROW*, *OLD_DNARROW*, *OLD_RGARROW*, *OLD_LFARROW*, *OLD_REDUCE, OLD_ZOOM* und *OLD_RESTORE* entfernt, es müssen die OEM Ressource-Namen ohne *OLD_* verwendet werden.

Behoben: Wurde am toolbar Objekt das .reffont, bzw. .xraster / .yraster Attribut geändert, dann wurde die Größe des toolbar Objektes nicht angepasst, wenn *.sizeraster = true* war.

Behoben: Wenn am poptext Objekt das .format Attribute dynamisch geändert wurde, dann wurde der Listeninhalt nicht angepasst.

Behoben: Wenn am image Objekt das Attribut .imagefgc gesetzt war, dann wurde diese Farbe fälschlicherweise auch als Textfarbe verwendet.

Änderung: Es gab bei manchen Objekten einen visuellen Unterschied zwischen einer gesetzten Schriftart, die es nicht gibt, und einer auf *null* gesetzten Schriftart. Dieser Unterschied tritt jetzt nicht mehr auf.

Neu: DPI-Awareness Informationen zum Tracefile-Header hinzugefügt. Sowie weitere Warnungen und Fehlermeldungen bei der DPI Initialisierung eingefügt, die auf nicht unterstützte Einstellungen hinweisen.

6.6 Motif

Behoben: Die Layoutbox hat beim Verkleinern die Kinder nicht sofort angepasst.

Behoben: Die Umrechnung der Positionen von Fenster und Toolbars (*dock_window*) erfolgt möglichst unter Berücksichtigung von realen Fensterdekorationen um beim zweifachen .visible-Tooglen das Fenster/Toolbar wieder an die gleichen Stelle zu positionieren.

Behoben: Insensitiver Text beim image Objekt wird ab einschließlich MOTIF-2.3 ausgegraut und mit Schattenwurf analog zu anderen MOTIF-Widgets gezeichnet.

Neu: Durch neuere Desktop- und Window-Manager vorgegebene WorkArea-Bereiche (z.B. durch Menüs oder Dock-Bereiche) werden vom IDM berücksichtigt und verkleinern somit die nutzbare Screen-Größe.

6.7 Qt

IDM-13186: Beim Umschalten von Hintergrund-Tiles in der groupbox wurde nicht beachtet, ob die groupbox eine virtuelle Größe hat.

Neu: Die Option **opt_show_ticks** wird nun vom Scrollbar-Objekt unterstützt.

Neu: Weitere bisher nicht beachtete System-Farben zur color-Ressource hinzugefügt.

Neu: DM_GetToolkitData unterstützt nun *AT_XTile* und *AT_XWidget*. *AT_XTile* liefert dabei immer den Typ *DM_GFX_QPIXMAP* mit QPixmap zurück.

Behoben: Wenn ein Tile nachträglich dynamisch an der splitbox gesetzt wurde, konnte es vorkommen, dass dieses nicht gezeichnet wurde.

Behoben: Wenn ein Tile nachträglich dynamisch an der layoutbox gesetzt wurde, konnte es vorkommen, dass dieses nicht gezeichnet wurde.

Behoben: Das window Objekt beachtete eine dynamische Änderung von .reffont nicht und hat in Folge dessen seine Größe nicht angepasst.

Behoben: Die Methode :openpopup() hatte das Kontextmenü unter Qt nur am Mauszeiger und nicht direkt am Objekt angezeigt.

Behoben: Beim Ändern der Richtung der scrollbar im Sichtbaren wurde die Größe der Scrollbar nicht angepasst.

6.8 DDM (Netz)

Neu: Das Schema ***ssh://*** unterstützt nun auch das OPENSSH Kommando. Dieses Schema findet Anwendung beim .exec Attribut um zu bestimmen wie die Serverseite gestartet wird. Es wird zuerst gesucht, ob die **libssh** DLL verfügbar ist. Ist diese nicht verfügbar wird überprüft, ob ein `ssh` Kommando aufrufbar ist. Zusätzlich gibt es das Schema ***sshlib://*** um nur **libssh** zu verwenden und ***sshcmd://*** um nur OPENSSH zu verwenden. Der Nachteil des Kommandos ist, dass kein Passwort verwendet werden kann. OPENSSH muss so konfiguriert sein, dass eine Verbindung ohne Passwort möglich ist (siehe Manpages von `ssh`).

Neu: Es wird jetzt auch OPENSSL Version 3 unterstützt.

Behoben: Alte OPENSSL DLLs konnten unter Umständen nicht geladen werden, da die Funktion **SSL_CTX_set_options** nicht verfügbar war. Das Problem konnte auftreten, wenn Bau- und Einsatzsystem unterschiedliche OPENSSL Versionen besaßen. Jetzt wird ggf. optional die alte Funktion **SSL_CTX_ctrl** verwendet.

Neu: Attribute .publickeyfile und .privatekeyfile hinzugefügt. Für .options[enum] die Optionen **opt_verify_peer**, **opt_cert_required** und **opt_no_ssl_v2**, sowie die entsprechenden Startoptionen hinzugefügt.

Behoben: Die Netzwerk-Seite verhielt sich zwischen UNIX und WINDOWS unterschiedlich wenn ein geöffneter Port von einem zweiten Prozess verwendet werden sollte. Jetzt wird unter Windows die Socketoption **SO_REUSEADDR** nicht mehr gesetzt, so dass es zu einem Fehler kommt wenn der Port bereits verwendet wird.

Behoben: Eine Verifizierung der SSL Zertifikate konnte scheitern, wenn Standardsuchpfade nicht geladen wurden.

Behoben: Die Anwendungsseite reagierte nicht mehr, wenn es zu SSL Fehlern kam. Jetzt wird bei Fehlern, die während dem Anlegen des SSL Kontexts geschehen, der Prozess nicht mehr beendet, sondern die Verbindung wird weiter aufgebaut und somit die entstehenden Fehler auf beiden Seiten mitprotokolliert.

6.9 Debugger

IDM-13002: Wenn eine zu große Tracedatei im IDM TRACEFILE ANALYZER abschnittweise geladen wurde, konnte es dazu kommen, dass die Anwendung nicht mehr reagiert hat. Dies wird nun verhindert.

Neu: Das abschnittweise Laden von zu großen Tracedateien im IDM TRACEFILE ANALYZER ist nun möglich.

Behoben: Der IDM TRACEFILE ANALYZER unter MICROSOFT WINDOWS hatte zum Teil die Darstellungsfläche nicht voll ausgenutzt.

6.10 Editor

Änderung: Die Preview eines Tiles ermittelt zuerst den realen Pfad vom editierten Modul, um so auch die Anzeige von Bildnamen mit „~“ besser zu unterstützen.

Änderung: Editiermöglichkeiten des .scale-Attributes bei tile-Ressourcen umgestellt auf Enum/Auswahl für .scalestyle.

Neu: Propscale-Eigenschaft bei font-Ressourcen nun editierbar.

Neu: Der IDM TRACEVIEW ANALYZER unter MICROSOFT WINDOWS hatte zum Teil die Darstellungsfläche nicht voll ausgenutzt.

6.11 USW

IDM-13057 (Keine Änderung): Hinweis zur High DPI Unterstützung des IDM für WINDOWS 11: Wenn die Anwendung als DPI Aware gestartet wurde, erwartet der IDM für WINDOWS 11 von der Task **UC_I_clientarea** der Funktion **UC_Inquire** die hochauflösenden, realen Werte.

Siehe auchKapitel „HighDPI Unterstützung“ im Handbuch „Programmiertechniken“

Behoben: Die USW-Beispiele für WINDOWS hatten schon geraume Zeit kein vernünftiges Bild in ***uctable*** dargestellt. Die Beispiele zeigen jetzt Icon in der richtigen Größe an.

Behoben: Tasten, die der IDM für die Navigation verwendet (bspw. `TAB`, `Cursor links` usw.), wurden unter MICROSOFT WINDOWS nicht an Kindobjekte des USW-Widgets weitergeleitet obwohl im **UC_I_querykey** Inquire die Taste als benötigt gekennzeichnet wurde.

Behoben: Das umgebende IDM Objekt von USW Widgets besitzt jetzt keinen Rahmen mehr. Das USW Widget sollte für seinen Rahmen selbst zuständig sein.

6.12 IFD

Änderung: Die IFD Option wird nicht mehr unterstützt.

6.13 Demos

IDM-13189: In der README Datei der Beispiele stand nur, dass das Project „usw\usw“ nicht gestartet werden kann. Ein Hinweis auf das Projekt „usw\ucsample“ fehlte.

Neu: USW Beispiel ***ucgrid*** eingefuehrt um *HiDPI*-Support für USW Widgets aufzuzeigen.

Behoben: Das **ucframe** Widget der USW-Beispiele hatte unter WINDOWS bei der Größenberechnung die gesetzte Schriftart nicht berücksichtigt, dadurch wurde Überschrift abgeschnitten.

6.14 Dokumentation

Neu: Kapitel „HighDPI UnterstützungSupport“ im Handbuch „Programmiertechniken“ hinzugefügt.

Änderung: Dokumentation zu DM_GetToolkitData und DM_GetToolkitDataEx grundlegend überarbeitet. Dokumentation für Toolkit datena-S strukcture DM_ToolkitDataArgs im Handbuch „C-Schnittstelle - Grundlagen“ erstellt.

IDM-13198: Wichtige Hinweise, dass die Option **opt_hscroll** am poptext unter MICROSOFT WINDOWS problematisch ist, nun in die entsprechenden Kapitel des Attribute Handbuch .options[enum] sowie Objekte Handbuch poptext übernommen.

7 A.06.02.o

7.1 Kern

IDM-13142: Spezielle Konstellationen bei vererbten Referenzen von Tablefield-Attributen führen nun beim Entaden des Modules aus dem die Referenzen kommen nicht mehr zum ASSFAIL.

IDM-13157: Beim Herausschreiben von Interface-Dateien werden nun die korrekten Argument-Bezeichner bei Methoden ausgegeben.

IDM-13158: Beim Herausschreiben von Interface-Dateien werden nun durch überschriebenen Methoden, die aus unterschiedlichen Modulen stammen, erkannt und nicht doppelt im Interface aufgeführt.

IDM-13159: Beim Laden von ASCII-Modulen/Dialogen erfolgt beim Auftreten von manchen Syntax-Fehler auf jeden Fall die Rückgabe als Fehler.

IDM-13093: Fehlerboxen im IDM enthalten nun einen **Cancel**-Button mit dem weitere Fehler (die z.B. während des Ladevorganges auftreten) unterdrückt werden können. Die unterdrückten Fehlermeldungen gelangen also in die Logdatei (als [E:-Meldung]).

Fehlermeldungen des IDM werden systemtypisch unter Windows in einer Dialogbox ausgegeben, unter UNIX/Linux hingegen ins Log- oder Tracefile. Zwar kann mittels Startoption **-IDMerrwinfile** auch unter Windows eine Umleitung in eine Datei passieren, falls diese Option jedoch nicht genutzt wird, kann es u.U. zu sehr vielen Fehlermeldungen führen, die meist den Anwender dazu verleiten das IDM-Anwendungprogramm zu „killen“. Die Fehlermeldungs-Dialogbox wurde nun mit einem Cancel-Button ausgestattet, sodass ein weiteres Aufpoppen von Fehlermeldungen in einer Dialogbox unterbunden wird. Statt dessen gelangen diese Meldungen als [E:...]-Meldungen in die Logdatei. Ein Abbruch der Anwendung findet aber nicht statt bzw. ist auch nicht möglich, sondern lediglich die Unterbindung der Dialogboxen. Wenn gewünscht kann dies durch den Anwendungsprogrammierer in einem Error-Handler passieren. Dabei ist das Laden eines Dialoges/Modules als ein „atomarer“ Block behandelt. D.h. das Unterbinden der Dialogboxen wirkt bis zum Ende diese Blocks. Grundsätzlich ist ein Unterdrücken von ASSFAIL-Dialogboxen nicht möglich.

7.2 Unix

Neu: Neue Plattform: **RHEL 9.0 for 32/64bit** mit Compiler **GCC11.2.1**, **Motif 2.3.4** und **Qt5.15**

Mit **RHEL9** werden standardmäßig keine rsh-server-Pakete ausgeliefert. Für die Nutzung von **DDM** ist dann zwingen die Verwendung von **SSH** für das Starten eines DDM-Servers nötig.

Der IDMED nutzt, falls keine Xft-Fonts vorhanden sind bzw. die Xft-Unterstützung deaktiviert wurde, die X11-Lucida-Fonts in der Unicode-Variante iso10646. Wir empfehlen deshalb die Installation der folgenden Pakete:

- xorg-x11-fonts-100dpi
- xorg-x11-fonts-75dpi

Da **RHEL9** ohne statische **PCRE**-Library geliefert wird, kann der **IDM/ IDMED** die PCRE-Funktionalitat nur dynamisch anbinden. Außerdem wird bei einer Desktop-Installation z.T. nicht automatisch **Motif** installiert. Wir empfehlen außerdem die Prüfung auf Installation der Pakete:

- motif
- libXpm
- pcre
- libxslt
- libxml2
- libssh

Da der **IDM** für **Qt** bestimmt Pakete verwendet, welche nicht zwangsläufig mit den Qt-Haupkomponenen installiert werden, empfehlen wir die Prüfung auf und ggf. die Installation der folgenden Pakete:

- libQt5X11Extras

Hinweis: Der **IDM** für **Motif/ Qt** ist als X11-Anwendung konzipiert und beinhaltet keine expliziten Support für den Display-Server WAYLAND. WAYLAND ist als X11-Nachfolger designed und beinhaltet einen eine „kompatible“ Umsetzung des X11-Protokolls. Da dies aber nicht zu 100% funktioniert, empfehlen wir **IDM**-Anwendungen unter einem X11-Server laufen zu lassen und z.B. beim Login die Desktop-Variante mit nativen X-Server auszuwählen.

Hinweis: Unter **RHEL9** wird standardmäßig der UI-Style ADWAITA angewandt. Dieser führt jedoch zu unerwünschten und unschönen Darstellungseffekten bei Qt-Anwendungen. Wir empfehlen daher für mit dem **IDM** für **Qt** erstellten Anwendungen den Einsatz des Styles FUSION.
Der verwendete Style kann über die Systemeinstellung geändert werden (sofern dort unterstützt). Alternativ kann über die Umgebungsvariable QT_STYLE_OVERRIDE der zu verwendende Style für ***alle*** Qt-basierten Anwendungen festgelegt werden oder man gibt direkt beim Start der Anwendung den *-style*-Parameter mit (z.B.: `idm -style Fusion MyDialog.dlg`).

7.3 Windows

IDM-13151: Das Setzen des Inhalts eines edittext (editierbarer Text) Objektes konnte verloren gehen, wenn es direkt davor Setzungen gab, die dafür sorgten, dass das edittext Objekt den Fokus verlor. Jetzt werden dynamische Attributwerte im WSI so lange gemerkt wie das Objekt sichtbar ist. Ein Zurückschreiben erfolgt erst beim unsichtbar Schalten, da dieser Vorgang synchronisiert abläuft. Hierdurch tritt das Problem nicht mehr auf, da es kein vorzeitiges Speichern von bereits geänderten Attributwerten gibt.

IDM-13143: Ein Model konnte im **IDM Editor** nicht angezeigt werden, wenn es gerade aus einer Instanz erzeugt wurde. Nach einem Neuladen des Dialogs wurde das Model angezeigt.

7.4 Qt

IDM-13160: Der Inhalt einer Tablefieldzelle konnte unsichtbar sein, wenn die Vorder- oder Hintergrundfarbe explizit für eine Zelle gesetzt wurde (Setzung mit Index). Ursächlich war derOptimizer von **g++**.

IDM-13146: Das tablefield Objekt hatte die Option *sel_header* bei .selection[enum] nicht beachtet. Dies führte dazu, dass Zellen im Header aktiviert werden konnten und auch am Drag & Drop teilnahmen. Die Header sind jetzt auf nicht selektierbar gesetzt. Es bleibt aber zu beachten, dass das Qt Selektions Model verwendet wird.

Hinweis für Qt-spezifisches Verhalten des tablefield Objektes:

Es werden aktivierte Zellen deaktiviert, wenn auf eine Header-Zelle bei *.selstyle[sel_header* = *false* geklickt wird.

Bei *.fieldfocusable = true* erfolgt auch bei Klick auf eine insensitive Zelle eine Umaktivierung. Es werden die entsprechenden *activate* und *deactivate* Ereignisse erzeugt, es kommt aber kein *select* Ereignis mit Index.

Wenn keines der Element *sel_single*, *sel_row* und *sel_single* beim .selection[enum] Attribut den Wert true besitzt, wird *sel_single* verwendet.

IDM-13144: Wenn ein tablefield Objekt unsichtbare Zeilen oder Spalten enthielt, dann konnte es vorkommen, dass die Header Zeilen oder Spalten verschwanden. Dies war abhängig on der Scrollposition und trat dann auf, wenn unsichtbare Zeilen oder Spalten nach oben bzw. link aus dem Darstellungsbereich gescrollt waren.

IDM-13153: Wenn bei einer scrollbar initial ein .curvalue und .maxvalue größer 100 angegeben wurde, so wurde der .curvalue auf 100 zurück gesetzt.

Behoben: Abstürze unter 64bit beim Zeichnen von Hintergründen mit Mustern beseitigt.

Hinweis: Objekte werden standardmäßig mit den vom Desktop-Style vorgegebenen Werten durch das WSI/ Toolkit dargestellt. Besonders unter neueren Desktops kann es u.U. vorkommen, dass insbesondere Rahmen, Hintergründe oder Scrollbars usw. nun abweichend, gar nicht oder sogar dynamisch dargestellt werden. Z.B.: Groupboxen könnten ohne Rahmen dargestellt werden, Scrollbars verändern ihr Aussehen beim Überfahren mit der Maus.
Der verwendete Style kann über die Systemeinstellung geändert werden (sofern dort unterstützt). Alternativ kann über die Umgebungsvariable QT_STYLE_OVERRIDE der zu verwendende Style für alle Qt-basierten Anwendungen festgelegt werden oder man gibt direkt beim Start der Anwendung den *-style*-Parameter mit (z.B.: `idm -style Fusion MyDialog.dlg`).

Neu: zusätzliche Option am filereq Objekt fuer LINUX/QT.

Normalerweise verwendet der IDM für das filereq-Objekt einen nativen Dialog des System. Dieser wird u.A. vom Desktop bereitgestellt und kann sich dadurch bzgl. dem Funktionsumfang und Verhalten deutlich unterscheiden, ist aber besser in den Desktop integriert und deshalb normalerweise zu präferieren.

Durch Setzung der Option *.options[fro_nativedialog]:=false* kann nun die Benutzung des Qt/Toolkit-spezifischen Dialoges forciert werden.

Auf anderen Systemen (WINDOWS/ MOTIF) hat diese Option aktuell keine Bedeutung. Der **IDMED**, **IDMDBG** sowie **IDMTA** benutzten nun den Qt/Toolkit-spezischen Dialog für das Laden/ Speichen/ Selektieren von Dateien/ Verzeichnissen.

7.5 DDM (Netz)

IDM-13148: Es konnte vorkommen, dass auf der Darstellungsseite kein *finish* Ereignis ausgelöst wurde, nachdem die Anwendungsseite abgebrochen wurde. Dies lag daran, dass die Anwendungsseite noch eine "Shutdown"-Anforderung geschickt hat. Jetzt wird diese Anforderung auch als Fehler betrachtet, wodurch dann das *finish* Ereignis ausgelöst wird. Ein reguläres Ende wird von der Darstellungsseite initiiert, so dass es hier nicht zu der **Shutdown**-Anforderung kommt.

IDM-13167: Mit der Einführung der SSL Unterstützung hat sich das Protokoll des Verbindungsaufbaus bei *.exec* geändert. Die zusätzliche Ausgabe wurde auch an die Netzwerk-Handler Callbackfunktion weitergeleitet. Diese Ausgabe wird jetzt als *DM_NET_CONNECTMESSAGE* gekennzeichnet damit diese von der normalen Ausgabe *DM_NET_MESSAGE* unterscheidbar ist.

- *DM_NET_CONNECTMESSAGE*: STDOUT-Ausgabe zum Verbindungsaufbau der mittels ".exec" gestarteten Anwendungsseite. Muss unverändert an IDM weitergeleitet werden.
- *DM_NET_MESSAGE*: STSOUT-Ausgabe der mittels *.exec* gestarteten Anwendungsseite. Die Ausgabe wird Zeilenweise an den NetHandler weitergegeben (Ausnahme wenn der Socket geschlossen wird). **Achtung:** Als erstes wird ein 0-Byte gesendet (rsh-Protokol ?) und der IDM schickt einen Zeilenumbruch vor dem Verbindungsaufbau.

Bei der **DM_NetHander** Funktion gibt es für die NetHandler Callbackfunktion jetzt den Kommunikationsvorgang (Strukturelement "action") *DM_NET_CONNECTMESSAGE* der die Stdout-Ausgabe zum Verbindungsaufbau der mittels *.exec* gestarteten Anwendungsseite kennzeichnet. Außerdem wurden einige kleinere Fehler in der Weiterleitung der Stdout-Ausgabe an die NetHandler Callbackfunktion behoben. Hierduch kann es zu Änderungen bei *DM_NET_MESSAGE* kommen:

- 0-Bytes wurden bisher zu '@' konvertiert, diese bleiben jetzt als 0-Bytes erhalten.
- Es konnte vorkommen, dass die Callbackfunktion verfrüht, vor Erhalt des Zeilenendes, aufgerufen wurde. Dies geschieht nun nur noch beim Schließen der Verbindung.
- Es konnte vorkommen, dass Daten nicht an die Callbackfunktion weitergeleitet wurden.

7.6 COBOL

IDM-13149: Auf Nicht-*Intel* Architekturen wurden zum Teil Parameter an COBOL Record-Funktionen falsch übergeben. Dies lag daran, dass die von **+writetrampolin** erzeugte Funktion Parameter, die keine Records waren, zum Teil direkt an die COBOL Funktion weitergegeben hatte. Dies war

falsch, wenn der von COBOL verwendete Datentyp kleiner war. Dies trat zum Beispiel bei *boolean* auf. Jetzt wird jeder Parameter erst einmal auf den richtigen Datentyp umkopiert.

IDM-13161: Die COBOL **DM-getsep** und **DM-setsep** Buchstaben wurden auf der Netzwerk Anwendungsseite nicht richtig gesetzt. Dies lag an einer Diskrepanz der **DM_Control** Bearbeitung.

Behoben: Auf Nicht-*Intel* Architekturen wurde das Endzeichen **DM-setsep** nicht erkannt. Hierdurch wurde das Endezeichen und eventuell folgende Zeichen an den IDM übertragen. Jetzt wird das Stringende wieder erkannt.

7.7 Dokumentation

IDM-13154: Beschreibung des checkbox Objekts bzgl Tab-Navigation verbessert und in der englischen Dokumentation aktualisiert.

IDM-13124: Beschreibung für **CP_ucp** und **DMF_SetUserCodePage** in C-Funktionsbeschreibung **DM_Control** und **DM_ControlEx** verbessert.

IDM-13165: Hinweise zur Kompatibilität/ Unterschiede vom tablefield Objekt unter Qt in die Beschreibung aufgenommen. Englische Beschreibung des Tablefield-Objekts aktualisiert.

IDM-13128: Fehlende Beschreibung für das indizierte Attribut .label[anyvalue] hinzugefügt.

8 A.06.02.n2

8.1 Unix

IDM-13147: Zusätzliche **AIX 7.1** Plattform verfügbar, kompiliert mit *xlC16.1* (32bit und 64bit).

8.2 DDM (Netz)

IDM-13148: Es konnte vorkommen, dass auf der Darstellungsseite kein finish-Ereignis ausgelöst wurde, nachdem die Anwendungsseite abgebrochen wurde. Dies lag daran, dass die Anwendungsseite noch eine "Shutdown"-Anforderung geschickt hat. Jetzt wird diese Anforderung auch als Fehler betrachtet, wodurch dann das *finish* Ereignis ausgelöst wird. Ein reguläres Ende wird von der Darstellungsseite initiiert, so dass es hier nicht zu der **Shutdown**-Anforderung kommt.

8.3 COBOL

IDM-13149: Auf Nicht-*Intel* Architekturen wurden zum Teil Parameter an COBOL Record-Funktionen falsch übergeben. Dies lag daran, dass die von +writetrampolin erzeugte Funktion Parameter, die keine Records waren, zum Teil direkt an die COBOL Funktion weitergegeben hatte. Dies war falsch, wenn der von COBOL verwendete Datentyp kleiner war. Dies trat zum Beispiel bei *boolean* auf. Jetzt wird jeder Parameter erst einmal auf den richtigen Datentyp umkopiert.

Behoben: Auf Nicht-*Intel* Architekturen wurde das Endzeichen **DM-setsep** nicht erkannt. Hierdurch wurde das Endezeichen und eventuell folgende Zeichen an den IDM übertragen. Jetzt wird das Stringende wieder erkannt.

9 A.06.02.n

9.1 Kern

IDM-9956: Ersetzung des unsicheren Funktionsaufrufes tempnam() um Linker-Warnungen auf Linux zu vermeiden.

IDM-13062: Die :openpopup()-Methode taucht nun nicht mehr fälschlicherweise in der idmlangdef-Datei für die ***Window***-Objektklasse auf.

IDM-13085: Attributsbeschreibung/ Sprachdefinition für indizierten Zugriff auf .label-Attribut korrigiert.

IDM-13087: Der **this**-Zugriff wurde in der Fehleranalyse inkorrekt weiterverwendet. Bei Modell-Regeln/Methoden wird nun der Modell-Scope weitergegeben da ein konkretes **this** nicht bestimmbar ist.

IDM-13088: Eine *anyvalue*-Variable darf in Regeln auch indiziert benutzt werden und erzeugt in der Fehleranalyse nun keine Typ-Warnung mehr. Typ-Warnungen werden in einem solchen Fall nur für einen Index, die einen listenartigen Typ haben, rausgegeben.

IDM-13095: Modulverändernde Konstruktionen wie **use <Child>** und die .application-Setzung bei Imports werden nun vom Builder-Server korrekt behandelt. Solche Konstruktionen führen dazu, dass die veränderten Module nicht wiederverwendet und erneut geladen werden müssen.

IDM-13107: Diverse Bugs in der Fehleranalyse behoben. (Andere Zugriffe auf Variablen außer .self/.value liefern nun den korrekten Typescope für ein Attributszugriff wie .dialog. Die Initialisierung von statischen Variablen erfolgt nun als undefined. Außerdem werden Vergleichen zwischen Id=null oder Id<>null nun in der Werteauswertung ebenso abgehandelt.)

IDM-13113: Zugriff auf das skalare Tablefield-Attribut .fieldfocus führt nun nicht mehr zu einer Warnung des Fehleranalysers da ein falscher Datentyp angenommen wird.

IDM-13118: Behoben wurde ein Absturz beim IDM unter 64bit, wenn bei einem Secret-Format (S*) und einem leeren Inhalt eine weitere Linksnavigation ausgeführt wurde.

IDM-13104: Geerbte Methoden von nicht exportierten Objekten werden nun beim Schreiben der Interface-Datei korrekt geklammert herausgeschrieben und Verursachen nun keinen Interface-Fehler mehr.

IDM-13071: Menuboxen mit geerbtem Attribut .visible=true werden nun bei einem dynamischen:create() auch sichtbar gemacht.

IDM-11753: Keine Änderung: Unklare Resolving-Fehler während dem Builder-Prozess treten nicht mehr auf. Es zu beachten, dass Resolving-Folgefehler entstehen, wenn durch ein Ladeabbruch (z.B. ein fehlendes Modell) die nachfolgenden Dialogzeilen nicht mehr geparst werden.

IDM-13123: Record-Funktionen in DLL's konnten nicht dynamisiert aufgerufen werden. Jetzt wird die Aufrufmethode für Record-Funktionen über der Funktionskennung bestimmt, so dass diese auf jeden Fall aufgerufen werden.

9.2 Unix

IDM-13099, : Die Installation der Microfocus COBOL Option unter Unix brach mit einem Fehler ab, dies führte dazu, dass die COBOL Copy-Strecken und Beispiele fehlten.

IDM-13070:, Das COBDEMO-Verzeichnis wird bei der Installation unter Unix nun wieder richtig entpackt.

9.3 Motif

IDM-13084: Absturz des IDM beim Lesen einer GIF-Datei mit einer nicht unterstützten **local color map** wird vermieden und eine Fehlermeldung ausgegeben.

9.4 Windows

IDM-11904: Keine Änderung: Multiscreen Support für IDM unter Windows. Die Erweiterung der Anzeige auf mehrere Monitor erfolgt in Windows 10 durch die Anordnung der Monitore in einem großen virtuellen Adressraum. Der primäre Monitor liegt bei 0/0. Wo ein Fenster erscheint wird durch seine Koordinaten geregelt, dies funktioniert auch mit dem IDM. Achtung negative Koordinaten sind erlaubt und möglich.

IDM-12934: Der gesetzte Wert des ***clipboard*** Objektes wurde nicht in das Windows Clipboard übernommen. Das ***clipboard*** Objekt ist jetzt mit der Drag & Drop Funktionalität gekoppelt, so dass Werte jetzt übernommen werden.

IDM-13069: Wurden Bilder gelöscht, die von einem treeview Objekt angezeigt wurden, dann wurden diese sofort wieder angelegt. Dies führte Fehlermeldungen, wenn Module entladen wurden.

IDM-13074: Beim Kompilieren gab es Compiler Warnung **C4161** "#pragma pack(pop...): more pops than pushes". Dies lag daran, dass keine Präprozessorsymbole definiert waren. Jetzt wird aus den vordefinierten Präprozessorsymbolen die Architektur bestimmt, bevor das "structure packageing" gesetzt wird.

IDM-13106: Die Breite eines edittext Objektes wurde zu klein berechnet, wenn keine Breite vorgegeben war. Die Berechnung ist jetzt korrigiert. Das Problem kann nur sicher bei einem Font mit fester Breite erkannt werden.

IDM-13114: Beim maximieren eines Fensters wurde der Hintergrund mehrfach gezeichnet. Das Problem wurde unter anderem von Kindobjekten verursacht, die wegen durchsichtiger Bereiche ein Neuzeichnen des Vaterobjektes verursachten. Es kann noch zu mehrfachem Zeichnen kommen, aber wesentlich weniger häufig als davor.

IDM-13116: Die editierbaren Objekte edittext und poptext ließen sich im IDM-Editor nicht mehr mit der linken Maustaste verschieben.

9.5 Qt

IDM-13052: Initiale Abstürze beim poptext mit Stil *listbox* wurden behoben. Außerdem wurde behoben, dass in diesem Style auch das initial angegebene .activeitem nicht umgesetzt wurde.

IDM-13078: Es wurden Warnungen und Fehlermeldungen in Zusammenhang mit dem Attribut .menu behoben, wenn Menüs dynamisch angelegt wurden.

IDM-13115: Gruppierungsobjekte, die ein Pattern-Tile als Hintergrund sowie .fgc und .bgc gesetzt haben, wenden diese nun auf das Tile an. Das Hintergrund-Tile wird somit nun farblich korrekt dargestellt.

IDM-13117: Gruppierungsobjekte lassen sich im IDM-Editor nun wieder verschieben und in der Größe ändern. Ebenso lassen sich poptext- und spinbox-Objekte nun überall anfassen und verschieben (also auch an den "inneren" Bereichen).

Behoben: Das window-Objekt im IDM-Editor hatte nicht auf Doppelklick reagiert um die Properties zu öffnen.

Behoben: Tooltips der editierten Objekte im IDM Editor wurden bisher nie angezeigt. Nun wird das Label angezeigt, wie bereits unter Windows und Motif.

9.6 Debugger

IDM-13105: Beim Beenden des Debuggers kam es zu einem Absturz, wenn man den Debugvorgang durch Beenden der laufenden Anwendung beendete.

9.7 DDM (Netz)

IDM-13098: Bei Verwendung des **DMF_CobolBindForLoader** Mechanismus kam es auf 64-Bit Architekturen zu einem Network Protocol Error.

9.8 COBOL

IDM-13103: COBOL-Record-Funktionen wurden auf der Anwendungsseite einer Netzwerkanwendung nicht aufgerufen, wenn die COBOL-Record-Funktionen mit **DMF-CobolBindForLoader** an dieNetzwerkanwendung gebunden wurden. Jetzt funktioniert der dynamische Aufrufmechanismus auch auf der Anwendungsseite.

IDM-13102: COBOL-Funktionen wurden auf der Anwendungsseite einer Netzwerkanwendung nicht aufgerufen, wenn die COBOL-Funktionen mit **DMF-CobolBindForLoader** an die Netzwerkanwendung gebunden wurden. In der Tracedatei wurde ein ***"Binding type mismatch"*** Fehler ausgeben. Jetzt wird der Anwendungstyp richtig gesetzt, so dass dieses Problem nicht mehr auftritt.

IDM-13101: Wenn COBOL-Record-Funktionen sowohl mit **DMF-CobolBindForLoader** als auch mit der generierten **RECINIT<label>**Funktion angemeldet wurden, dann wurden diese nicht aufgerufen. Wenn "DMF-CobolBindForLoader" verwendet wird ist der Aufruf der generierten

RECINIT<label>Funktion optional. Insbesondere spielt die Reihenfolge der Aufrufe keine Rolle mehr. Zu empfehlen ist den Aufruf wegzulassen wenn **DMF-CobolBindForLoader** verwendet wird.

IDM-13097: Es gab offizielle und interne Attribute die sich nur in der Groß-/Kleinschreibung unterschieden. Der internen Variante, wird jetzt eine spezielle Endung angehängt, so dass die Namen eindeutig sind.
Außerdem ist zu beachten, dass die "_FIRST" und "_LAST" Definitionen für die einzelnen Enumbereiche unter COBOL mit "-FIRST-VALUE" und "-LAST-VALUE" benannt werden.

IDM-13096: Cobol-Funktionen die über **BindTrueLoader** angebunden wurden, werden nun korrekt aufgerufen - sowohl in der Anbindung als Modul-Funktion auf ein Applikations-Objekt über das .application-Attribut am Import als auch über das neue .masterapplication[enum]-Attribut am Dialog/Modul.

9.9 Demos

Behoben: Compilierprobleme der Beispiele unter Motif bereinigt.

Behoben: Compilierprobleme der Beispiele unter Qt bereinigt.

9.10 Dokumentation

IDM-8699: Anmerkung hinzugefügt, dass sowohl .titlebgc als auch .titlefgc in den aktuellen WSIs nicht unterstützt werden.

IDM-9160: Hinweise zu .fgc und .bgc in den bereits existierenden extra Abschnitt für Windows beim notepage Objekt eingetragen.

IDM-10952, IDM-12602: Release Notes sind in allen angebotenen Sprachen bis zur aktuellen Version verfügbar.

IDM-11784: Die beiden Attribute .errorcode und .systemerror sind nun beschrieben. Fehlende Querverweise und Verknüpfungen wurden nachgetragen.

IDM-12298: Hinweis zu System-Menütaste unter Windows im Attributshandbuch hinzugefügt. Ebenso beim menubox Objekt in den Abschnitt "Hinweise zu MSW".

IDM-12791: Keine Änderung: Keine Referenzen zu Windows 3.1 und MSVC 1.5 in der Dokumentation mehr.

IDM-12940: Datentyp *class* in die Datentyp-Tabelle von Kapitel „Datentypen“ des Regelsprache Handbuchs hinzugefügt.

IDM-13014: Keine Änderung: Schlüsselwort **child** ausreichend dokumentiert.

IDM-13033: Im Handbuch Kurzreferenz bei „Vordefinierte Ereignisse“ eingetragen, dass unter Qt das *help*-Ereignis nur mittels `Shift`+`F1` aufgerufen wird. Ebenso bei Handbuch Regelsprache unter „Ereignisse“ sowie im Handbuch Profiler Kapitel „Das Menü“.

IDM-13091: Keine Änderung: Methode :represent() ist ausreichend dokumentiert.

IDM-13092: Formulierungen für die Beschreibung von .masterapplication[enum] optimiert. Darüber hinaus auch noch Beschreibung aus den Release Notes in das Kapitel Programmiertechniken „Objekt Application“ und einen Verweis auf Regelsprache „Funktionen (functions)“ übernommen.

10 V.06.02.m2

10.1 COBOL

IDM-13096: Cobol-Funktionen die über **BindTrueLoader** angebunden wurden, werden nun korrekt aufgerufen - sowohl in der Anbindung als Modul-Funktion auf ein Applikations-Objekt über das .application-Attribut am Import als auch über das neue .masterapplication[enum]-Attribut am Dialog/Modul.

10.2 DDM (Netz)

IDM-13098: Bei Verwendung des **DMF_CobolBindForLoader** Mechanismus kam es auf 64-Bit Architekturen zu einem Network Protocol Error.

10.3 Unix

IDM-13099: Die Installation der Microfocus COBOL Option unter Unix brach mit einem Fehler ab, dies führte dazu, dass die COBOL Copy-Strecken und Beispiele fehlten.

11 A.06.02.m

11.1 Kern

Neu: Einführung des .masterapplication[enum]-Attributes für die Applikation-Zuordnung von Modul-Funktionen für mit **use** Modularisierte Dialoge um eine Funktionalität analog zum .application-Attribut am ***import***-Objekt zu ermöglichen.

	Bezeichner		Datentyp
Regelsprache	**.masterapplication**		object [application]
C	AT_masterapplication		DT_application
COBOL	AT-masterapplication		DT-application
Klassifizierung	Objektspezifisches Attribut		
Objekte	dialog, module		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	–

Ein spezieller Fall für die Benutzung von Funktionen in modularisierten Dialogen stellt die Verwendung des *.application*-Attribut am ***import***-Objekt dar um alle Funktionen eines Modules von „außen“ an eine spezielle Applikation zu koppeln. Diese Vorgehensweise ist im Kapitel „Programmiertechniken“/ „Modularisierung“ / „Objekt Application“ beschrieben.

Beim Importieren von Modulen über **use** steht diese Vorgehensweise allerdings nicht zur Verfügung. Als Alternative bzw. Ergänzung kann allerdings die Koppelung von Funktionen über das *.masterapplication*-Attribut am Modul bzw. Dialog verwendet werden. Damit sind Funktionen die direkt unter einem Modul/ Dialog definiert sind einer Applikation zuordenbar. Falls keine Zuordnung am Modul vorhanden ist wird die Zuordnung vom Dialog herangezogen. Das Attribut kann auch mit einem enum-Wert indiziert werden. Dieser Index-Wert sollte aus dem Bereich *lang_default…lang_java* oder *func_normal..func_data* sein. Dadurch lassen sich Funktionen entsprechend ihrer Sprache- oder Funktionsart unterschiedlichen Applikationen zuordnen. Eine explizite Zuordnung (auch eine null-Setzung) am Modul überschreibt/ überdeckt immer die Zuordnung des Dialogs.

Beispiel

Modularisierter Dialog mit den Funktionen im Modul „Functions.mod“, bei dem COBOL-Funktionen der Server-Applikation „ApplServer“ zugeordnet werden und alle anderen Funktionen der „dynlib“-Applikation „ApplLocal“.

```
// Dialog.dlg
dialog Dlg
{
  .masterapplication ApplLocal;
}

application ApplLocal {
  .transport "dynlib";
  .exec "libusercheck.so";
  .active true;
}

use Functions;

on dialog start
{
  variable string Username := setup.env["USER"];
  if  ValidateUser(Username) then
    print "Age = "+GetAge(Username);
  endif
  exit();
}

// Functions.mod
module Functions
{
   .masterapplication[lang_cobol] ApplServer;
}

use Server;

record RecUser
{
  string[50] FirstName;
  string[50] LastName;
  integer Age;
}

export function boolean ValidateUser(string Username);
function cobol integer GetUserProfile(string[50] Username,
                                      record RecUser output);

export rule integer GetAge(string Username)
{
  if GetUserProfile(Username, RecUser)>0 then
    return RecUser.Age;
  endif
```

```
  return 0;
}

// Server.mod
module Server

export application ApplServer {
  .connect "server:4712";
  .active true;
}
```

PLUGIN-12377: Im Idm EclipsePlugin wurde der Datentyp *type* vorgeschlagen, den es nicht gibt.

IDM-13082: Enthalten exportierte Regeln von Modulen einen Default-Wert - also optionale Parameter - wird dies beim Herausschreiben der Interface-Datei (Version 3) beruecksichtigt und der Parameter korrekt als "optional" markiert.

IDM-13089: Methodendefinitionen von **:represent()**, **:retrieve()** und **:apply()** korrigiert, so dass fehlerhafte "Wrong number of arguments" Meldungen der Fehleranalyse nicht mehr passieren sollten.

IDM-13090: Indizierter Zugriff bei einer globalen Variablen mit einem listenartigen Datentyp wird nun in der Fehleranalyse korrekt mit dem Wertetyp der Liste weiterbehandelt und erzeugt nun keine "bad type combination"-Meldung mehr.

11.2 Debugger

IDMDBG-132: Expressions, die über mehrere Zeilen gingen, konnten nicht ausgewertet werden.

12 A.06.02.I

12.1 Kompatibilität

Das Format der Interface-Dateien hat sich geändert. Alle Interface Dateien sollten neu generiert werden.

12.2 Kern

Neu: Die erweiterte Fehleranalyse erkennt nun auch unbenutzte, importierte Module, die per **use**- oder **import**-Anweisung importiert werden. Ein importiertes Modul gilt als unbenutzt, wenn keine exportierten Objekte daraus referenziert werden. Unbenutzte Imports werden mit dem neuen Fehlercode DME_ImportNotUsed (*„Imported module never used“*) gemeldet.

Ist die erweiterte Fehleranalyse aktiv (z.B. explizit durch `-IDMerroranalysis=true` eingeschaltet), prüft der IDM für jedes geladene ASCII-Modul, ob es ***import***-Objekte im Modul oder Dialog gibt, aus denen keine exportierten Objekte referenziert werden. Diese werden dann als „unbenutzt“ gemeldet.

Dabei werden importierte Module nicht geprüft, wenn sie wiederum exportiert werden oder wenn ihre Attribute *.load* oder *.static* auf *true* gesetzt sind.

Imports von Modulen, die besondere exportierte Objekte enthalten, gelten immer als „benutzt“. Zu den besonderen Objekten zählen dabei Default-Objekte und Acceleratoren.

Eine Referenzierung des Imports zählt ebenso als Verwendung. Allerdings kann aus technischen Gründen die Referenzierung eines ***import***-Objektes direkt im *.userdata*-Attribut eines Dialogs oder Moduls nicht als Verwendung gezählt werden.

Neu: Die Interface-Dateien enthalten nun die Definitionen aller benutzerdefinierten Attribute und Methoden, sodass die erweiterte Fehleranalyse kein unnötiges Nachladen von Modulen forciert.

Das Herausschreiben als Kommentare in der Form „// @define“ entfällt damit. Die Interface-Dateien enthalten deshalb den Hinweis „version 3“ in der Zeile „interface of module…“.

Diese Definitionen sind notwendig und sinnvoll um die erweiterte Fehleranalyse zu unterstützen und ein unnötiges Nachladen zu vermeiden bzw. um fehlerhafte Zugriffe und Verwendungen von Methoden und Attributen zu erkennen.

Ein Prüfung auf Konsistenz mit den benutzerdefinierten Attributen und Methoden des nachgeladenen Moduls findet nicht statt. Eine Inkonsistenz führt hier typischerweise zu Warnungen oder Fehlern bei der Fehleranalyse oder Ausführung.

Neu: In die Typprüfungen bei Zuweisungen, Ausdrücken, Parametern und Rückgabewerten fließt nun die Wertepropagierung aus der Variablenanalyse ein. Außerdem wird die Identifikation von Variablen, Objekten und Kindzugriffen herangezogen. Mögliche Fehler sind durch die Wörter *„maybe“* und *„possible“* in den Fehlermeldungen gekennzeichnet.

Objekt-Referenzierung und Attributzugriffe

Ein absoluter Pfad, der beim Laden des Moduls oder Dialogs komplett aufgelöst werden kann, wird als konstanter Wert (Objekt-ID) behandelt. Dadurch können solche Werte auch für die Datenflussanalyse und Typprüfung direkt herangezogen werden.

Durch überschreibbare **:set()**- und **:get()**-Methoden, durch den Aufruf von Regeln und Funktionen oder durch die Rückkopplung mit dem Datenmodell und Shadow-Attributen können keine konkreten Werte von Attributen in die Fehlerprüfung einfließen.

Paradigma

Für die Fehleranalyse wird davon ausgegangen, dass Attribute mit Ihrem Typ vordefiniert werden, eine Vater-Kind-Hierarchie eingehalten wird und Regeln typischerweise auf die Kind-Objekte zugreifen. Insofern sind die wichtigsten Zugriffsmöglichkeiten auf Kinder und Väter auch initial sinnvoll vorhanden. Die **:set()**- und **:get()**-Methoden sollten vom Anwender ebenso nicht missbraucht werden, um beliebige Wertumwandlungen durchzuführen. Mit diesen Annahmen kann die Fehleranalyse die Typinformation von vordefinierten und benutzerdefinierten Attributen, Regeln und Methoden unter Vorbehalt verwenden. Dieser Vorbehalt drückt sich dann durch *„maybe“* oder *„possible“* bei möglichen Fehlermeldungen aus.

Handelt es sich um einen relativen Pfad mit bezeichneten Kind-Objekten, die erst zur Laufzeit ausgewertet werden können, so wird das zur Ladezeit referenzierte Objekt als „maybe“-Gültigkeitsobjekt verwendet, um so Typprüfungen z.B. bei der Zuweisung durchzuführen. Außerdem können so Hinweise auf möglicherweise fehlende Attribute gegeben werden.

Beispiel

Dieses Beispiel zeigt absolute und relative Attributzugriffe mit den zu erwartenden Fehlern (WARN-Meldungen).

```
dialog D
model record MRecUser {
  string Name := "";
  integer Age := 1;
}

window WiMain {
  edittext EtName {}
  MRecUser RecUser {}
}

on dialog start {
  !! WARN: possible value mismatch type - integer conflicts with expected
string
  WiMain.RecUser.Name := 4711;
  !! WARN: possible value mismatch type - text conflicts with expected
integer
  this.WiMain.RecUser.Age := "uralt";
  !! WARN: Child or attribute maybe undefined - .Alter
```

```
  this.WiMain.RecUser.Alter := "uralt";
  !! WARN: possible value mismatch type - integer conflicts with expected
string
  EtName.content := D.timeout;
  !! WARN: possible value mismatch type - integer conflicts with expected
boolean
  this.WiMain.EtName.visible := 1;
  exit();
}
```

Prüfungen bei Regeln und Methoden

Durch die Objekt-Erkennung ermittelt die Fehleranalyse die vorhandenen Methoden und kann fehlerhafte Typen, Argumente und Rückgabetypen oder Unstimmigkeiten in der Parameterzahl entdecken. Eine Identifizierung von Methoden, die am Modell verwendet, aber erst an der Instanz vorhanden sind, führen typischerweise zu einer Fehlermeldung *„Method maybe undefined“*.

Beispiel

Das Beispiel zeigt eine Modellkonstellation mit Methodenaufrufen und ihren Fehlermeldungen.

```
dialog D
variable list Data := [];

model window MWiDialog {
  rule integer Query() {
    !!  WARN: Method maybe undefined
    this:Fill("", -1, 0);
    return 0;
  }
}

MWiDialog WiDialog {
  rule boolean Fill(string Data, integer Id, integer Idx:=-1) { return
false; }

  on select {
    variable boolean B;
    !! WARN: value mismatch type - integer conflicts with expected boolean
    B := this:Query();
    if B then
      !! WARN: Invalid argument #1
      this:Fill(Data, 1);
    else
      !! WARN: Method maybe undefined
      MWiDialog:Fill("", -1, 0);
    endif
  }
}
```

Fehlermeldungen mit „maybe“

Folgende Fehlermeldungen deuten auf Kind-Objekte, Attribute oder Methoden hin, bei denen ein Zugriff während der Laufzeit möglicherweise ein anderes Ergebnis liefert, sodass der Fehler nicht zwangsläufig auftritt. Diese Fehler werden z.B. bei dynamisch angelegten oder veränderten benutzerdefinierten Attributen oder dynamischen Vater-Kind-Beziehungen gemeldet.

Fehlercode	Fehlermeldung
DME_TypeMayClash	Value maybe mismatch type
DME_ScopeMayClash	Value maybe mismatch scope
DME_ArgMayInvalid_1 … DME_ArgMayInvalid_16	Argument #1 maybe invalid or mismatch scope … Argument #16 maybe invalid or mismatch scope
DME_ChildAttrMayUndef	Child or attribute maybe undefined
DME_MethodMayUndef	Method maybe undefined
DME_BuiltinMayUndef	Builtin maybe undefined
DME_FunctionMayUndef	Function maybe undefined

Der Schweregrad dieser Fehlermeldungen kann mit **-IDManalyseseverity** gesteuert werden.

Neu: Die Konfigurationsmöglichkeiten der erweiterten Fehleranalyse wurden erweitert.

Die erweiterte Fehleranalyse wertet bei `for`-Schleifen den Variablenzugriff auf die Laufvariable als „write“ und nicht als „read“, da der IDM auch intern keinen Lesezugriff für die Bedingungsauswertung braucht. Dies begünstigt einerseits die Erkennung von ungenutzten Schleifenvariablen, führt andererseits aber auch zu unnötigen Warnungen.

Die Fehleranalyse kann nun so konfiguriert werden, dass für die Analyse von Schleifenvariablen ein „read“-Zugriff angenommen wird. Dadurch verschwinden *„Variable is never read“*-Warnungen, die sich auf `for`-Schleifenvariablen beziehen.

Die Konfiguration erfolgt über die Option `-IDManalyzerconfig=<configstr>` oder die Umgebungsvariable IDM_ANALYZERCONFIG, wobei die Option Vorrang hat.

Für *<configstr>* sind aktuell folgende Tokens erlaubt:

loop_var_read

Bei `for`-Schleifenvariablen wird immer ein „read“-Zugriff auf die Schleifenvariable angenommen, auch wenn im Schleifenrumpf keiner erfolgt.

Für zukünftige Erweiterungen ist vorgesehen, mehrere Tokens Komma-separiert anzugeben.

Die Konfiguration wird auch im Builder-Modus an den Builder-Prozess weitergereicht.

Neu: Mit der Option `-IDMdisable_errors=<namepart>` können mehrere Fehlermeldungen gleichzeitig deaktiviert werden. Durch die Angabe *<namepart>* werden alle Fehlermeldungen deaktiviert, deren Fehlercode *<namepart>* enthält.

IDM-13076: Bei der erweiterten Fehleranalyse wurden mehrere Fehler behoben:

- Aufrufe der **:delete()**-Methode ohne Parameter an einem ***doccursor*** erzeugen keine Fehlermeldung *„Wrong number of arguments“* mehr.
- In `case`-Anweisungen wird der Variablenzugriff nun auch vom letzten Knoten korrekt zum `case`-Knoten weitergereicht.
- Für Aufrufe von Toplevel-Regeln und -Funktionen wird von der Fehleranalyse kein *„Function may be undefined“* mehr gemeldet.
- Beim Schreiben und Lesen der Interface-Dateien werden optionale Funktionsparameter jetzt korrekt berücksichtigt.
- Für **:call()** und **:recall()** werden nun die Methodeneigenschaften berücksichtigt, sodass es nicht mehr zu unberechtigten Meldungen *„Method maybe undefined“* kommt. Außerdem wurden in Fehlermeldungen die Methodennamen ergänzt.

IDM-13067: Die erweiterte Fehleranalyse von Regeln wertet jetzt den Indexausdruck beim indizierten Zugriff auf lokale und globale Variablen sowie Parameter korrekt aus. Ein Lesezugriff in einem Indexausdruck, der auf lokale Variablen zugreift, wird nun auch als „read“-Zugriff gewertet. Dadurch kommt es nicht mehr zu fehlerhaften Meldungen der Art *„Variable is never read“*.

Behoben: Für die Variablenbehandlung und die Erkennung von „totem“ Code in Schleifen wurde die Fehleranalyse auf zwei Phasen umgestellt. Variablen, die in Schleifen (`while`, `for`, `foreach`) genutzt werden, erhalten für die Variablenanalyse innerhalb der Schleife den Wert „undefined“. Bei Laufvariablen in `for`-Konstrukten werden bei einer toten Schleife die Variablenwerte unverändert zum nächsten Knoten im Kontrollfluss „weitergereicht“.

Behoben: Benutzerdefinierte Attribute mit Indexgröße *0* werden nicht mehr als Skalar behandelt, sodass ein mit *integer* indizierter Zugriff nicht mehr als Fehler gewertet wird. Aufgrund der aktuellen Implementierung gilt dies nur für nicht geerbte Attribute.

Behoben: Die Reinitialisierung der Kontrollflussanalyse wurde korrigiert, sodass ihre Konfiguration für jedes Build-Kommando korrekt übernommen wird.

Behoben: Eine Zugriffsverletzung, die bei Funktionsaufrufen mit optionalen Parametern und *null*-IDs unter der 64-Bit-Version von WINDOWS auftreten konnte, wird nun vermieden.

13 A.06.02.k

13.1 Kern

IDM-13072: Das „Sharing“ von importierten Modulen, welches mit der Umgebungsvariablen DM_SHARED_MODULES aktiviert werden kann und die Notwendigkeit desselben Imports im übergeordneten Modul bzw. Dialog (Master-Import) aufhebt, ist eingeschränkt um die Konsistenz hinsichtlich Sichtbarkeit von Objekten, Dialogzugehörigkeit sowie *start*, *finish*, **:init()** und **:clean()** zu gewährleisten. Unter „Sharing“ wird verstanden, dass der IDM ein mehrfach verwendetes Modul nur einmal lädt.

Das „Sharing“ von Modulen ist nur im gleichen Dialog möglich!

Das Aktivieren des „Sharing“ wird nun auch in der Trace-Datei mit einem *IM*-Tracecode signalisiert.

Außerdem wurden Tracing und Dumpstate-Ausgabe beim Starten und Stoppen von Modulen sowie beim Erzeugen und Zerstören von Objekten verbessert um Fehler, die aus problematischen Importstrukturen entstehen, leichter zu erkennen.

Achtung

Das „Sharing“ von Modulen mit DM_SHARED_MODULES ist keine dokumentierte Standardfunktionalität des IDM. Sinnvoller ist der Import von Modulen mit „use“ um das mehrfache Laden von Modulen in komplexen Dialogen sicher zu unterbinden (siehe Kapitel „Import mit use“ im Handbuch „Programmiertechniken“).

IDM-13066: Von mehreren Dialogen gleichzeitig verwendete Module werden nun erst gestoppt (mit oder ohne Zerstören), wenn der Dialog mit dem Master-Import gestoppt wird. Dies entspricht dem Verhalten der IDM-Version A.06.01.e.

Achtung

Die Verwendung von „Shared Modules“ über Dialoggrenzen hinweg ist **ausdrücklich nicht** empfohlen, da es sich um keine dokumentierte Funktionalität des IDM handelt. Es führt zu Inkonsistenzen beim Stoppen (*finsh*) der mehrfach verwendeten Module, d.h. Module werden nicht vor den importierenden Modulen gestoppt. Außerdem ist die Zuordnung von Objekten zu einem Dialog damit nicht gewährleistet!

Für das dynamische Entladen von Modulen sollte entweder *Import.load := false* verwendet werden oder Module sollten per **use** angebunden und mit :unuse() entfernt werden.

IDM-13065 (keine Änderung): Das inkorrekte Entladen von Modulen beim Stoppen eines Dialogs, das in früheren IDM-Versionen (bis einschließlich A.06.02.f) in bestimmten Konstellationen mit „Shared Modules“ passierte, tritt nicht mehr auf.

IDM-13064: Beim Zerstören von Modulen werden nun auch Referenzen entfernt, die durch Modell-Gültigkeitsbereiche bei Argumenten und lokalen Variablen von Regeln in anderen Modulen und Dialogen entstehen.

Die Fehlersituation beim Zerstören von Modulen entstand durch eine inadäquate Nutzung von „Shared Modules“, die eigentlich vermieden werden sollte, da sie die Ablauf-Konsistenz verletzt.

Neu: Die erweiterte Fehleranalyse des Regelcodes kann nun unbenutzten, „toten“ Code erkennen. Diese Fehler werden mit dem Fehlercode „DME_UnreachableStmts“ (*„Statements are unreachable“*) und der entsprechenden Fehlerstelle ausgegeben.

In folgenden Situationen kann „toter“ Code erkannt werden:

1. Wenn im Regelcode eine `return`-, `pass`- oder `throw`-Anweisung steht und im Kontrollfluss dahinter noch Anweisungen folgen, werden diese als *„Statements are unreachable“* gemeldet.
2. Wenn eine Auswertung von arithmetischen und logischen Ausdrücken ergibt, dass Regelcode nicht erreichbar ist. Dies ist nur für Ausdrücke möglich, die nur aus konstanten Werten bestehen oder aus skalaren Werten, die sich direkt und zweifelsfrei aus lokalen Variablen oder Parametern ableiten lassen.
   Diese können dabei auch aus einem auswertbaren, vorangegangenen Ausdruck entstanden sein, solange es sich um einen eindeutigen und einmaligen Schritt in der Regelabarbeitung (Kontrollfluss) handelt.

Beispiele

```
rule void R1() {
  print "Hello";
  return;
  print "World";         // => DEAD CODE (statements are unreachable)
}

rule void R2() {
  if 5 > 0 then
    print "OK";
  else
    print "Too small!";  // => DEAD CODE (statements are unreachable)
  endif
}

rule void R3() {
  variable integer I := 10;
  I := I - 6;
  if I > 5 then
    print "I > 5";       // => DEAD CODE (statements are unreachable)
  else
    print "I <= 5";
  endif
}

variable integer DX := 6;
...
rule void R4() {
```

```
  variable integer I := 10;
  I := I - DX;           // => value of DX might be changed outside
                         //    so value of I becomes uncertain!
  if I > 5 then
    print "I > 5";       // => UNDETECTABLE: DEAD CODE
  else
    print "I <= 5";
endif
}
```

Grundsätzlich wird eine „fehlerfreie“ Ausführung angenommen. Es ist zu beachten, dass weiterhin eine statische Analyse der Verletzung von Gültigkeitsbereichen stattfindet.

Es können nur ASCII-Module und -Dialoge analysiert werden, keine Binärdateien!

Hinweise

» *if-elseif-else-endif*
Die Verwendung von *elseif…endif* ist eine vereinfachte Schreibweise für die Schachtelung von *if-else if…endif endif*. Ein Hinweis auf „toten“ Code in einem *elseif* bezieht sich immer auf den Code bis zum zugehörigen *endif*.

» *Schleifen for, foreach, while*
Innerhalb von *for*-, *foreach*- und *while*-Schleifen werden aktuell keine Variablenwerte für die Auswertung herangezogen. Deshalb kann innerhalb von Schleifenrümpfen „toter“ Code nicht erkannt werden, der aufgrund von Variablenwerten unerreichbar ist.

» *Behandlung von Variablenwerten*

1. Für die Auswertung konstanter Ausdrücke wird entweder ein konstanter Wert angenommen oder ein „undefinierter“ Wert, wenn sich Änderungen im Kontrollfluss ergeben.
2. Bei *for*-Schleifen, die „tot“ sind, wird der Initialisierungswert für die Laufvariable angenommen. Ansonsten wird die Laufvariable als „undefiniert“ behandelt.
3. Bei Schleifen werden alle im Schleifenblock modifizierten Variablen als „undefiniert“ angenommen, da beim mehrmaligen Durchlaufen keine sichere Annahme über ihren Wert getroffen werden kann.

Anmerkung

Ab dieser IDM-Version erfolgt die erweiterte Fehleranalyse standardmäßig nur noch bei Build-Aktionen wie **-writeexport**, **-writebin**… und nicht mehr beim Laden oder Simulieren von ASCII-Dialogen mit der IDM-Simulation oder beim Laden einer selbst gebauten IDM-Anwendung.

Neu: Die erweiterte Fehleranalyse ist nun konfigurierbar.

Folgende Optionen sind für die Steuerung der erweiterten Fehleranalyse hinzugekommen. Diese Optionen wirken nur in der Entwicklungsversion des IDM, da nur die Entwicklungsversion ASCII-Dateien laden kann.

-IDMerroranalysis <boolean>

Durch Angabe von *true* wird die erweiterte Fehleranalyse aktiviert, durch *false* wird sie deaktiviert.

Standardmäßig ist die erweiterte Fehleranalyse eingeschaltet, wenn eine Build-Aktion ausgeführt wird und ausgeschaltet, wenn ein Dialog simuliert oder ein ASCII-Dialog in einer selbstgebauten Anwendung geladen wird.

-IDManalyseseverity <severity-char>

Diese Option bestimmt den Schweregrad, mit dem Analysefehler gemeldet werden. Folgende Angaben sind möglich:

W
: Ausgabe als Warnung.

E
: Ausgabe als Fehler (und somit ins Error-Log).

F
: Abbruch nach dem ersten Fehler als „Fatal Error“ (führt zum Anwendungsabbruch).

0 (Null)
: Keine Ausgabe.

13.2 COBOL

Behoben: Die im Beispiel „adr“ angezeigten Meldungen sind jetzt nicht mehr leer. Ursache des Problems war der von **-writetrampolin** erzeugte Code, in dem *DM-Index* gesetzt aber nicht auf *0* zurückgesetzt wurde.

Behoben: Im Beispiel „tabcont“ wird initial nicht mehr ein leeres ***tablefield***-Objekt angezeigt. Das Problem entstand dadurch, dass die Objekt-ID noch in einem Datenbereich mit dem Format *PIC 9(4)* gespeichert wurde.

Behoben: Es werden nun „Structure Packing“ auf *1* gesetzt und FILLER verwendet, um zu vermeiden, dass der C-Compiler Felder an 8-Byte.Grenzen ausrichtet und die C-Struktur dann nicht mehr mit dem Speicherlayout des COBOL-Datenbereichs übereinstimmt. Das Problem wurde bei **DMcob_CallFunction()** auf 64-Bit-Architekturen bemerkt.

Behoben: Das Skript **idmcob** wurde angepasst sodass neuere Optionen für den Linker nun explizit weitergegeben werden.

Behoben: Die BYTE_ORDER wird jetzt in der Datei **IDMcobol.h** richtig erkannt, sodass die COBOL-Anbindung nun auch auf Intel-Architekturen beispielsweise mit RHEL 7.6 (64-Bit) funktioniert.

Behoben: Die Distribution enthält nun ein **Makefile.mfvis** für den Bau der COBOL-Demos mit MICRO FOCUS VISUAL COBOL. Außerdem wurden obsolete Dateien (z.B. **Makefile.vs**) entfernt und kleinere Korrekturen an den Beispieldialogen durchgeführt (z.B. ***tablefield***-Header sichtbar gemacht, *.borderwidth*-Warnungen entfernt).

Neu: Zu den COBOL-Demos gibt es nun eine **README**-Datei.

13.3 Dokumentation

IDM-13045: Für die Methoden **:clear()**, **:exchange()**, **:find()**, **:insert()** und **:move()** wurden neue Tooltips im „IDM Help Plugin“ für ECLIPSE erstellt.

IDM-12946: Bei der Methode :setinherit() ist nun der Rückgabewert beschrieben und in den Code-Abschnitten wurden `return`-Anweisungen mit *boolean*-Werten ergänzt.

IDM-12932: Die Methode :move() ist nun in der „Methodenreferenz“ enthalten.

IDM-12787: Die Beschreibungen von :clear(), :exchange(), :insert() und :move() in der „Methodenreferenz“ wurden grundlegend überarbeitet und entsprechend **:delete()** in die Varianten für Listenobjekte und Felder unterteilt.

Die Beschreibung von **:delete()** wurde leicht angepasst um mit den genannten Methoden konsistent zu sein.

Die neuen Methodenbeschreibungen für Felder werden nun auch im Kapitel „Methoden für Arrays benutzerdefinierter Attribute“ des Handbuchs „Benutzerdefinierte Attribute und Methoden“ verwendet.

IDM-12176: Bei der Methode :insert() ist nun beschrieben, dass neue Einträge am Ende angehängt werden, wenn der Parameter *Position = 0* ist.

IDM-12083: Die Beschreibung des .direction-Attributs für das ***tablefield*** in der „Attributreferenz“ ist nun ausführlicher. Dieselbe Beschreibung wurde auch beim tablefield in der „Objektreferenz“ eingebunden. Die Beschreibung von .direction in der „Attributreferenz“ wurde auch für die anderen Objekte überarbeitet.

IDM-9690: Übersetzungen der Methoden **:clean()**, **:destroy()**, **:get()**, **:recall()**, **:set()**, **:setclip()** und **:setinherit()** wurden in der englischen „Method Reference“ ergänzt. Damit ist die englische „Method Reference“ nun auf dem Stand der deutschen „Methodenreferenz“.

Behoben: In mehreren Attribut-Tooltips des „IDM Help Plugins“ für ECLIPSE wurden die Wertebereiche und/oder Indexbereiche ergänzt. Außerdem wurde die Reihenfolge vereinheitlicht und ist nun bei allen Attribut-Tooltips Kurzbeschreibung (Synopsis), Definition, Wertebereich (falls vorhanden), Indexbereich (falls vorhanden), zusätzliche Beschreibung (falls vorhanden).

Behoben: Der falsche Parameter *DoIt* bei der :clean()-Methode wurde entfernt. Die **:clean()**-Methode hat keine Parameter.

14 A.06.02.j3

14.1 Kern

IDM-13066: Von mehreren Dialogen gleichzeitig verwendete Module werden nun erst gestoppt (mit oder ohne Zerstören), wenn der Dialog mit dem Master-Import gestoppt wird. Dies entspricht dem Verhalten der IDM-Version A.06.01.e.

Achtung

Die Verwendung von „Shared Modules“ über Dialoggrenzen hinweg ist **ausdrücklich nicht** empfohlen, da es sich um keine dokumentierte Funktionalität des IDM handelt. Es führt zu Inkonsistenzen beim Stoppen (*finsh*) der mehrfach verwendeten Module, d.h. Module werden nicht vor den importierenden Modulen gestoppt. Außerdem ist die Zuordnung von Objekten zu einem Dialog damit nicht gewährleistet!

Für das dynamische Entladen von Modulen sollte entweder *Import.load := false* verwendet werden oder Module sollten per **use** angebunden und mit :unuse() entfernt werden.

IDM-13065 (keine Änderung): Das inkorrekte Entladen von Modulen beim Stoppen eines Dialogs, das in früheren IDM-Versionen (bis einschließlich A.06.02.f) in bestimmten Konstellationen mit „Shared Modules“ passierte, tritt nicht mehr auf.

IDM-13064: Beim Zerstören von Modulen werden nun auch Referenzen entfernt, die durch Modell-Gültigkeitsbereiche bei Argumenten und lokalen Variablen von Regeln in anderen Modulen und Dialogen entstehen.

Die Fehlersituation beim Zerstören von Modulen entstand durch eine inadäquate Nutzung von „Shared Modules“, die eigentlich vermieden werden sollte, da sie die Ablauf-Konsistenz verletzt.

15 A.06.02.j2

15.1 COBOL

Behoben: Die im Beispiel „adr“ angezeigten Meldungen sind jetzt nicht mehr leer. Ursache des Problems war der von **-writetrampolin** erzeugte Code, in dem *DM-Index* gesetzt aber nicht auf *0* zurückgesetzt wurde.

Behoben: Im Beispiel „tabcont“ wird initial nicht mehr ein leeres ***tablefield***-Objekt angezeigt. Das Problem entstand dadurch, dass die Objekt-ID noch in einem Datenbereich mit dem Format *PIC 9(4)* gespeichert wurde.

Behoben: Es werden nun „Structure Packing“ auf *1* gesetzt und FILLER verwendet, um zu vermeiden, dass der C-Compiler Felder an 8-Byte.Grenzen ausrichtet und die C-Struktur dann nicht mehr mit dem Speicherlayout des COBOL-Datenbereichs übereinstimmt. Das Problem wurde bei **DMcob_CallFunction()** auf 64-Bit-Architekturen bemerkt.

Behoben: Das Skript **idmcob** wurde angepasst sodass neuere Optionen für den Linker nun explizit weitergegeben werden.

Behoben: Die BYTE_ORDER wird jetzt in der Datei **IDMcobol.h** richtig erkannt, sodass die COBOL-Anbindung nun auch auf Intel-Architekturen beispielsweise mit RHEL 7.6 (64-Bit) funktioniert.

Behoben: Die Distribution enthält nun ein **Makefile.mfvis** für den Bau der COBOL-Demos mit MICRO FOCUS VISUAL COBOL. Außerdem wurden obsolete Dateien (z.B. **Makefile.vs**) entfernt und kleinere Korrekturen an den Beispieldialogen durchgeführt (z.B. ***tablefield***-Header sichtbar gemacht, *.borderwidth*-Warnungen entfernt).

Neu: Zu den COBOL-Demos gibt es nun eine **README**-Datei.

16 A.06.02.j

16.1 Windows

IDM-13038: Das ***tablefield*** und andere, vom IDM selbst gezeichnete Objekte, zeichnen nicht mehr in den Bereich der Scrollbars.

16.2 Kern

IDM-13058: Das Hinzufügen einer lokalen Variablen oder eines Parameters zu einer Sammlung wird von der Fehleranalyse jetzt korrekt als Lesezugriff erkannt. Dadurch kommen für diese Variablen und Parameter keine Warnungen mehr, dass sie nie verwendet werden.

16.3 Dokumentation

Neu: Das IDM HELP PLUGIN enthält nun spezielle Inhalte für die **Tooltips** von Schlüsselwörtern und *enum*-Werten.

17 A.06.02.i

17.1 Unix

Neu: Die Plattform RHEL 8.1 (32- & 64-Bit) mit dem Compiler GCC 8.3.1 und den Toolkits Motif 2.3.4 und Qt 5.11 wird unterstützt.

Mit RHEL 8 und CentOS 8 werden standardmäßig keine **rsh-server**-Pakete mehr ausgeliefert. Beim DISTRIBUTED DIALOG MANAGER (DDM) ist daher die Verwendung von **SSH** für das Starten eines DDM-Servers notwendig. Unterstützung von SSH und SSL im DDM wurde mit IDM-Version A.06.02.g eingeführt.

17.2 Windows

IDM-13030: Es kommt nun ein *help*-Ereignis beim Drücken von `F1` in einem aktivierten ***edittext***, der an ein ***tablefield*** angebunden ist.

Behoben: Der „WSI-Runstate“ wird nun korrekt zurückgesetzt wenn **queryhelp()** mit einer ungültigen Objekt-ID aufgerufen wird.

Neu: Beim ISA DIALOG MANAGER für den Compiler VISUAL C++ 15.9 (VISUAL STUDIO 2017) ist nun an mehreren Stellen deutlich erkennbar, dass diese IDM-Version für WINDOWS 10 gebaut ist. Dies geht nun beispielsweise aus Verzeichnisname, Name der Installationsdatei, Benutzeroberfläche des Installationsprogramms und der Ausgabe von **-IDMversion** hervor.

Hinweis

Die Version der Windows-Schnittstelle WSIWIN ist *0x603*, da der IDM für WINDOWS 10 kompatibel zu WINDOWS 8.1 ist.

17.3 Motif

IDM-13031: Es kommt nun ein *help*-Ereignis beim Drücken von `F1` in einem aktivierten ***edittext***, der an ein ***tablefield*** angebunden ist.

17.4 Qt

IDM-13032: Es kommt nun ein *help*-Ereignis beim Drücken von `Shift` + `F1` in einem aktivierten ***edittext***, der an ein ***tablefield*** angebunden ist.

17.5 Kern

IDM-13034: Dynamisierte Anbindungen und Aufrufe von **C-Funktionen** mit ***record***-Parametern funktionieren nun korrekt. Das fehlerhafte Ausführen der Simulationsregel unterbleibt nun.

Änderung IDM-13053: Im **Builder-Prozess-Modus** gibt es nun ein Tracing um die Nutzung des Builder-Prozesses und die Aktionen in Client und Server nachvollziehbarer zu machen. Außerdem wurde die Funktionsweise der Startoptionen **+builder** bzw. **-builder** überarbeitet.

Der Modus und die Arbeit des Builders können anhand folgender Tracecodes nachvollzogen werden:

[BM]
Builder message
[BX]
Builder command
[BA]
Builder action
[BD]
Builder return

Bei **+/-builder** erfolgt die Erkennung von Server- und Client-Modus nun anhand der Aktionen (**-writebin**, **-writeexport**…) und nicht mehr anhand der Builder-ID um eine Aktion auch bei **-builder** wie erwartet als Client auszuführen. **+builder** aktiviert explizit den Client-Modus. Bei **-builder** wird automatisch vom Client-Modus ausgegangen, wenn es mit einer Aktion verwendet wird. Ohne Aktion wird im Builder-Prozess (Server) gearbeitet.

Hinweis für UNIX/LINUX

Falls ein zu kleiner **Builder-Timeout** gewählt wird, kann dies dazu führen, dass überhaupt keine Kommunikation zwischen Client und Server zustande kommt. Die dadurch entstandene und vom System nicht aufgeräumte **pipe**-Datei (meist unter */tmp/pipe_idmbuilder…* zu finden) muss dann unter Umständen von Hand gelöscht werden.

17.6 Debugger

IDM-13041: Es ist wieder möglich, **Breakpoints** auf die letzte Zeile mit der schließenden Klammer von Regeln und Methoden zu setzen.

IDM-13040: Die Kontextmenü-Befehle *„Alles anzeigen“* und *„Oberste Ebene anzeigen“* des *„Regel-Browsers“* werden nun auch bei sehr vielen Regeln und Modulen deutlich schneller ausgeführt, da die Baumansicht dafür kurz unsichtbar gemacht wird.

IDM-13029: Die **Regelliste** im Debugger sollte nun auch bei komplexen modularisierten Dialogen für alle geladenen Module vollständig sein.

17.7 Dokumentation

Neu: Das IDM HELP PLUGIN enthält nun spezielle Inhalte für die **Tooltips** von Objekten, Ressourcen, vordefinierten Attributen und Methoden, eingebauten Funktionen, Datentypen und Ereignissen.

Hinweis

Die Bereitstellung der Tooltip-Inhalte wirkt sich an vielen Stellen auch auf Struktur und Formatierung in den anderen Dokumentationsformaten (HTML, PDF, Eclipse-Hilfesystem) aus. Die augenscheinlichsten Auswirkungen sind:

- Die Objektkapitel in der „Objektreferenz“ und die Attributkapitel in der „Attributreferenz“ beginnen nun mit einer kurzen Beschreibung des Objekts bzw. Attributs gefolgt von der Definition.
- Die Attributdefinitionen in der „Attributreferenz“ wurden überarbeitet. In der HTML-Dokumentation und im IDM HELP PLUGIN werden zuerst die Informationen zu Datentyp, Zugriff und Auslösung eines *changed*-Ereignisses bei Änderung angezeigt. Zusätzliche Informationen wie Bezeichner und Datentyp in den Programmiersprachen C und COBOL sowie die Klassifizierung eines Attributs können ein- und ausgeblendet werden, indem man auf *„Mehr…“* klickt.
- In der „Ressourcenreferenz“ wurden die separaten varianten Definitionen in die Ressourcendefinitionen integriert. Wenn es verschiedene Definitionsmöglichkeiten gibt (zum Beispiel als Bitmap, Datei oder Grafikressource bei der ***tile***-Ressource) wurden diese ebenfalls in der Ressourcendefinition zusammengefasst.

Neu: Bei der **Suche** in der HTML-Dokumentation wird nun eine **Vorschau** des besten Treffers über der Ergebnisliste angezeigt, wenn nach Objekten, Ressourcen, vordefinierten Attributen und Methoden, eingebauten Funktionen, Datentypen und Ereignissen gesucht wird.

Bei mehrdeutigen Suchbegriffen erhält man in der Regel das passende Vorschauergebnis, wenn man den Typ des gesuchten Elements in den Suchbegriff aufnimmt, zum Beispiel „objekt window“ oder „attribut window“, je nachdem ob man Informationen zum Objekt oder zum Attribut „window“ sucht.

18 A.06.02.h

18.1 Kompatibilität

Das Binärformat hat sich geändert. Die Binärdateien müssen neu generiert werden.

Das Netzwerkprotokoll hat sich geändert, die alten Versionen sind mit dieser Version nicht mehr kompatibel.

Die Trampolin-Module für COBOL müssen neu generiert werden.

18.2 Windows

IDM-12931: Beim Linken einer IDM-Anwendung konnte es zu vielen Warnungen wegen fehlender PDB-Dateien kommen. Dies sollte nicht mehr auftreten, da jetzt alle Projekte in der Release-Konfiguration ohne Debug-Information gebaut werden.

Behoben: Die Distributionspakete enthalten nun wieder die korrekte, aktuelle Version der Anwendung **serial.exe**. Bei einzelnen IDM-Versionen führte eine veraltete **serial.exe**, dazu, dass die Installation wegen eines vermeintlich ungültigen Lizenzschlüssels fehlschlug.

Neu: Die IDM-Dokumentation wird nun auch als „Eclipse Help Plugin“ ausgeliefert um sie mit dem Eclipse-Hilfesystem nutzen zu können.

Im Vergleich zur HTML-Dokumentation des IDM enthält das IDM HELP PLUGIN keine PDF-Dokumente, außerdem sind die „Installationsanleitung“ und die Handbücher zu den IDM-Werkzeugen (Editor, Debugger, Profiler, Tracefile Analyzer) im IDM HELP PLUGIN nicht enthalten.

Installation des IDM HELP PLUGINS

Die mit dem IDM ausgelieferte Datei **de.isa.idmdoc.eclipseplugin_<version>.jar** aus dem Verzeichnis *<IDM-Installationsverzeichnis>\doc\eclipse* in das Unterverzeichnis „dropins“ oder „plugins“ von Eclipse kopieren. Ein eventuell vorhandenes IDM HELP PLUGIN mit älterer Versionsnummer löschen.

Beim nächsten Start von Eclipse ist dann im Eclipse-Hilfesystem (Menübefehl *„Help“* → *„Help Contents“*) links unter *„Contents“* der Eintrag „ISA Dialog Manager Dokumentation“ vorhanden und kontextsensitive Informationen aus dem IDM HELP PLUGIN werden in der *„Help View“* als *„Related Topics“* sowie in Tooltips angezeigt.

18.3 Kern

IDM-13003: Das interne Zerstören von Kindobjekten eines ***Control***-Objekts erfolgt nun synchron mit dem Zerstören des ***Control***-Objekts. Es sollte daher nicht mehr zu „Assfails“ beim *finish*-Ereignis bzw. **destroy** eines ***Control***-Objekts mit Kindern kommen.

IDM-12998: Eine **:init**-Methode zur Objekt-Initialisierung wird nun auch ausgeführt, wenn innerhalb einer **:init**-Methode **:create()** rekursiv aufgerufen wird nachdem schon ein Aufruf von **:super()** erfolgte.

IDM-12989: Beim Einlesen der Interface-Datei von Modulen mit einem `export use` werden die *use*- bzw. *import*-Informationen nun fehlerfrei aufgebaut, sodass das per `export use` importierte Modul korrekt erkannt wird und keine Fehler *„cannot resolve label“* mehr auftreten.

IDM-12988: Mehrfache, gleiche `use`-Anweisungen in komplexen, modularisierten Dialogen führen nicht mehr zu einer Endlosschleife beim Starten und Beenden.

IDM-12987: Der Modellpfad eines geerbten, hierarchischen Kindes wird von **-compile** nun korrekt als kompletter, absoluter Pfad binär herausgeschrieben.

IDM-12962: In der Fehlermeldung *„Module file <> not found“* ist der Name des Use-Pfads nicht mehr fehlerhaft oder leer.

Behoben: Wegen einer Änderung am Use-Pfad konnte es vorkommen, dass trotz gesetztem **-buildersetdirty** Dateien vom **Builder-Prozess** nicht erneut geladen und Module deshalb nicht gebaut wurden. Dieses Problem wurde beseitigt.

Behoben: Das Builder-Protokoll wurde erweitert um die Option **-writeonchange** an den Builder-Prozess weiterzugeben. Zuvor wurde die Option nur beim Start des Builder-Prozess gesetzt.

Behoben: Das Builder-Protokoll wurde erweitert um den COBOL-Typ an den Builder-Prozess weiterzugeben. Zuvor wurden die Optionen für den COBOL-Typ nur beim Start des Builder-Prozess gesetzt.

Behoben: In der Sprachdefinitionsdatei **idmlangdef.xml** für das IDM ECLIPSE PLUGIN wurden fehlende Methoden, Attribute, ***Font***-Modifizierer und die Bezeichner für die Rasterfaktoren von ***Fonts*** ergänzt.

Änderung: Bei der Speicherverwaltung für den Parse-Baum wurde die Speichereinheit von *char* auf *long* umgestellt, sodass nun 4- oder 8-mal 64kB möglich sind und die Fehlermeldung *„rule exceeds internal size“* seltener auftauchen sollte.

Neu: Für die Kontexthilfe des IDM ECLIPSE PLUGINS wurden in der Sprachdefinitionsdatei **idmlangdef.xml** Namen für Symbole hinzugefügt.

18.3.1 Prüfung der Verwendung lokaler Variablen

Im Rahmen der Fehleranalyse erfolgt zusätzlich eine Prüfung auf nicht verwendete **lokale** Variablen oder auf Variablen, die vor der Verwendung nicht korrekt initialisiert bzw. gesetzt wurden.

Ziel der Prüfung ist, mögliche Fehler schon vor der Ausführung zu erkennen. Für die Prüfung wird von folgenden Annahmen ausgegangen:

- Grundsätzlich sollte auf eine Variable lesend und schreibend zugegriffen werden, sonst ist sie „nutzlos“.
- Erfolgt der lesende Zugriff vor dem schreibenden Zugriff ist von einem Fehler auszugehen.

Für die gefundenen Problemstellen werden Warnungen der Form *„*** Warning in…“* mit Details zum Problem ausgegeben. Das Laden wird nicht abgebrochen.

Für die Prüfung von Variablen gilt folgendes:

- Es werden nur lokale Variablen und Parameter für die Prüfung herangezogen. Eine Prüfung von globalen Variablen findet nicht statt.
- Zielsetzung ist die Erkennung von ungenutzten, lokalen Variablen, die Verwendung nicht initialisierter Variablen und das fehlende Setzen von `output`-Parametern.
- `input`-Parameter bzw. `input output`-Parameter sind von sich aus „initialisiert“, ein nicht initialisierter Zugriff kann hier **nicht** festgestellt werden. Nur von reinen `input`-Parametern wird ein fehlender lesender Zugriff festgestellt, wenn sich nur schreibende Zugriffe in einer Regel befinden.
- Bei einem dynamischen Objekt-, Methoden- oder Regelbezug kann die Aufrufsignatur **nicht** bestimmt werden. Dies führt eventuell zu Warnungen, weil die `input`- bzw. `output`-Eigenschaft der Argumente nicht bestimmt werden kann. Für Parameter wird dann die `input`-Eigenschaft angenommen. Für Methodenaufrufe empfiehlt sich daher die Nutzung von *object*-Variablen mit Gültigkeitsbereich (Modellbezug) bzw. eine Initialisierung der lokalen Variablen.
- Bei Verzweigungen oder Konditionen ist **nicht** ermittelbar, welcher Zweig durchlaufen wird. Es erfolgt **keine** Wertepropagierung und Berechnung. Deshalb empfiehlt sich, Variablen zu initialisieren oder in jedem möglichen Ausführungszweig zu setzen.
- Bei Schleifen mit `for` und `foreach` erfolgt kein implizierter, gedachter lesender Zugriff auf die Laufvariable um eine fehlende Nutzung zu erkennen.
- Bei einem indizierten Zugriff kann **nicht** ermittelt werden ob der Index-Wert gesetzt bzw. initialisiert ist.
- Nebenwirkungen von Anweisungen werden bei der Variablen-Auswertung **nicht** berücksichtigt. Beispielsweise kann ein Methodenaufruf in einer Variablen-Initialisierung durchaus Sinn machen, die Variable ist unter Umständen trotzdem als „never used“ eingestuft.

Fehlertext Fehlername	Ursache Behebung
„Variable is not initialized“ *DME_VarNotSet*	Die lokale Variable ist bei der Verwendung nicht gesetzt bzw. initialisiert. Variable bei der Deklaration initialisieren oder in einer vorherigen Anweisung setzen.
„Variable is never used“ *DME_VarNotUsed*	Die lokale Variable ist zwar definiert und eventuell auch initialisiert. Es erfolgt aber kein weiterer Zugriff auf den Variablenwert. Variable entfernen oder benutzen (auslesen).
„Output argument never set“ *DME_OutArgNotSet*	Der `output`-Parameter wird in der Regel nicht gesetzt. Eine Zuweisung auf den `output`-Parameter einbauen.

Fehlertext Fehlername	Ursache Behebung
„Output argument maybe never set“ *DME_OutArgMayNotSet*	Bei einem verzweigten Kontrollfluss in der Regel wird der `output`-Parameter nicht in allen Zweigen gesetzt. Es kann also durchaus vorkommen, dass während des Regelablaufs ein Zweig ohne Wertzuweisung durchlaufen wird. `output`-Parameter in allen Zweigen setzen.
„Variable maybe used uninitialized“ *DME_VarMayUnset*	Bei einem verzweigten Kontrollfluss in der Regel wird die Variable nicht in allen Zweigen gesetzt. Es kann also durchaus vorkommen, dass während des Regelablaufs ein Zweig ohne Wertzuweisung durchlaufen wird. Variable initialisieren oder in allen Zweigen setzen.
„Variable is never read“ *DME_VarNotRead*	Eine lokale Variable wird zwar gesetzt aber niemals ausgelesen. Fehlenden Zugriff ergänzen.

Beispiel

Dieses Beispiel zeigt, welche Probleme bei der Verwendung von Variablen erkannt werden sowie die Grenzen der Erkennung.

18.3.2 Deaktivierung von Meldungen der Fehleranalyse

Die Meldung von Fehlern, welche von der erweiterten Fehlerprüfung erkannt werden, kann durch den Anwender gesteuert werden. Dazu dient die Startoption **-IDMdisable_error=<errorname>**, die mehrfach als Argument angegeben werden kann.

Dabei ist <*errorname*> ein DME_-Fehlername aus der Datei **IDMuser.h**, wobei „DME_“ zur Vereinfachung weggelassen werden kann.

Das Deaktivieren von Fehlern, welche die Gültigkeitsprüfung betreffen, ist nicht möglich.

Beispiel

Mit der Startoption `-IDMdisable_error=VarMayUnset` kann die Ausgabe der Fehlermeldung *„Variable maybe used uninitialized“* unterbunden werden.

18.4 Debugger

IDM-13001 und IDM-12997: Der *„Regel-Browser“* bleibt nun während des Ladens unsichtbar um die Ladezeiten zu verkürzen und sein Flackern beim Laden von Dialogen mit sehr vielen Modulen zu vermeiden. Ebenfalls um die Ladezeiten zu verkürzen wurde die Verwaltung der Regelliste optimiert und die Anzahl von Aufrufen zwischen IDM und Debugger beim Laden von Modulen verringert.

Außerdem wird die Benutzeroberfläche nun nach Anzeige einer Fehlermeldung wieder freigegeben.

IDM-12996: Beim erneuten Verbinden eines Dialogs mit dem separat gestarteten Debugger kommt es nicht mehr zu Abstürzen durch interne Inkonsistenzen beim Freigeben und Sperren der Benutzeroberfläche für den Benutzer. Diese Inkonsistenzen konnten den Eindruck erwecken, dass Benutzeraktionen möglich waren, obwohl der Debugger nicht gestoppt hatte. Ebenso sollte es zu keinen Abstürzen nach einem Stopp-Kommando mehr kommen.

Außerdem wird die Benutzeroberfläche nun nach Anzeige einer Fehlermeldung und dem Umsetzen von „Autostop“ oder „Errorstop“ wieder freigegeben.

IDM-12990: **Breakpoints** werden nun zuverlässig auf die gewünschte Zeile bzw. die nächste Zeile mit Break-Möglichkeit gesetzt. Zuvor konnte es zu Abstürzen beim Debuggen kommen, weil Breakpoints an unzulässige Stellen umgesetzt wurden.

IDM-12959: Im TRACEFILE ANALYZER wird nun die Scroll-Position beibehalten, wenn bei aktivem Filter eine Zeile selektiert und anschließend der Filter deaktiviert wird. Die Scroll-Position wird nicht mehr auf den Dateianfang zurückgesetzt, wodurch kein erneutes Suchen der selektierten Stelle notwendig ist.

IDM-12890: Index-Zahlenwerte mit drei Ziffern werden vom TRACEFILE ANALYZER nicht mehr fälschlicherweise als Tracetime identifiziert.

IDMDBG-92: Das Kommando „Next“ („Step Over“, `F6` im IDM DEBUGGER PLUGIN) springt nun wieder an die richtige Stelle, wenn sich Annotationen (`!!`-Kommentare) vor einem Regel- oder Methodenaufruf befinden.

Außerdem werden mehrzeilige Annotationen in der *„Code-Ansicht“* wieder vollständig angezeigt.

IDMDBG-91: **Breakpoints** in Zeilen nach einer `return`-Anweisung werden nicht mehr fälschlicherweise an das Regelende verschoben.

Behoben: Im TRACEFILE ANALYZER für MICROSOFT WINDOWS bleibt nun die gelbe Markierung der selektierten Zeile auch dann erhalten, wenn in dieser Zeile so weit nach rechts gescrollt wird, dass kein Zeichen mehr angezeigt wird.

Behoben: Die Anzeige des TRACEFILE ANALYZERS unter MICROSOFT WINDOWS wird nun aktualisiert, wenn das Fenster verbreitert wird und sich das erste, links anzuzeigende Zeichen ändert. Dadurch kommt es nicht mehr zu Folgefehlern, beispielsweise dass nicht mehr nach links gescrollt werden konnte.

Behoben: Im TRACEFILE ANALYZER für MICROSOFT WINDOWS wurden diverse Probleme beim Scrollen behoben und Diskrepanzen zwischen Anzeige und Scrollbar-Position beseitigt.

Neu: Im Debugger kann nun die aktuelle Anweisung übersprungen werden ohne sie auszuführen.

Die aktuelle Anweisung kann während des Debuggens wie folgt übersprungen werden:

» mit der Tastenkombination `Shift` + `F10`,

» mit dem Befehl *„Überspringe Anweisung“* (englisch *„Skip“*) im Menü *„Debug“*,

» mit dem Befehl *„Anweisung überspringen“* (englisch *„Skip“*) im Kontextmenü der *„Regelanzeige“*.

18.4.1 Teilweises Laden sehr großer Tracedateien im Tracefile Analyzer

Wenn mit dem TRACEFILE ANALYZER eine Tracedatei geöffnet wird, die zu groß ist um vollständig geladen zu werden, dann wird das Dialogfeld *„Ladekonfiguration“* angezeigt in dem das weitere Vorgehen bestimmt werden kann.

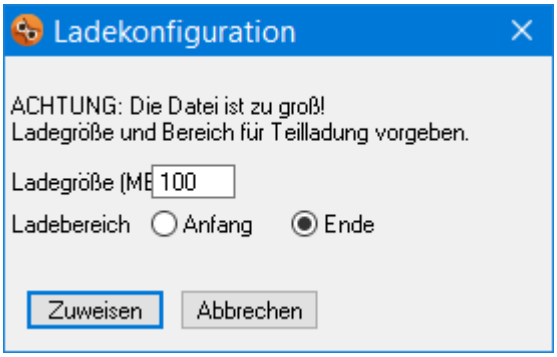


Abbildung 6: *Abfrage beim Laden einer sehr großen Trace-Datei*

Der Wert im Eingabefeld *„Ladegröße“* legt fest, wie viel von der Trace-Datei geladen werden soll.

Die Optionsschaltflächen *„Anfang“* und *„Ende“* bestimmen, welcher Teil der Trace-Datei geladen wird.

Mit *„Zuweisen“* wird angegebene Teil der Trace-Datei geladen.

„Abbrechen“ bricht das Laden der Trace-Datei ab.

Wird eine Datei nur teilweise geladen, dann werden Informationen zum geladenen Teilbereich in der Zeile *„Map Range“* des *„Infobereichs“* angezeigt.

Beim Laden des Dateiendes kann der geladene Teil mitten in einer Trace-Zeile beginnen und die Zeilennummerierung beginnt bei der ersten geladenen Zeile mit *1*.

Hinweis

Der TRACEFILE ANALYZER unterstützt auf 32- und 64-Bit-Systemen das Laden von Dateien, die größer als 2 GB sind. Bis zu welcher Größe eine Datei tatsächlich geladen werden kann, hängt allerdings von mehreren Faktoren ab, wie dem adressierbaren bzw. allokierbaren Speicher, der Zeilenzahl und anderen.

Es gelten folgende Obergrenzen:

» Zeilenzahl: 2 Milliarden.

» Dateigröße: 4 GB auf 32-Bit-Systemen, auf 64-Bit-Systemen sind auch größere Dateien möglich.

» Speicherbedarf: Obergrenze ist der adressierbare Arbeitsspeicher, wobei zu beachten ist, dass der TRACEFILE ANALYZER zusätzlichen Speicher für die Verwaltung von Trace-Struktur und

Zeileninformationen allokiert.
Die Abfrage zum teilweisen Laden erfolgt, wenn kein vollständiges „Mapping“ der Datei in den Arbeitsspeicher möglich ist oder der heuristisch ermittelte Speicherbedarf für Trace-Struktur und Zeileninformationen über 500 MB liegt.

18.5 DDM (Netz)

Behoben: **SSL** kann nun unter UNIX auch mit älteren OpenSSL-Versionen, in denen **SSL_CTX_set_options** nicht als Funktion sondern als Makro definiert ist, initialisiert werden. Das Makro wird verwendet, wenn es definiert ist, ansonsten wird die Funktion geladen.

Behoben: Unter WINDOWS funktioniert **SSL** nun auch im Modus **-IDMserve**. Die Option **-IDMtransport** wird jetzt korrekt in den Argumentvektor für den Unterprozess kopiert.

Behoben: Der Modus **-IDMserve** funktioniert nun auch wieder unter WINDOWS. Zuvor kam es beim Kopieren des Argumentvektors zum Abbruch, weil zusätzliche Argumente für das Netzwerkprotokoll hinzukamen, die beim Parsen zwar auf *NULL* gesetzt aber nicht aus dem Argumentvektor entfernt wurden.

18.5.1 Neue Attribute und Optionen für SSL und SSH

Für die Unterstützung von SSL und SSH wurden am ***Application***-Objekt die Attribute *.publickeyfile* und *.privatekeyfile* sowie die Optionen *opt_verify_peer*, *opt_cert_required* und *opt_no_ssl_v2* des Attributs *.options[enum]* und eine entsprechende Startoption ergänzt.

18.5.1.1 Attribut .privatekeyfile

(neu)

In diesem Attribut wird die Datei mit dem privaten Schlüssel definiert, der für das SSH-Protokoll verwendet wird.

	Bezeichner		Datentyp
Regelsprache	**.privatekeyfile**		string
C	AT_privatekeyfile		DT_string
COBOL	AT-privatekeyfile		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	application		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Standardmäßig wird die Datei **private.pem** aus dem Installationsverzeichnis der Anwendung verwendet.

18.5.1.2 Attribut .publickeyfile

(neu)

In diesem Attribut wird die Datei mit dem öffentlichen Schlüssel definiert, der für das SSH-Protokoll verwendet wird.

	Bezeichner		Datentyp
Regelsprache	**.publickeyfile**		string
C	AT_publickeyfile		DT_string
COBOL	AT-publickeyfile		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	application		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Standardmäßig wird die Datei **public.pem** aus dem Installationsverzeichnis der Anwendung verwendet.

18.5.1.3 Attribut .options[enum]

(erweitert)

	Bezeichner		Datentyp
Regelsprache	**.options[enum]**		boolean
C	AT_options		DT_boolean
COBOL	AT-options		DT-boolean
Index-Datentyp	enum		
Klassifizierung	Objektspezifisches Attribut		
Objekte	Application		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Das ***Application***-Objekt besitzt folgende zusätzlichen Optionen für eine SSL-Verbindung.

» *opt_verify_peer*
Gibt an, ob die andere Seite verifiziert werden soll.

» *opt_cert_required*
Gibt an, ob ein Zertifikat benötigt wird.

» *opt_no_ssl_v2*
Gibt an, dass die Version 2 des SSL-Protokolls nicht verwendet wird.

18.5.1.4 Startoption -IDMssl_options

(neu)

Diese Startoption dient zur Konfiguration der SSL-Verbindung auf der Anwendungsseite. Folgende Optionen können durch Komma, Semikolon oder Leerzeichen getrennt angegeben werden. Mindestens eine der Optionen muss angegeben werden.

» *opt_verify_peer*
Gibt an, ob die andere Seite verifiziert werden soll.

» *opt_cert_required*
Gibt an, ob ein Zertifikat benötigt wird.

» *opt_no_ssl_v2*
Gibt an, dass die Version 2 des SSL-Protokolls nicht verwendet wird.

Diese Startoption ist nur auf der Anwendungsseite erlaubt.

18.6 COBOL

IDM-13007: Das dynamische Anbinden von COBOL-Funktionen (**BindThruLoader-Funktionalität**) funktioniert wieder. Beim Einschalten dieser Funktionalität wird jetzt der Funktionstyp definiert, so dass die Funktionen wieder aufgerufen werden.

Bei den Funktionen **DM_BindCallbacks()** und **DM_BindFunctions()** wird im Trampolin-Modul die neue interne Option *DMF_FC_RecordStub* benutzt, um anzuzeigen, dass der angegebene Funktionszeiger eine Stub-Funktion in der Programmiersprache C referenziert.

Beim dynamischen Anbinden von **Record-Funktionen** (Option **+writeheader**) werden nun diverse „Fallstricke“ erkannt und eine Funktion nicht aufgerufen, deren Anbindung nicht zur Aufrufoption passt.

Behoben: Beim dynamischen Anbinden von **Record-Funktionen** (Option **+writeheader**) unter der 64-Bit-Version von WINDOWS kommt es nicht mehr zu Abstürzen weil der Aufrufstack falsch aufgebaut wurde.

Neu: Als Endung für COBOL-Quelldateien wurde prinzipiell „.cob“ verwendet. MICRO FOCUS VISUAL COBOL verwendet aber standardmäßig „.cbl“. Jetzt wird als Endung „.cbl“ verwendet, wenn einer der Optionen für Micro Focus COBOL (**-ufcob**, **-mfviscob** oder **-mfviscob-u**) angegeben wird.

Hinweis zur Startoption +writetrampolin

Die Dateiendung der generierten COBOL-Dateien ist jetzt „.cbl“, wenn eine der COBOL-Startoptionen **-ufcob**, **-mfviscob** oder **-mfviscob-u** angegeben wird. Wird keine dieser COBOL-Optionen angegeben, bleibt es bei der Dateiendung „.cob“.

19 A.06.02.g

19.1 Kompatibilität

Das Binärformat hat sich geändert. Die Binärdateien müssen neu generiert werden.

Das Format der Interface-Dateien hat sich geändert. Alle Interface Dateien sollten neu generiert werden.

Das Netzwerkprotokoll hat sich geändert, die alten Versionen sind mit dieser Version nicht mehr kompatibel (siehe Kapitel 19.20 „Unterstützung von SSH und SSL im DDM“).

19.2 Unix

Neu: Die Plattform RHEL 7.6 (32- & 64-Bit) mit dem Compiler GCC 4.8.5 und den Toolkits Motif 2.3.4 und Qt 5.9 wird unterstützt.

Änderung: Die Makefiles der **Beispiele** wurden so umgestellt, dass standardmäßig (*all*-Target) nur Ziele gebaut werden, die für die installierte IDM-Version möglich sind.

Außerdem sind Beispiele, die Anwendungen bauen, nun so gestaltet, dass sowohl mit der Entwicklungs- als auch der Runtime-Bibliothek des IDM gelinkt wird. Die Runtime-Anwendungen enden dann mit dem Suffix „run“.

Die folgenden, generische Bauziele sind möglich: *all*, *dvl*, *run*, *net*, *ndx*, *usw*, *clean*, *start*. Letzteres dient zum sequenziellen Start aller Beispiel im Verzeichnis nach einem erfolgreichen Bau.

Wenn ein Bauziel angegeben wird, das nicht zur Installation passt (z.B. *net* für eine Installation ohne DDM-Option), dann wird ein Fehlerhinweis auf die fehlende Option ausgegeben.

Siehe auch

README-Datei bei den Beispielen

19.3 Windows

IDM-12925: Bei einem Klick auf den untersten, nur teilweise sichtbaren, Eintrag in der Liste eines ***Poptexts*** mit *.style = listbox* wird jetzt nur noch ein *select*-Ereignis erzeugt.

IDM-12289: Beim interaktiven Vergrößern und Verkleinern von Zeilen und Spalten am ***Tablefield*** kommt es nicht mehr zu temporären Darstellungsfehlern, bei denen Bereiche unter der Verschiebelinie nicht aktualisiert wurden und dadurch Trennlinien verschwanden. Die Darstellungsfehler korrigierten sich nach einer Mausbewegung meist selbst.

Durch die Fehlerbehebung flackert das Verschieben jetzt auch weniger.

Hinweis

Theoretisch können ähnliche Darstellungsprobleme auch bei den Objekten ***Splitbox*** und ***Toolbar*** auftreten.

IDM-12284: Beim **Drag & Drop** innerhalb desselben ***Edittexts*** kommt es nicht mehr dazu, dass zwei Cursor gezeichnet werden (einer wo das Verschieben begonnen wurde und einer, der sich mit der Maus mitbewegt). Nun wird nur noch der Cursor gezeichnet, der sich mit der Maus mitbewegt und die aktuelle Einfügeposition anzeigt.

Außerdem wird nun das Textende richtig bestimmt, sodass der Cursor beim Verschieben ans Ende nicht mehr zu weit rechts positioniert wird.

Neu: MICROSOFT VISUAL C++ 2017 wird nun unterstützt.

Hinweis zu den Beispielen

Die Beispielprojekte für MICROSOFT VISUAL STUDIO 2017 verwenden die Standardeinstellung für das Windows SDK. Damit sie funktionieren muss die Komponente „Windows 8.1 SDK und UCRT SDK“ installiert sein. Wenn nur ein aktuelleres Windows SDK installiert ist, dann müssen die Projekte angepasst werden. Hierzu unter *„Projekt“* → *„Eigenschaften“* → *„Konfigurationseigenschaften“* → *„Allgemein“* → *„Windows SDK-Version“* die installierte Version auswählen.

19.4 Qt

Neu: Die Option *opt_sim_insensitive* des ***Image***-Objekts wird jetzt auch unter Qt unterstützt. Wenn diese Option aktiv und am Image kein *.picture[tile_insensitive]* gesetzt ist, dann wird das generierte Bild für den Modus „QIcon::Disabled“ aus dem „QIcon“ des ***Tiles*** verwendet.

Neu: Die ***Tile***-Ressource wurde erweitert, sodass sie Bilder aus einer Qt-Ressourcendatei nutzen kann. Wenn Qt-Ressourcendateien zu einer IDM-Anwendung gelinkt werden, dann können darin enthaltene Bildressourcen einer ***Tile***-Ressource zugewiesen werden.

Beispiel

```
tile Ti ":/ResourceIdentifier";
```

Siehe auch

„The Qt Resource System“ in der Qt-Dokumentation

Der IDM FÜR QT stellt nun (analog zum IDM FÜR WINDOWS) eine Qt-Ressource mit den IDM-Standardsymbolen für IDM-Anwendungen (entspricht dem Symbol der IDM-Simulation) und die Fenster-Titelleiste bereit.

Beim Hinzufügen eigener Ressourcendateien zu einer IDM-Anwendung sollte beachtet werden, dass die Ressourcenbezeichner „IDM_DefIcon“ und „IDM_AppIcon“ vom IDM für das Symbol in der Fenster-Titelleiste – und unter MICROSOFT WINDOWS zusätzlich für das Anwendungssymbol – reserviert sind. Durch Definition einer eigenen Ressource mit dem Namen „IDM_AppIcon“ kann

das Symbol für die eigene Anwendung selbst bestimmt werden. Ist kein „IDM_AppIcon“ vorhanden, so wird das vom IDM bereitgestellte „IDM_DefIcon“ verwendet.

Informationen zum korrekten Anbinden einer Ressource sollten der Entwicklerdokumentation des jeweiligen Systems entnommen werden.

19.5 Motif

Neu: Das Attribut *.top_most* des ***Fensters*** wird nun auch unter Motif unterstützt. Hinsichtlich Klicks auf den Fenstertitel und Erkennen von Überdeckungen durch andere Fenster gibt es die gleichen Restriktionen und Problemstellen wie bei Dialogboxen.

19.6 Kern

IDM-12964: Das Kommando **+writeheader** verfängt sich nicht mehr in einer Endlosschleife, wenn in einem Modul eine Annotation (!!-Kommentar) zu einer COBOL-Funktion mit Record-Parametern enthalten ist.

IDM-12960: Beim **Binärschreiben** von Modulen oder Dialogen mit mehr als 32.768 Labels kommt es nicht mehr zu Fehlern, die sich dann beim Lesen als *„Can't determine external attribute“* zeigten.

IDM-12957: Beim Anziehen von Modulen ohne Interfaces über `use` kommt es nicht mehr zum Fehler *„can't create exported object, create it without export“* obwohl noch freie „Object Slots“ vorhanden sind. Der Fehler wurde durch viele, nicht exportierte Objekte (z.B. ***Text***-Ressourcen) hervorgerufen.

IDM-12938: Bei der **Trampolin-Generierung** im IDM-Builder-Modus werden nun **Sub-Records** auch korrekt herausgeschrieben, wenn sie durch ein vorheriges **-writetrampolin** schon einmal herausgeschrieben wurden.

IDM-12935: Die Zeilenzuordnung von Fehlern und Warnungen bei mehreren aufeinanderfolgenden Anweisungen wird nun vor Erreichen der nächsten Anweisung ermittelt und ist nun korrekt(er). Unter anderem sollten damit keine falschen Zuordnungen von *„Wrong number of Arguments“* Warnungen bei mehreren aufeinanderfolgenden Regelaufrufen mehr auftreten.

IDM-12915: Es gibt keine fehlerhaften Warnungen über die falsche Anzahl von Argumenten mehr, wenn die Fehleranalyse bei einem Aufruf der **:find**-Methode keinen Werttyp für das erste (optionale) Argument ermitteln kann. Dies ist zum Beispiel bei einem Aufruf in der Form `Tf:find (Tf.content[2,2])` der Fall.

IDM-12881: Es kommt nicht mehr zu fehlerhaften *„Missing return“* Warnungen in Regeln, die dynamisch ein Modul nachladen.

IDM-12843: RecordInfo-Strukturen bei Funktionen mit ***record***-Parametern und gesetzter Codepage *CP_acp* (MICROSOFT WINDOWS) werden nun fehlerfrei initialisiert.

IDM-12831 und IDM-12828: Es gibt zwei neue Werte für das *task*-Element der **DM_ErrorInfo**-Struktur, die an einen Fehlerhandler übergeben wird:

» *EITK_analyzer*
Für Fehler, die im Rahmen der Fehleranalyse erkannt werden.

Siehe auch
Kapitel 19.11.4 „Aufruf von Fehlerhandlern“

» *EITK_undefined*
Für Fehler, die keiner Komponente direkt zugeordnet werden können.

Behoben: Der Zugriff auf die Objekt-ID in *thisevent.value* beim *paste*-Ereignis eines Drag & Drop mit *type_object* liefert nun *null* falls diese ID ungültig ist. Dadurch wird auch ein Absturz des Regelinterpreters vermieden.

Behoben: Eine Variablenzuweisung erhält nun den Wertetyp bei *vector*-Variablen mit einem Datentyp als Gültigkeitsbereich.

Behoben: Es kommt nicht mehr zu einem Absturz, wenn ein Modell zur Laufzeit als Kind eines unsichtbaren Objekts instanziiert wird. Beobachtet wurde das Problem unter WINDOWS beim Instanziieren eines ***Toolbar*** Objekts.

Behoben: Externe Attributnummern, die durch die Definition von Attributen in USW-Klassen entstehen, werden im Regelcode nun korrekt umgesetzt. Außerdem sollte es zu keiner fehlerhaften Zuordnung bei einer geänderten Ladereihenfolge der USW-Bibliotheken mehr kommen.

Neu: Für **Datenfunktionen** können nun Simulationsregeln definiert werden.

Neu: Beim *help*-Ereignis steht nun im *.value*-Attribut des ***thisevent***-Objekts die Hilfe-ID aus dem obersten Objekt, das ein gesetztes *.help*-Attribut aufweist.

Änderung: Enthält die Redefinition einer **:init**- oder **:clean**-Methode keinen **:super**-Aufruf, so wird dieser am Methodenende nachgeholt um die Initialisierung und Bereinigung von Modellen und Kindern sicherzustellen.

Neuerungen zur Unterstützung optimierter Bauprozesse

» Die **benutzerdefinierten Attribute und Methoden** der exportierten Objekte werden nun als Kommentare mit der `@define`-Syntax in die **Interface-Datei** eines Moduls geschrieben.

» Wenn die neue Option **-IDMwriteonchange** zusammen mit IDM-Optionen, die Dateien generieren (z.B. **-writebin**, **-writeexport**) angegeben wird, dann wird der neue Inhalt mit einer gegebenenfalls schon vorhandenen Datei verglichen und nur in die Datei geschrieben, wenn es Änderungen gibt. Dieses Verhalten kann auch durch Setzen der Umgebungsvariablen `IDM_WRITEONCHANGE` mit einem beliebigen Wert aktiviert werden.

» Im Builder-Prozess-Modus ist die Wiederverwendung von schon geladenen Dateien möglich. Bisher war nur eine Wiederverwendung importierter Module durch Nutzung von `DM_SHARED_MODULES` möglich. Die Entscheidung, ob eine Datei wiederverwendet werden kann, erfolgt anhand ihres Änderungsdatums. Geänderte Module und Dialoge werden nun als „dirty“ markiert um sie von einer Wiederverwendung auszuschließen.

» Mit der Option **-buildermodlimit <percent>** kann für den Builder-Prozess definiert werden, wie viel freie Kapazität für Module minimal noch vorhanden sein muss, bevor ein Neustart des Builder-Prozesses erzwungen wird.

» Die neue Option **-buildersetdirty** für den Builder-Prozess erzwingt das Neuladen abhängiger Quelldateien. Damit ist es beispielsweise möglich, frühzeitig eine Fehleranalyse anzustoßen, wenn sich eine Interface-Datei geändert hat, nicht aber die abhängigen Dialog- oder Moduldateien.
Die Option **-buildersetdirty** markiert die angegebene Datei als „dirty“, sodass sie vom Builder-Prozess neu geladen wird. Bei der Option ist die Schreibweise (auch des Dateipfads) zu beachten.

19.7 Editor

Neu: Das Instanziieren von Modellen ist nun auch über Modulgrenzen hinweg möglich. Außerdem wird nun wenn möglich ein Import über **use** durchgeführt.

Der *„Datei-Browser“* wechselt bei einer Änderung der Selektion ohne eine Objekt-Selektion durchzuführen. Dadurch ist das Instanziieren von Modellen in ein anderes Modul einfach möglich.

Wenn nötig wird dabei das Modell exportiert und das Modul mittels ***import*** oder **use** eingebunden. Dabei wird **use** zum Importieren nur verwendet, wenn das Modul keine ***import***-Anweisungen aufweist und über einen Use-Pfad zugänglich ist.

Über die Einstellung *„Import-Benutzung erzwingen“* auf der Seite *„Editierung“* des Konfigurationsdialogs kann die Verwendung von ***import*** erzwungen werden.

19.8 Debugger

IDMDBG-76: Der Debugger schließt beim Aufbau der Regelliste jetzt auch Regeln aus geladenen aber ungestarteten Modulen ein.

IDMDBG-50: Der Debugger hält nun nach „Continue“ oder „Resume“ an **Breakpoints in Modulen**, auch wenn die Pfadangaben beim Laden der Moduldateien und Debuggen im Quelltext nicht exakt übereinstimmen (z.B. wegen relativer Pfadteile, unterschiedlicher Verzeichnisseparatoren oder unterschiedlicher Groß- und Kleinschreibung).

IDMDBG-48: Bei aktiviertem **Autostop** hält der Debugger nun wieder in einem erneut nachgeladenen Modul an.

19.9 DDM (Netz)

IDM-12355: Die Netzwerkschnittstelle überträgt nun auch die Bits *inherit*, *changed* und *defvalue* sowie den *index*-Wert der **DM_Value**-Struktur.

Damit diese zusätzlichen Werte übertragen werden, müssen sowohl Darstellungs- als auch Anwendungsseite mindestens die Protokollversion *1.2* aufweisen.

Behoben: Die Anwendungsseite bricht nun die Verbindung bei einer unpassenden Major-Version des Netzwerkprotokolls ab. Bisher kam es zu einem Abbruch mit einer irreführenden Fehlermeldung, weil die Darstellungsseite sich beendete.

19.10 COBOL

Neu: Die Ausgabeoptionen des IDM wurden erweitert, sodass nun eigene Zielverzeichnisse für die generierten COBOL-Dateien definiert werden können.

Der IDM verfügt nun über folgende Startoptionen zur besseren Konfigurierbarkeit der COBOL-Ausgabedateien:

- **-cobname**
  Gibt den Basisnamen der erzeugten COBOL-Datei an. Diese Option überschreibt den Wert, der aus dem Namen berechnet wird, der bei **-writetrampolin** angegeben ist. Die Dateiendung dieser Datei ist „.cob“.
- **-cpyname**
  Gibt den Basisnamen der erzeugten COBOL-Copy-Strecke an. Diese Option überschreibt den Namen, der durch **-cobname** bzw. **-writetrampolin** vorgegeben ist. Die Dateiendung dieser Datei ist „.cpy“.
- **-cobbasename**
  Gibt den Basisnamen an, der in der COPY-Anweisung in der erzeugten COBOL-Datei verwendet wird. Diese Option überschreibt den Wert, der aus den Optionen **-cpyname**, **-cobname** bzw. **-writetrampolin** berechnet wird.

19.11 Fehleranalyse

19.11.1 Geänderte Ausgabe bei Fehlern und Warnungen in Regeln

Der IDM meldet Fehler in Regeln typischerweise mit zwei Meldungen:

- *„***ERROR IN EVAL“* für Fehler, die vom Regelinterpreter während der Ausführung gefunden werden.
- Meldungen der Art *„***EXPR ERROR“* weisen auf Gültigkeitsfehler hin, die bei der Regelanalyse während des Ladens gefunden werden.

Diese beide Arten von Fehlermeldungen beinhalten, wenn es sich um geladene ASCII-Dialoge handelt, außerdem den Regeltext mit markierter Fehlerstelle. Andernfalls enthalten sie den Hinweis, dass es sich um eine Binär-Regel handelt.

Statt der bisherigen Ausgabe *„***EXPR ERROR“* erfolgt nun die Ausgabe als *„*** Error in file …“* mit dem Fehlercode *„IDM-E-ExprError“* und den Fehlerdetails bzgl. der Gültigkeitsverletzung wie z.B. *„value mismatch scope“*.

Anstatt der Ausgabe des markierten Regeltextes erfolgt nun die Angabe der Regel, des Dateinamens in dem sich die Regel befindet und der Zeile in der die Warnung oder der Fehler gefunden wurden. Ebenso wird der Quelltext der fehlerhaften Zeile hinter „source:“ ausgegeben.

```
*** Error in file 'dialog.dlg', line 19: rule D.MyRule
    value mismatch scope
    source: variable object[MRec] := setup;
    IDM-E-ExprError: Expression error
```

Eine analoge Ausgabe erfolgt auch für Fehlermeldungen des Regelinterpreters. Nur stehen hier, wie bisher auch, die Fehlerdetails in der ersten Zeile hinter dem „:“.

```
***ERROR IN EVAL: Bad operation
    in file '/home/hugo/tmp/r.dlg', line 46: rule on dialog start
    source: print 30>=itoa(10);
```

Neben den Fehlermeldungen gibt es noch Warnungen der Art *„*** Warning in file“*, die nicht zu einen Abbruch des Ladens führen.

19.11.2 Erweiterte Prüfungen in der Regelsprache

Die erweiterte Prüfung hat als Ziel, fehlerhaften Regelcode schon vor der Ausführung zu erkennen und den Applikationsprogrammierer auf die Fehlerstelle hinzuweisen. Dabei handelt es sich um eine **semantische Prüfung** von eingesetzten Typen bei Regelcode, der ansonsten syntaktisch korrekt ist.

Die gefunden Problemstellen werden als Warnung in der Form *„*** Warning in“* mit den Details zum Fehler ausgegeben. Es erfolgt kein Abbruch des Ladens.

Ab IDM-Version A.06.02.g erfolgt beim Laden von Modulen eine erweiterte Prüfung des Regelcodes. Dabei werden folgende Prüfungen durchgeführt:

- Der Rückgabetyp des Ausdrucks bei `return` und `pass` wird auf Kompatibilität mit dem Rückgabetyp der Regel überprüft.
- Die korrekte Anzahl und der Datentyp von Parametern beim Aufruf von eingebauten Funktionen, vordefinierten Methoden und benutzerdefinierten Regeln, Funktionen und Methoden wird geprüft.
- Eine Division oder Modulo durch *0* wird erkannt.
- Arithmetische Operationen, die nicht auf Zahlen ausgeführt werden, werden erkannt.
  Erkannt und erlaubt ist noch die String-Konkatenation über den Operator „+“.
- Eine Typprüfung bei Vergleichsoperatoren wird ebenso ausgeführt.
- Eine Typprüfung auf *boolean* wird bei den Operanden von booleschen Operatoren wie `and`, `andthen`, `or`, `orelse` und `not` durchgeführt. Ebenso wird die Konvertierbarkeit in einen *boolean*-Typ bei einer `while`-Anweisung geprüft.
- Der korrekte Typ bei einer Zuweisung zu einer lokalen oder globalen Variablen wird ebenso geprüft.

Für die Typbestimmung gelten folgende Einschränkungen:

- Für die Prüfungen werden die Typen von lokalen und globalen Variablen herangezogen, außerdem statische Werte, sowie die Werte von konstanten, globalen Variablen.
- Ebenso herangezogen werden die Parameter- und Rückgabetypen von eingebauten Funktionen, vordefinierten IDM-Methoden, benutzerdefinierten Regeln, Funktionen und Methoden.
- **Nicht** herangezogen werden die Typen von Attributzugriffen (beim lesenden wie auch beim schreibenden Zugriff).
- **Nicht** herangezogen werden Werte von lokalen und globalen Variablen, die nicht konstant sind.
- Bei der Prüfung werden keine arithmetischen oder booleschen Ausdrücke ausgewertet (bzw. Kalkulationen durchgeführt).
- **Nicht** herangezogen werden über Variablen oder Attribute weiterpropagierte konstante Werte.
- Es erfolgt keine Wertepropagierung über Regel- oder Modulgrenzen hinweg.
- **Nicht** erkennbar sind Fehler und Warnungen aufgrund dynamischer Modell- oder Typ-Änderungen an Attributen.
- Ebenso **nicht** ermittelbar sind Typinformationen von Attributen und Methoden, die über eine indirekte Referenzierung (relativer Pfad oder Variablen) erfolgen.
- Für die Prüfungen von Variablen mit Datentyp *object* und Gültigkeitsbereich wird das Modell-Objekt herangezogen.
- Beim Typ *anyvalue* wird automatisch ein passender Datentyp bei der Prüfung vorausgesetzt.

Beispiel für erkennbare Typfehler und andere Warnungen sowie seine Grenzen:

```
dialog D

window Wi
{
  .title "Main Window";
}

constant integer GVarI := 0;

rule integer Rule(string S, boolean B:=false)
{
  variable integer   I   := .xleft;  // value mismatch variable type
  variable object    O   := Rule;
  variable anyvalue  A   := .xleft;
  variable object[D] Ref := null;

  if S < 10  then  // Bad operation
    return S;      // value mismatch return type
  endif
  setvalue(123, .xleft, Wi);    // Invalid argument #1
  Wi:delete(button_ok);         // Invalid argument #1
  perform Rule("a", true, -1);  // Wrong number of arguments
  I := itoa(123);               // value mismatch type
```

```
  print 10 / GVarI;             // Division by zero
  I := D:Rule(atoi("4711"));    // Invalid argument #1
  Ref:Rule(4711);               // Invalid argument #
  if false then
    return "#" + Rule(S);       // value mismatch return
  endif

  // Errors exclusively detected by the rule interpreter
  print 3 % (1 - 1);  // Undetected: 3%0 => modulo by zero
  O(4711);            // Undetected: Rule(4711) => bad type of call
                      //                           by value parameter 1
  I := Wi.title;      // Undetected: assign mismatch
  I := this.Wi.xleft + this.Wi.title;  // Undetected: assign mismatch
  S := I + A;         // Undetected: Bad operation
  this:Rule(4711);    // Undetected: error in method call
}

on dialog start
{
  Rule("abc");
  exit();
}
```

19.11.3 Steuerung des Schweregrads

Die bei der Analyse entdeckten Problemstellen lassen sich durch die Startoption **-IDManalyseseverity=<string>** in ihrer Behandlung beeinflussen.

Für das <*string*>-Argument der Option werden folgende Werte erkannt:

Wert	Bedeutung
W	Die in der erweiterten Typprüfung gefundenen Problemstellen werden als Warnung ausgegeben (Standard).
E	Die in der erweiterten Typprüfung gefundenen Problemstellen werden als Fehlermeldung (im Fehlerkanal, somit auch als Fehlerdialog unter MICROSOFT WINDOWS) ausgegeben und führen zum Abbruch des Ladevorgangs.
F	Die gefundenen Problemstellen werden als fatale Fehler angesehen. Beim ersten Problem wird abgebrochen und die Anwendung beendet.
0	Die erweiterte Prüfung meldet keine Problemstellen (Kompatibilitätsmodus).

Nicht beeinflussbar sind Gültigkeitsverletzungen. Diese werden immer als Fehler ausgegeben und führen zum Abbruch des Ladevorgangs.

19.11.4 Aufruf von Fehlerhandlern

Registrierte Fehlerhandler werden auch für Fehler aufgerufen, die im Rahmen der Fehleranalyse gefunden werden. Dabei wird eine Fehlerstelle mit Datei und Zeilennummer an den Fehlerhandler weitergegeben. Fehlermeldungen aus der Fehleranalyse sind an der *task EITK_analyzer* in der **DM_ErrorInfo**-Struktur erkennbar. Die Fehlerausgabe erfolgt nach dem Aufruf der Fehlerhandler.

19.11.5 Erkennung fehlender `return`-Anweisungen

Im Rahmen der Fehleranalyse wird aus dem Syntaxbaum auch ein Kontrollflussgraph erzeugt. Auf Basis der **Kontrollflussanalyse** werden fehlende `return`-Anweisungen erkannt und entsprechende Warnungen *„Missing return Statement“* ausgegeben.

Die Analyse umfasst benannte Regeln, Ereignisregeln (mit und ohne Vorbedingungen), Simulationsregeln und Methoden (benutzerdefiniert und redefiniert). Sie berücksichtigt auch `pass`- und `throw`-Anweisungen, die ebenfalls zum Beenden einer Regel führen.

Es wird eine Fehlerstelle ausgegeben, wobei versucht wird, dem Block mit dem fehlenden `return` möglichst nahe zu kommen.

Das Fehlen von `return` wird für folgende Anweisungen erkannt:

- `if then else`, `elseif`
- `case`, `in`
- `for`, `foreach`, `while`

Die Kontrollflussanalyse kann mit der Startoption **-IDMflowgraph** oder der Umgebungsvariablen `IDM_FLOWGRAPH` ein- (=*true*, Standardwert) und ausgeschaltet (=*false*) werden, wobei die Umgebungsvariable Vorrang vor der Option hat.

Hinweis zur Position von *„Missing return“* Warnungen

„Missing return“ Warnungen werden für die letzte Zeile des Codeblocks angezeigt, in dem die `return`-Anweisung fehlt. Wenn in diesem Block allerdings keine Anweisung vorhanden ist, wird die Warnung für eine Codezeile **vor** dem betreffenden Block angezeigt. Dabei ist zu beachten, dass die „Blockgrenzen“ (z.B. das `else` einer `if`-Anweisung oder das `otherwise:` einer `case`-Anweisung) **nicht** als Anweisungen behandelt werden.

19.12 Neues Werkzeug TRACEFILE ANALYZER

Das Auswerten von IDM-Tracedateien kann aufgrund der Datenmengen unübersichtlich und zeitintensiv sein. Der TRACEFILE ANALYZER soll hierbei als effizientes Hilfsmittel zu Fehler- oder Problemsuche bei IDM-Anwendungen dienen und dem Benutzer eine detaillierte und schnelle Auswertung von IDM-Tracedateien ermöglichen. Der TRACEFILE ANALYZER macht dabei die Analyse von Tracedateien durch seine Darstellung, interaktive Anpassungsmöglichkeiten und Hilfswerkzeuge noch einfacher und übersichtlicher als ein herkömmlicher Text-Editor.

So stehen dem Anwender verschiedene Navigationsmechanismen für die Navigation der Trace-Hierarchie, Fehler oder benutzerdefinierten Markierungen sowie eine Navigationshistorie zur Verfügung. Neben praktischen Hilfswerkzeugen wie Filterung, Suche, Markierungen und dem Aufrufstapel sollen farbliche Hervorhebungen und Inhaltstypisierung als Erleichterung für die Bewältigung großer und unübersichtlicher Datenmengen dienen.

Zusätzlich ermöglicht der TRACEFILE ANALYZER das parallele Arbeiten an mehreren Tracedateien oder an verschiedenen Stellen derselben Tracedatei.

Verfügbarkeit

Der TRACEFILE ANALYZER ist verfügbar für den IDM FÜR QT und den IDM FÜR WINDOWS.

Siehe auch

Handbuch „Tracefile Analyzer“

19.13 Alternativer Import mit use

19.13.1 Der alternative Import-Mechanismus

Zusätzlich zum bisherigen Weg, auf ein Modul über ein ***Import***-Objekt zuzugreifen, kann das Schlüsselwort **use** genutzt werden um den Umgang mit Imports, Modulen, Interface- und Binärdateien zu erleichtern.

Im Gegensatz zum Import erfolgt die Auswahl des Moduls nicht durch einen Dateipfad auf die Interface-Datei, sondern über einen Bezeichner-Pfad – genauer gesagt dem **Use-Pfad**.

Der letzte Bezeichner im Use-Pfad ist, analog zum Bezeichner eines Imports, der Vater-Bezeichner mit dem die exportierten Objekte im importierenden Modul bekannt gemacht werden. Die vorderen Bezeichner im Use-Pfad definieren eine „Paket-Hierarchie“ und können somit zur Untergliederung der Module genutzt werden. Bei der Evaluation von Objekt-Pfaden bleiben diese Teile unberücksichtigt.

Beispiel

```
!! file: Main.dlg

dialog MainDlg

use Models;
use Base.Fonts;

MWiMain WiMain {
  .font Fonts.FnBig;
}
```

```
!! file: Models.mod

module ModModels

export window MWiMain {
  on close {
    exit();
  }
}
```

```
!! file: Base/Fonts.mod

module ModFonts

export font FnBig "24.Arial";
```

Die Verzeichnisse, in denen die eigentliche Modul- und Interface-Datei gesucht werden, können als Startoption oder eine spezielle Umgebungsvariable, über einen Auf ruf von **DM_ControlEx()** oder in der Regelsprache gesetzt werden.

Das Vorhandensein einer Interface-Datei ist dabei nicht zwingend notwendig, aber sinnvoll um das Laden von Modulen ohne direkte Nutzung zu erleichtern und das Nachladen der Implementierung bei Bedarf zu unterstützen.

19.13.1.1 Besonderheiten

Module die über **use** angezogen werden, sind im Dialog immer nur einmal geladen. Dieser Umstand erleichtert die Verwendung von Basismodulen, da nicht versehentlich durch eine fehlerhafte Import-Kette ein Modul mehrfach geladen wird.

Die Entkopplung von einem Interface- oder Modul-Dateinamen hat mehrere Vorteile:

1. Die Verwendung von eigenen Umgebungsvariablen für den Ort der Interface- und Binärmodule entfällt.
2. Die Erzeugung der Interface- und Binärdateien wird vereinfacht.
3. Das Laden von Modulen ohne Interface-Datei ist nun möglich, da ein zentraler Suchpfad für Interfaces, Module und Binärmodule vorhanden ist.

Eine weitere Besonderheit ist die Vorgabe von Dateiendungen (Datei-Suffixe) für Quelltext-Dateien (*.dlg* und *.mod*), Interface-Dateien (*.if*) und Binärdateien (*.bin*).

Die Umsetzung von einem Use-Pfad zu einem Dateipfad geschieht dabei wie folgt: Die vorangestellten Paketbezeichner entsprechen einem Verzeichnispfad, der letzte Bezeichner dem Basisnamen des Moduls.

Das heißt im Beispiel oben ergibt sich für den Use-Pfad `Base.Fonts` der Basis-Dateiname „Base\Fonts“. Das Quelltext-Modul heißt *Base\Fonts.mod*, die dazugehörige Interface-Datei *Base\Fonts.if* und das Binärmodul *Base\Fonts.bin*.

Wird nun über `use Base.Fonts` ein Modul angezogen, dann wird im Suchpfad nach diesen drei Dateinamen gesucht um das Interface bzw. die Implementierung zu laden. In der Entwicklungsversion des IDM wird dabei in der Reihenfolge Interface-Datei → Quelltextdatei → Binärmodul vorgegangen. Die Laufzeitversion des IDM kann nur Binärdateien laden.

Einige weitere Unterschiede zum ***Import***-Objekt sind noch vorhanden: Es gibt keine Steuermöglichkeiten über die Attribute *.application*, *.static* und *.load*. Es wird immer versucht, erst das Interface heranzuziehen und dann die Implementierung (Quelltext, Binär). Die Implementierung wird nur wenn benötigt nachgeladen. Ein direkte Steuerung des Ladens entfällt somit.

Für ein dynamisches Laden und Entladen gibt es die neuen Methoden **:use()** und **:unuse()**.

Grundsätzlich können Binärmodule und Interface-Dateien, die für die Nutzung durch ein ***Import***-Objekt erzeugt wurden, unverändert auch mit **use** genutzt werden. Umgekehrt gilt dies nicht zwangsläufig da hier meist der Hinweis auf den Modulpfad im Interface fehlt.

19.13.1.2 Groß- und Kleinschreibung in Dateinamen

Sie ist im IDM bei Bezeichnern und somit auch beim Use-Pfad sowie für die Dateinamen von Bedeutung!

19.13.1.3 Empfehlungen

Zwar ist eine gemischte Nutzung von **import** und **use** möglich, aber **nicht** empfehlenswert, da das ***Import***-Objekt ein mehrfaches Laden von Modulen provoziert. Deshalb wird empfohlen, komplett von **import** auf **use** umzustellen.

Außerdem wird empfohlen , für Datei- und Verzeichnisnamen grundsätzlich die gleiche Schreibweise wie im Use-Pfad zu wählen. Am besten sollten Module einen eindeutigen Namen besitzen, der auch nicht in einer abgewandelten Form mit anderer Groß- und Kleinschreibung auftaucht.

Statt ein Modul über **import** zu laden, sollte man Modelle verwenden, die problemlos mehrfach instanziiert werden können.

19.13.2 Sprachbeschreibung und Use-Pfad

Folgende Sprachbeschreibung gilt für das Schlüsselwort **use**:

```
{ export | reexport } use { <Use-Pfad> }

<Use-Pfad>   ::= [ <Paket-Pfad> ] <Bezeichner>
<Paket-Pfad> ::= [ <Paket-Pfad> ] <Bezeichner> .
```

Die **Bezeichner** sollten mit einem Großbuchstaben anfangen ('A' – 'Z') und kann danach auch Kleinbuchstaben, Ziffern und Unterstriche ('_') enthalten. Ein Bezeichner darf maximal 31 Zeichen haben.

Beispiele für gültige use-Verwendungen

```
use Defaults;
use DEFAULTS;
use Max_Wi09_;
use Modul78.Aa.MasterA__1z;
use BaseMod.M_ods.Wins.MWiEnter1_9;
```

Der **Use-Pfad** besteht dabei aus einem Bezeichner (Identifikator) am Ende und repräsentiert den Basisnamen des Moduls, sowie den Vater-Bezeichner für den weiteren Zugriff auf die exportierten Objekte (falls diese nicht eindeutig aufzulösen sind). Dieser Bezeichner muss nicht dem Modulbezeichner entsprechen.

Der **Paket-Pfad** wiederum dient zur logischen Untergliederung von mehreren Modulen und wird bei der Suche nach dem Modul als Verzeichniskette vorangestellt.

19.13.3 Use-Pfad, Dateinamen und Namensrestriktionen

Der IDM konvertiert intern einen Use-Pfad in einen Dateipfad, um diesen bei der Suche von Interface-Datei, Quelltextdatei oder Binärmodul zu verwenden.

So wird der Use-Pfad `Modul78.Aa.MasterA__1z` zum Dateipfad *Modul78\Aa\MasterA__1z*.

Außerdem gibt es einen vorgegebenen Satz von Dateiendungen, die bei Suche und Zugriff relevant sind und intern an den Basis-Dateipfad angehängt werden:

Dateiendung	Bedeutung
.dlg	Quelltext eines Dialogs
.mod	Quelltext eines Moduls
.if	Interface-Datei
.bin	Binärdatei eines Dialogs oder Moduls

Da der Use-Pfad über IDM-Bezeichner aufgebaut ist, bedeutet dies natürlich, dass Datei- und Verzeichnisnamen mit einem Großbuchstaben beginnen können sowie die Groß- und Kleinschreibung eine Bedeutung hat und wichtig ist! Dies ist bei der Nutzung auf Dateisystemen, die dies nicht unterscheiden, zu beachten.

Das heißt im IDM gilt: `use Default;` ist etwas anderes als `use DEFAULT;`. Unter Windows kann das Modul aber durchaus bei Vorhandensein der Modul-Datei „DEFault.Mod“ gefunden werden.

Grundsätzlich kann auch der Dialog über einen Use-Pfad angesprochen (geladen) werden.

Falls unüberbrückbare Probleme bei der Umwandlung zwischen Use-Pfad und Dateipfad bestehen, kann das interne Konvertierungsverhalten über die Option **-IDMusepathmodifier** gesteuert werden.

19.13.4 Suchpfad für Interface-, Modul-, Dialog- und Binärdateien

Der IDM besitzt nun einen zentralen Suchpfad für das Auffinden von IDM-Dateien. Dieser Suchpfad kann eine oder mehrere absolute oder relative Verzeichnisangaben mit Semikolon („;“) getrennt beinhalten.

Standardmäßig ist der Suchpfad auf „~;“ gesetzt, also auf den Sonderpfad „~“ und den Leerpfad *""*. Diese haben folgende Bedeutung:

~ oder *~:*	Sucht unterhalb des Verzeichnisses, in dem sich die Applikation befindet.
"" (Leerpfad)	Sucht im aktuellen Arbeitsverzeichnis (Verhalten wie in den Vorgängerversionen).
<ENVNAME>:	Sucht in den Pfaden, die in der Umgebungsvariablen `<ENVNAME>` definiert sind.

Bei der Suche nach einer IDM-Datei, z.B. über **DM_LoadDialog()** oder die eingebaute Funktion **load ()**, wird wie folgt vorgegangen:

1. Wenn der angegebene Pfad absolut ist, wird die Datei an dem exakt angegebenen Pfad gesucht ohne das Anhängen einer Erweiterung (kompatibles Verhalten wie bisher).
2. Handelt es sich um einen relativen Pfad mit Dateierweiterung, wird dieser im Suchpfad mit exakt dieser Erweiterung gesucht. Dabei stellt der Standard-Eintrag *""* (Leerpfad) das kompatible Verhalten wie bisher dar.
3. Ist am Dateinamen keine Erweiterung vorhanden, werden je nach Aktion die möglichen Erweiterungen bei allen Suchpfaden durchprobiert. Zum Beispiel werden bei **DM_LoadDialog()** die Erweiterungen *.dlg*, *.mod*, *.bin* durchprobiert. Bei einem „use“ wird versucht, mit Vorrang das Interface zu bekommen, also wird die Reihenfolge *.if*, *.bin*, *.mod* durchprobiert.

Folgende Einstellmöglichkeiten für den Suchpfad sind vorhanden:

- Beim Start einer IDM-Anwendung über die Option **-IDMsearchpath <Suchpfad>**.
- Über die Umgebungsvariable `IDM_SEARCHPATH`.
- Durch den Aufruf von **DM_ControlEx()** mit *DMF_SetSearchPath* und dem Suchpfad als *data*-Argument.
- Über das Attribut *.searchpath* am ***Setup***-Objekt.

Für eine selbst kompilierte IDM-Anwendung empfehlen wir die Nutzung von **DM_ControlEx()** in der **AppMain()**-Routine mit Angabe eines Applikations-relativen Pfades über „~“ und ohne andere relative Angaben wie den Leerpfad oder *"."*. So würde die Ausführung von *DM_ControlEx((DM_ID)0, DMF_SetSearchPath, "~:dlg")* sicherstellen, dass die Dialogdateien nur im Unterverzeichnis *dlg* parallel zur Anwendung gesucht werden. Ist der Suchpfad erst einmal gesetzt, kann dann auch über **DM_LoadDialog()** das initiale Laden des Hauptdialogs über einen relativen Dateipfad ohne Dateiendung erfolgen.

Beispiele für korrekte und erlaubte Suchpfade

""	Sucht im aktuellen Arbeitsverzeichnis.
"."	
"~"	Sucht im Verzeichnis in dem sich die Applikation befindet.
"if;bin;mods;customer/modules"	
"~:dlg;~:../mods;."	
".;MODPATH:if;MODPATH:bin"	Sucht im Arbeitsverzeichnis sowie in den Verzeichnissen *bin* und *if* innerhalb der Verzeichnisse, die in der Umgebungsvariablen `MODPATH` angegeben sind.

Beim Öffnen und Laden von IDM-Dateien versucht der IDM den Dateityp festzustellen, um zu erkennen, ob die Datei eine Interface-Datei, ein Dialog, Modul oder eine Binärdatei ist. Zu diesem Zweck untersucht er die ersten 1.024 Bytes der Datei, ob sie die Signatur für eine IDM-Binärdatei hat, oder entsprechende Schlüsselwörter die ein Interface, Dialog oder Modul kennzeichnen. Andernfalls wird die Dateiendung zur Bestimmung herangezogen.

19.13.5 Vergleich zwischen import und use

Funktion, Konzept	use	import
Mehrfaches Laden von Modulen.	**Nicht** möglich. Modul ist in einem Dialog immer nur einmal geladen (gilt nur für den Zugriff mit „use“).	Mehrfaches Laden des gleichen Moduls durch anderen Import-Bezeichner bzw. bei Lücken in der Import-Hierarchie der importierenden Module bis hoch zum Dialog möglich.
Ankopplung von Funktionen an ein Applikationsobjekt.	Über das *.masterapplication[enum]*-Attribut möglich.	Über das *.application*-Attribut möglich.
Erzeugung von Interface- und Binärdateien in einem Schritt.	Über **-compile** bzw **-recompile** möglich. Der Dialog sollte aber alle Module über „use“ angezogen haben.	**Nicht** möglich.
Quelltext, Interface und Binärmodul im gleichen Verzeichnis.	**Ja**, möglich durch die unterschiedlichen Dateiendungen.	**Nicht** möglich.
Verteilung der Module in Unterverzeichnisse.	**Ja**, über Paketpfad (Teil des Use-Pfads) oder über den Suchpfad.	**Ja**, durch Suchsymbol, Pfadlisten in Such-Variablen und als Auflistung in Interface- und Moduldatei.

19.13.6 Kompilieren von Interface- und Binärdateien

Während der Entwicklung einer IDM-Applikation und der Nutzung mit Quelltext-Dateien dienen die Interface-Dateien als Repräsentation der extern verfügbaren (exportierten) Objekte eines Moduls, also der Schnittstelle nach außen. Durch die Trennung von Schnittstellen-Definiton und Implementierung ist der IDM in der Lage, das Laden der Implementierung (also des Moduls) nur bei Bedarf durchzuführen.

Durch die Erweiterung um „use“ kann diese Schnittstellen-Definition auch direkt aus dem Modul, also der Implementierung erfolgen. Aus oben genannten Grund macht es aber dennoch Sinn Interface-Dateien zu nutzen.

Für die Auslieferung einer IDM-Anwendung an den Endkunden bedarf es einer Binärfassung der Dialoge und Module.

Grundsätzlich können Interface- und Binärdateien auf dem herkömmlichen Weg über die Aktionen **-writeexport** und **-writebin** erzeugt wegen. Solange diese Dateien mit der richtigen Dateiendung an der richtigen Stelle im Suchpfad stehen, hat die Anweisung „use“ damit kein Problem.

Für modularisierte Dialoge die „use“ verwenden gibt es nun aber viel einfachere Aktionen, um alle Interface- und Binärdateien in einem Schritt zu erzeugen.

-compile	Erneuert alle Interface- und Binärdateien für geänderte Module und Dialoge.
-recompile	Erzeugt alle Interface- und Binärdateien neu.
-cleancompile	Löscht alle Interface- und Binärdateien.

Dazu muss der Dialog fehlerlos geladen werden können. Erzeugt werden Interface- und Binärdateien nur für den angegebenen Dialog und alle per „use“ geladenen Module. Für Module, die per „import“ angezogen werden, wird **keine** Interface- und Binärdatei erzeugt!

Die zu ladenden Modul- und Dialogdateien müssen als Quelltext vorliegen. Bei einem Ladefehler werden keine Dateien erzeugt.

Beispiel

```
!! file: Main.dlg

dialog MainDlg

use Models;
use Base.Fonts;

MWiMain WiMain {
  .font Fonts.FnBig;
}
```

```
!! file: Models.mod

module ModModels

export window MWiMain {
  on close {
    exit();
  }
}
```

```
!! file: Base/Fonts.mod

module ModFonts

export font FnBig "24.Arial";
```

Über folgendes Kommando kann man in einem Schritt alle Interface- und Binärdateien erstellen:

```
pidm -compile Main
```

Nun sollten die folgenden Dateien parallel zu den Quelltext-Dateien stehen:

```
Main.bin
Models.bin
Models.if
Base/Fonts.bin
Base/Fonts.if
```

Fügt man nun in der Datei *Base/Fonts.mod* noch die Zeile *export font FnSmall…* hinzu, so aktualisiert man einfach mit folgendem Kommando:

```
pidm -compile Main
```

Was zum erneuten Schreiben der folgenden Dateien führt:

```
Base/Fonts.bin
Base/Fonts.if
```

Die anderen Interface- und Binärdateien werden nicht neu geschrieben, da das Dateidatum dieser Dateien neuer ist als das der entsprechenden Quelldatei.

Möchte man auf jeden Fall alle Interface- und Binärdateien neu erzeugen lassen, so geht dies über:

```
pidm -recompile Main
```

Möchte man alle Interface- und Binärdateien löschen, so geht dies über:

```
pidm -cleancompile Main
```

Um die Interface- und Binärdateien in ein anderes Verzeichnis hinein zu erzeugen, dienen die Optionen **-ifdir <Verzeichnispfad>** und **-bindir <Verzeichnispfad>**.

19.13.7 Dynamisches Laden und Entladen von Modulen

Für die dynamische Nutzung von Modulen gibt es zwei neue Methoden:

- **:use()**
  Um ein Modul zugänglich zu machen und wenn nötig zu laden. Dies entspricht dem `use`-Kommando in einem Dialog oder Modul.
- **:unuse()**
  Dient dazu die `use`-Beziehung wieder aufzulösen, also das Modul zu „entladen“, wobei das Modul erst dann komplett entfernt wird, wenn es keine weiteren `use`-Beziehungen (z.B. von anderen importierenden Modulen) gibt.

Beispiel

```
!! file DynLoad.dlg
dialog Dlg

window Wi
{
  pushbutton PbUnload
  {
    .yauto -1;
    .text "Unload";
    .sensitive false;

    on select
    {
      this.dialog:unuse("DynMod");  // Entladen des Moduls DynMod
      this.sensitive := false;
    }
```

```
  }

  on close { exit(); }
}

on dialog start
{
  variable object Module, Model, Child;

  Module := this:use("DynMod");  // Laden des Moduls DynMod
  if Module<>null then
    PbUnload.sensitive := true;
    Model := parsepath("MPbBeep");
    if Model<>null then
      Child := create(Model, Wi);
    endif
  else
    print "Cannot load DynMod module!";
    exit();
  endif
}

!! file DynMod.mod
module ModDyn

export model pushbutton MPbBeep
{
  .text "Beep";

  on select
  {
    beep();
  }
}
```

19.13.8 Umgebung und IDM-Startoptionen

Folgende Startoptionen sind hinzugekommen:

-IDMsearchpath <searchpath>

Diese Option setzt den Suchpfad über den Dialog-, Modul-, Interface- und Binärdateien gesucht werden. Diese Option überschreibt den Suchpfad, der auch durch Angabe der Umgebungsvariablen `IDM_SEARCHPATH` gesetzt sein kann.

Der Suchpfad ist eine Semikolon-separierte Liste von Pfaden (absolute wie relative) mit folgenden Besonderheiten:

~ oder *~:*	Sucht unterhalb des Verzeichnisses, in dem sich die Applikation befindet.
"" (Leerpfad)	Sucht im aktuellen Arbeitsverzeichnis (Verhalten wie in den Vorgängerversionen).
<ENVNAME>:	Sucht in den Pfaden, die in der Umgebungsvariablen `<ENVNAME>` definiert sind.

Hinweis

Der Suchpfad kann auch über **DM_ControlEx()** und über das ***Setup***-Objekt gesetzt werden.

Siehe auch

Kapitel „Suchpfad für Interface-, Modul-, Dialog- und Binärdateien“

-IDMusepathmodifier <string>

Diese Option steuert die Umwandlung eines Use-Pfads in einem Dateinamen. Sie erlaubt die Steuerung der Dateinamensumwandlung in Groß- bzw. Kleinbuchstaben. Die Option sollte nur mit Bedacht eingesetzt werden.

Folgende Möglichkeiten bestehen:

L Der gesamte Dateipfad wird in Kleinbuchstaben gewandelt.
Aus dem Use-Pfad `Base.Colors` werden also die Dateinamen *base/colors.if*, *base/colors.mod* und *base/colors.bin*.

U Der gesamte Dateipfad inklusive Erweiterung wird in Großbuchstaben gewandelt.
Aus dem Use-Pfad `Base.Colors` werden also die Dateinamen *BASE/COLORS.IF*, *BASE/COLORS.MOD* und *BASE/COLORS.BIN*.

u Der gesamte Dateipfad ohne Erweiterung wird in Großbuchstaben gewandelt.
Aus dem Use-Pfad `Base.Colors` werden also die Dateinamen *BASE/COLORS.if*, *BASE/COLORS.if* und *BASE/COLORS.if*.

l Nur der erste Buchstabe in den Dateipfadteilen wird in Kleinbuchstaben gewandelt.
Aus dem Use-Pfad `BaseModels.MWin` werden also die Dateinamen *baseModels/mWin.if*, *baseModels/mWin.mod* und *baseModels/mWin.bin*.

19.13.9 Startoptionen für idm.exe, pidm.exe und Builder-Prozess

Folgende Startoptionen sind hinzugekommen:

-compile

Diese Option bewirkt, dass für das geladene Modul bzw. den geladenen Dialog sowie alle per **use** importierten, geladenen Untermodule die Interface- und Binärdateien geschrieben werden.

Diese werden allerdings nur neu geschrieben, wenn das Dateidatum der Quell-Datei(en) neuer ist als die Interface- und Binärdateien. Diese Option ist für den Einsatz in einem Makefile gedacht.

Eventuell notwendige Verzeichnisse, die sich aus dem Use-Pfad ergeben, werden wenn erforderlich angelegt. Die Ausgabeverzeichnisse der Zieldateien können über die Optionen **-ifdir** und **-bindir** gesteuert werden.

Da die erzeugten Interface- und Binärdateien vorgegebene Dateiendungen (*.if* und *.bin*) besitzen, ist die Erzeugung dieser Dateien gemischt mit den Quellen möglich.

Achtung

Vorhandene Dateien werden überschrieben.

-recompile

Diese Option funktioniert analog zu **-compile**, nur dass die Zieldateien immer neu erzeugt werden, also keine Datumsprüfung erfolgt.

This option works the same way as **-compile**, except that the target files are always regenerated, thus there is no date check.

-cleancompile

Diese Option löscht alle Interface- und Binärdateien für die geladenen Dialog- und Moduldateien sowie alle mit **use** importierten Untermodule.

-compile1
-recompile1
-cleancompile1

Diese Optionen führen die jeweiligen Aktionen lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

-bindir <Verzeichnispfad>

Diese Option erlaubt die Angabe des Zielverzeichnisses in das die Binärdateien (mit der Dateiendung *.bin*) geschrieben werden. Standardmäßig erfolgt die Ausgabe in das Arbeitsverzeichnis.

-ifdir <Verzeichnispfad>

Diese Option erlaubt die Angabe des Zielverzeichnisses in das die Interface-Dateien (mit der Dateiendung *.if*) geschrieben werden. Standardmäßig erfolgt die Ausgabe in das Arbeitsverzeichnis.

-noif

Mit dieser Option wird das Generieren der Interface-Dateien unterbunden.

Siehe auch

Kapitel „Der alternative Import-Mechanismus“

19.13.10 Ergänzungen zur Funktion DM_ControlEx

Folgende Aktionen sind hinzugekommen:

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

19.13.11 Attribut .searchpath

Mit diesem Attribut kann der aktuelle Suchpfad, der für Dialog-, Modul-, Interface- und Binärdateien verwendet wird, erfragt und umgesetzt werden.

	Bezeichner		Datentyp
Regelsprache	**.searchpath**		string
C	AT_searchpath		DT_string
COBOL	AT-searchpath		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	setup		
Zugriff	get, set	Standardwert	*"~;"*
changed-Ereignis	nein	Vererbung	nein

Siehe auch

Kapitel „Suchpfad für Interface-, Modul-, Dialog- und Binärdateien“

19.13.12 Methode :use()

Mit dieser Methode wird an einem ***Dialog***- oder ***Module***-Objekt ein neues Modul über seinen Use-Pfad importiert. Man kann damit das Modul analog zu einer statischen **use**-Anweisung einbinden, sodass das Modul im Dialog nur einmal geladen wird.

Ist der *Check*-Parameter beim Aufruf auf *true* gesetzt, so wird nur geprüft ob schon ein Import mit „use“ an diesem Objekt vorhanden ist.

Definition

```
object :use
(
      string  Usepath input
  { , boolean Export  := false input
  { , boolean Check   := false input } }
)
```

Parameter

string *Usepath* input

Mit Hilfe dieses Parameters wird als Use-Pfad (in Form eines Bezeichner-Pfades) das Modul angeben.

*boolean **Export** := false input*

Wenn dieser optionale Parameter *true* ist, wird der „use“ beim Anlegen zusätzlich als exportiert gekennzeichnet (Standardwert *false*).
Der Parameter ist nur von Bedeutung wenn *Check* = *false* ist.

*boolean **Check** := false input*

Ist dieser optionale Parameter *true*, so wird nur geprüft, ob das per Use-Pfad angegebene Modul schon mit „use“ importiert wurde (Standardwert *false*).

Rückgabewert

null	Bei *Check* = *false*: Modul konnte nicht gefunden oder geladen werden.
null	Bei *Check* = *true*: Kein „use“ für diesen Use-Pfad vorhanden.
<Id>	Id des importierten Moduls.

Objekte mit dieser Methode

- dialog
- module

Beispiel

```
window Wi
{
  edittext EtUsepath
  {
    .xauto 0;
    .content "Plugins.View.CustomerModels";
  }

  pushbutton PbUse
  {
    .yauto -1;
    .text "Use";

    on select
    {
      if this.module:use(EtUsepath.content, false, true) = null then
        if this.module:use(EtUsepath.content) = null then
          print "Can't import module!";
        endif
      endif
    }
  }
}
```

Siehe auch

Kapitel „Der alternative Import-Mechanismus“

Kapitel „Sprachbeschreibung und Use-Pfad“

19.13.13 Methode :unuse()

Mit dieser Methode wird an einem ***Dialog***- oder ***Module***-Objekt ein über „use“ eingebundenes Modul wieder entfernt bzw. entladen. Falls es nicht anderweitig verwendet wird, so wird es komplett entladen (mitsamt der Entfernung aller Instanzen).

Definition

```
boolean :unuse
(
  anyvalue UsepathOrModul input
)
```

Parameter

anyvalue *UsepathOrModule* input

Mit Hilfe dieses Parameters wird als Use-Pfad (in Form eines Bezeichner-Pfades) oder direkt als Module-ID das Modul angeben.

Rückgabewert

false	Modul ist gar nicht bzw. nicht durch „use“ importiert.
true	Der Zugriff auf das Modul per „use“ wurde entfernt.

Objekte mit dieser Methode

- dialog
- module

Beispiel

```
dialog D
use Customer.Models;

window Wi
{
  MGbCustomer {}
  pushbutton PbUnuse
  {
    .yauto -1;
    .text "Unuse";

    on select
    {
      if this.module:unuse("Customer.Models") then
        this.sensitive := false;
      endif
    }
  }
}
```

Siehe auch

Kapitel „Der alternative Import-Mechanismus“

Kapitel „Sprachbeschreibung und Use-Pfad“

19.13.14 Ergänzungen am IDM-Editor

19.13.14.1 Objekte-Browser

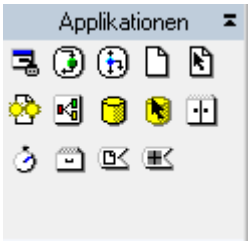
Der Import von Modulen über **use** wird mit folgendem Symbol angezeigt:

use <Use-Pfad>

Anstatt des Bezeichners wird der gesamte Use-Pfad angezeigt.

19.13.14.2 Applikationen in der Werkzeugkiste

Die *„Werkzeugkiste“* wurde ergänzt:

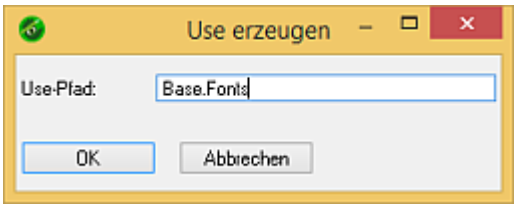


Die Symbole für die Applikationen sind im einzelnen:

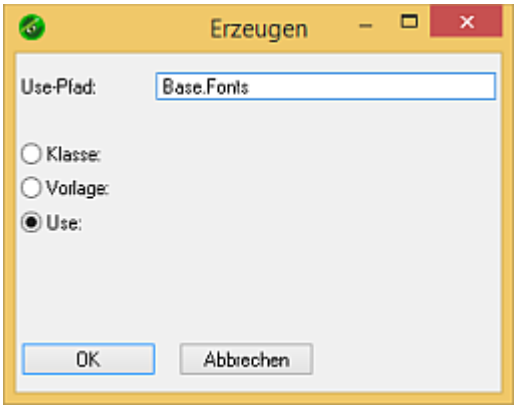
Modul-Import über **use**

19.13.14.3 Änderung an den Erzeugen-Dialogen

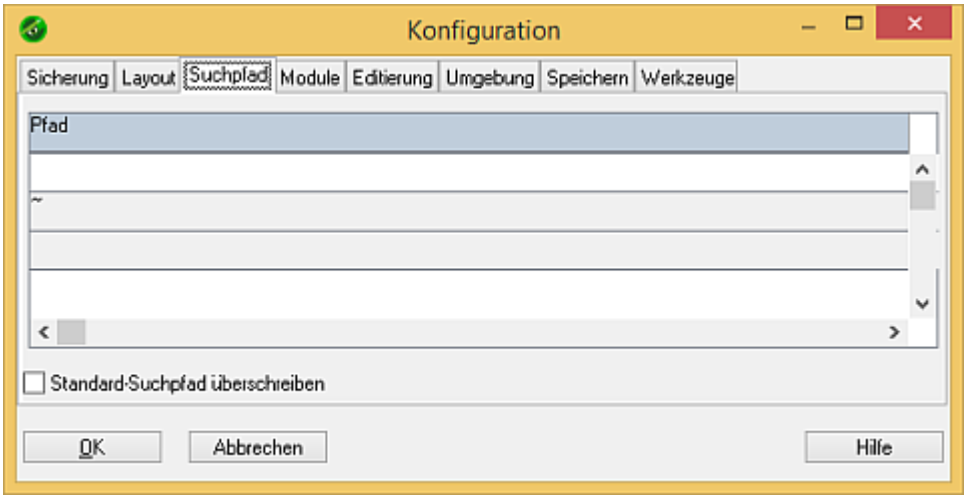
Das direkte Erzeugen über die *„Applikationen“* in der *„Werkzeugkiste“* führt zum Dialog:



Der allgemeine *„Erzeugen“*-Dialog wurde um eine neue Auswahl *„Use“* ergänzt:



19.13.14.4 Konfigurationsseite Suchpfad



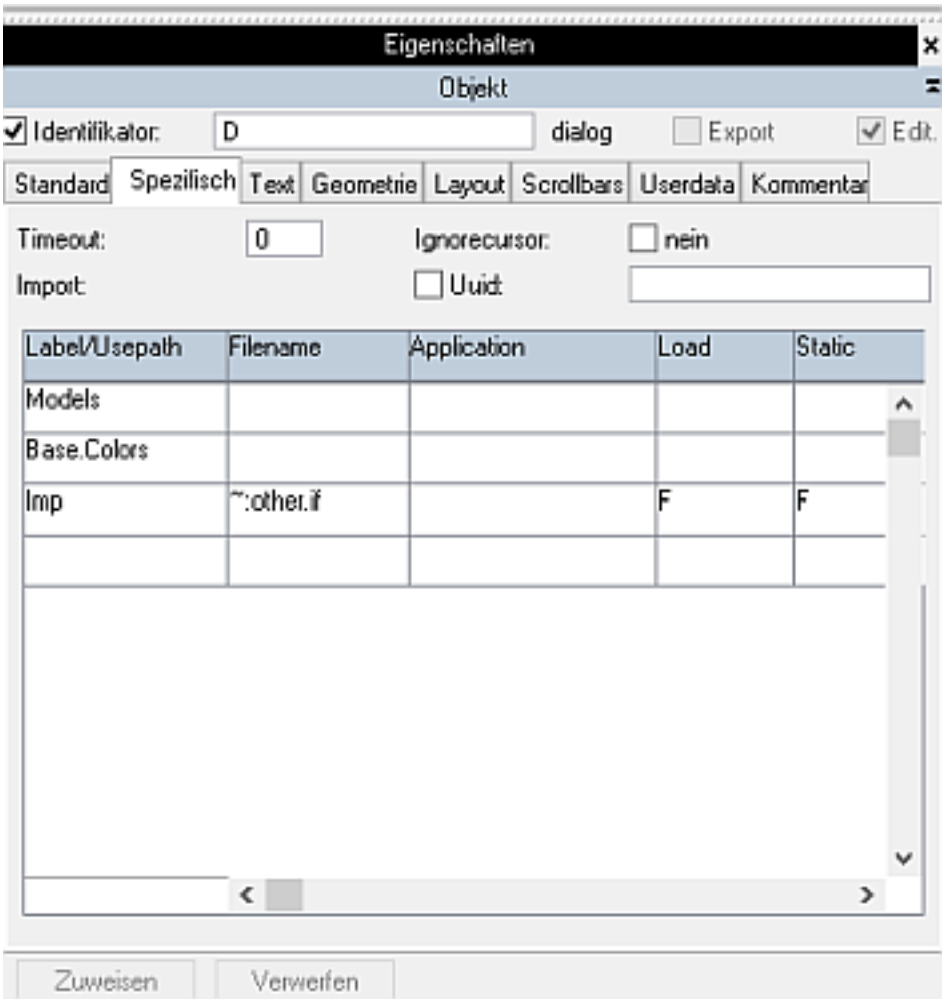
In der Seite *„Suchpfad“* sind die Verzeichnispfade hinterlegt, in denen Modul- und Interface-Dateien gesucht werden, die über „use“ angezogen werden.

Dabei sind die Standard-Suchpfade die beim Starten des IDMED angegeben wurden, grau hinterlegt und können normalerweise nicht geändert werden. Diese sind nur überschreibbar, wenn die Checkbox *„Standard-Suchpfad überschreiben“* aktiviert wird. Das leere Eingabefeld vor den grau hinterlegten Suchpfaden erlaubt das Hinzufügen von neuen Suchpfaden.

Das leere Feld im Standard-Suchpfad repräsentiert dabei die Standardsuche im Arbeitsverzeichnis. Das Zeichen „~“ erlaubt die Suche parallel zur Anwendung. Man kann auch über einen Pfad der Form *„ENVNAME:“* eine Suche über eine Umgebungsvariable durchführen.

19.13.14.5 Spezifische Objekt-Eigenschaften für Dialog und Modul

Die Import-Tabelle auf der Seite *„Spezifisch“* für die Objektklassen ***Module*** und ***Dialog*** erlaubt in der Spalte *„Label/Usepath“* zusätzlich die Eingabe eines Use-Pfades. Zeilen ohne einen Interface-Dateinamen werden als Imports per **use** behandelt und erlaubten keine Angabe von *„Application“*, *„Load“* und *„Static“*.



19.13.14.6 Spezifische Objekt-Eigenschaften für use (Import-Objekt)

Diese Seite besitzt nun eine Anzeige für den Use-Pfad. Dieser kann und darf nicht geändert werden. Änderungen sollten durch Zerstören und Anlegen erfolgen.

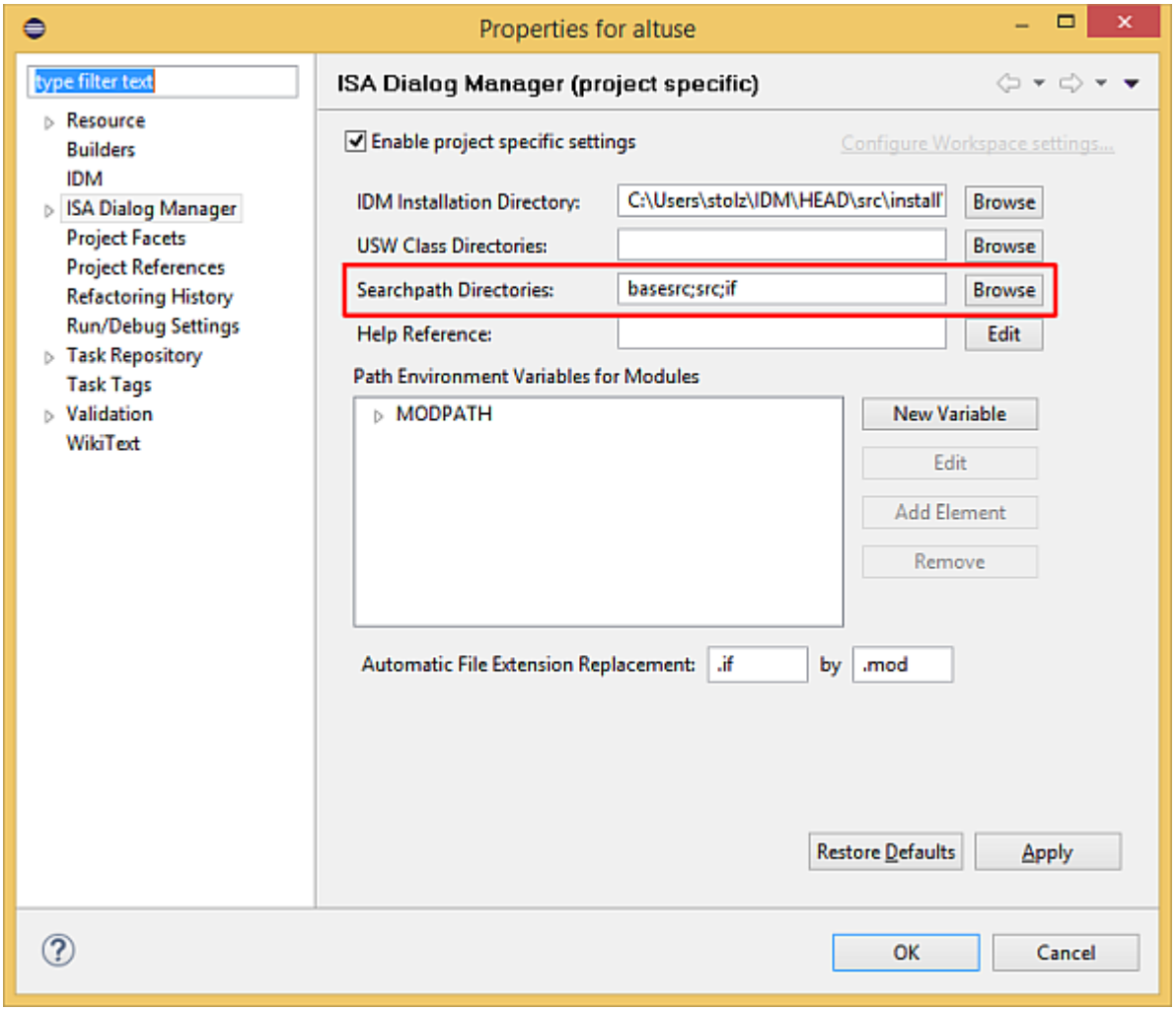


19.13.15 Ergänzungen am IDM-Eclipse-Plugin

19.13.15.1 Einstellungen (Preferences und Project-Properties)

Die folgende Einstellung ist hinzugekommen:

„Searchpath Directories“ – Definiert den Suchpfad für Interface- und Moduldateien die über „use“ angezogen werden. Auswählbar sind Verzeichnisse, die im Projekt liegen.



19.13.15.2 Informationsspeicher

Die Struktur der Informationsdateien hat sich geändert und der Strukturparser wie auch das Vervollständigen wurden um die Behandlung von „use“ zum Importieren von Modulen ergänzt.

19.13.15.3 IDM-Applikations-Starter

Der Starter berücksichtigt die Einstellung *„Searchpath Directories“* bei der Zusammenstellung der Argumente und gibt diese über **-IDMsearchpath** weiter.

19.13.15.4 Strukturansicht

Werden Module über „use“ importiert, so sind diese in der Strukturansicht (*„Outline View“*)mit dem Symbol  dargestellt.

19.14 Reihenfolge beim Initialisieren und Beenden von Modulen

Mit der IDM-Version A.06.02.g ergeben sich Änderungen bzw. Konsistenzanpassungen bei der Modul-Initialisierung und beim Beenden von Modulen.

1. Der **:init**-Aufruf für Objekte und Modelle aus einem importierten Modul passiert vor dem **:init**-Aufruf für Objekte im importierenden Modul oder Dialog. Objekte sind du diesem Zeitpunkt noch nicht sichtbar geschaltet.
2. Der **:clean**-Aufruf für Objekte und Modelle aus einem importierten Modul passiert nach dem **:clean**-Aufruf für Objekte im importierenden Modul oder Dialog.
3. Die `start`-Ereignisregeln eines Moduls oder Dialogs werden nach dem **:init**-Aufruf des Moduls ausgeführt. Objekte sind zu dieser Zeit schon sichtbar gemacht. Die `start`-Regeln eines importierten Moduls werden vor den `start`-Regeln des importierenden Moduls oder Dialogs aufgerufen.
4. Die `finish`-Ereignisregeln eines Moduls oder Dialogs werden vor dem **:clean**-Aufruf des Moduls ausgeführt. Objekte werden nach den `finish`-Regeln unsichtbar gemacht. Die `finish`-Regeln eines importierten Moduls werden vor den `finish`-Regeln des importierenden Moduls oder Dialogs aufgerufen.

19.15 Reguläre Ausdrücke

19.15.1 PCRE-Bibliothek zur Unterstützung Regulärer Ausdrücke

Um Reguläre Ausdrücke über die eingebaute-Funktion **regex()** oder als Format im IDM zu verwenden, ist die freie Bibliothek PCRE (Perl Compatible Regular Expression, siehe hierzu auch **www.pcre.org**) notwendig. Falls Sie diese Funktion also in ihrem Produkt einsetzen, sollten Sie die Lizenzbedingungen von PCRE beachten.

Der IDM setzt eine PCRE-Bibliothek ab der Version *3* mit aktivierter Unicode-Unterstützung und mit der „normalen“ PCRE-Schnittstelle voraus. Die mit PCRE-Version *10* eingeführte PCRE2-Schnittstelle wird noch nicht unterstützt. Empfohlen wird die **aktuell stabile Version *8.**** der PCRE-Bibliothek. Typischerweise sind die meisten aktuellen Linux-Distributionen schon von Haus aus mit der PCRE-Bibliothek ausgestattet oder erlauben eine problemlose Nachinstallation. Für die Nutzung unter Windows bietet sich neben der eigenen Kompilierung auch der Download einer vorkompilierten Bibliothek, z.B. von **www.pcre.org** oder **www.airesoft.co.uk** an.

Wichtig

Je nach Version ist immer mit einem unterschiedlichem Funktionsumfang und Fehlerstand der PCRE-Bibliothek zu rechnen. Es ist zu beachten, dass die ISA keine Gewährleistung auf die PCRE-Bibliothek und ihre Funktionen geben kann.

Die PCRE-Bibliothek wird normalerweise dynamisch angebunden indem die Funktionen **pcre_compile**, **pcre_study**, **pcre_exec**, **pcre_version** und **pcre_free** gesucht werden. Der IDM übergibt die

Strings in einer UTF8-Kodierung weshalb die angebundene PCRE-Bibliothek auch mit UTF8-Support ausgestattet sein sollte.

Folgende Anbindungsarten und damit verbundenen Suchreihenfolgen lässt der IDM zu:

Anbindungsart		Windows	Unix/Linux[1]
E	Funktionssuche direkt im Executable		
A	Applikationsbezogen (relativ zum Pfad der Anwendung)	pcre3.dll dll\pcre3.dll pcre.dll dll\pcre.dll	pcre.(so|sl) lib/pcre.(so|sl) ../lib/pcre. (so|sl)[2]
S	Systemspezifische Library-Suche (z.B. über die Pfadvariablen PATH oder LD_ LIBRARY_PATH)	pcre3.dll pcre.dll	pcre.(so|sl)

Bei Erstellung einer Anwendung mit den IDM-Bibliotheken erfolgt der Versuch einer Anbindung in der Reihenfolge E – A – S. Damit ist die einfachste Art, die eigene IDM-Anwendung mit der Unterstützung für Reguläre Ausdrücke auszustatten, das Hinzulegen der dynamischen PCRE-Bibliothek neben dem Executable. Andernfalls wird die im System befindliche Bibliothek angezogen.

Die von der ISA ausgelieferten IDM-Anwendungen zum Entwickeln und Simulieren (IDM, RIDM*, IDMED und Debugger) haben die PCRE-Bibliothek schon statisch eingebaut und benutzen die Anbindungssuche A – E – S, sodass die Nutzung einer externen PCRE-Bibliothek damit ebenso möglich ist.

Falls in der eigenen IDM-Anwendung ebenso eine statische Anbindung erwünscht ist, sollte folgendes beachtet werden: Falls ein Linken der Anwendung ohne Referenzierung der PCRE-Funktionen erfolgt, muss diese zwangsweise komplett angezogen werden (typische Linkoptionen sind z.B. **--whole-archive**, **+forceload** oder **/opt:notref**) und dafür gesorgt werden, dass die PCRE-Funktionen über die systemspezifische Funktionspointersuche gefunden wird (dies macht unter Umständen ein „Exportieren“ der Funktionen von der Anwendung notwendig). Die ISA empfiehlt aber die Anbindung über eine externe Bibliothek (DLL, Shared Library), sodass für die eigene Produktauslieferung ein Austausch der PCRE-Fassung erleichtert wird.

Die Anbindungsreihenfolge der PCRE-Bibliothek lässt sich vom Anwendungsprogrammierer durch die Schnittstellenfunktion **DM_Control()** bzw. **DM_ControlEx()** kontrollieren. Dies erfolgt über die neue Aktion *DMF_PCREBinding*. Folgende Einstellmöglichkeiten über den *options*-Parameter sind dabei möglich:

---

[1]Entsprechend der jeweiligen Plattform wird die Endung „.so“ oder „.sl“ für den Dateinamen verwendet.

[2]Die Suche über *../lib/* erfolgt nur wenn die Anwendung in einem Verzeichnis mit dem Namen „bin“ liegt.

action	options	Bedeutung
DMF_PCREBinding	*0*	Deaktiviert die Anbindung an die PCRE-Bibliothek, Reguläre Ausdrücke sind somit nicht mehr möglich.
DMF_PCREBinding	*1*	Anbindung an statisch im Executable vorhandene PCRE-Funktionen (Anbindungsart E).
DMF_PCREBinding	*2*	Nur dynamische Anbindung von PCRE-Bibliotheken relativ zur Applikation (Anbindungsart A).
DMF_PCREBinding	*3*	Bindung zu PCRE-Bibliotheken relativ zur Applikation oder aus dem System (Anbindungsreihenfolge A – S).
DMF_PCREBinding	*4*	Bindung mit Vorrang für Funktionen im Executable (Anbindungsreihenfolge E – A – S), dies ist für selbstgebaute IDM-Applikationen der Standardfall.
DMF_PCREBinding	*5*	Bindung analog zu *4* aber in umgekehrter Reihenfolge, also Vorrang für die im System installierte PCRE-Bibliothek (Anbindungsreihenfolge S – A – E).

19.15.2 Neue eingebaute Funktion regex()

Mit dieser Funktion kann ein Regulärer Ausdruck auf einen String oder eine Liste von Strings angewendet werden.

Um einen möglichst hohen Funktionsumfang zu ermöglichen, bindet der IDM hierzu die freie PCRE-Bibliothek dynamisch ein. Damit sind Musterausdrücke analog zu PERL möglich. Falls Sie diese Funktion also in ihrem Produkt einsetzen, sollten Sie Lizenzbedingungen und Dokumentation von PCRE (**www.pcre.org**) beachten. Bezüglich der Anbindung ist Kapitel „PCRE-Bibliothek zur Unterstützung Regulärer Ausdrücke“ zu berücksichtigen. Die ISA empfiehlt eine PCRE-Bibliothek in der Version *8.** für ein ordnungsgemäßes Funktionieren. Die PCRE-Funktionen werden dabei für „Pattern-Matching“ und „Capturing“ von String-Teilen benutzt. Der IDM übernimmt dann Auswertung und Ersetzung.

Syntaktisch werden folgende Reguläre Ausdrücke vom IDM akzeptiert um so die Operationen „Matching“ und „Substitution“ zu ermöglichen:

» Matching

```
m/<pattern>/<modifiers>
```

» Substitution

```
s/<pattern>/<replacement>/<modifiers>
```

Für die Konfiguration der Operation sind folgende Modifikatoren (*<modifiers>*) erlaubt:

s	single-line string (PCRE_DOTALL)
m	multi-line (PCRE_MULTILINE)
i	ignore case (PCRE_CASELESS)
g	global search – Mustererkennung wird wiederholt
x	extended (PCRE_EXTENDED)
f	first line (PCRE_FIRSTLINE)
W	unicode (wide) character classes (PCRE_UCP)
X	extra (PCRE_EXTRA)
U	ungreedy (PCRE_UNGREEDY)

In Klammern steht dabei, wie die Modifikatoren an PCRE weitergereicht werden. Deren Anwendung erfolgt also nicht durch den IDM sondern durch die PCRE-Bibliothek.

Ohne Fehlermeldung überlesen werden die Modifikatoren „o“ und „e“.

Die eigentliche Anwendung von *<pattern>* und *<replacement>* erfolgt dann mit der PCRE-Bibliothek. Weitere Details und Informationen zu den Regulären Ausdrücken sind deshalb der Dokumentation der verwendeten PCRE-Bibliothek zu entnehmen.

Der IDM erlaubt auch die Aufteilung der Bestandteile *<pattern>* und *<replacement>* des Regulären Ausdrucks in zwei getrennte Parameter, wobei dann allerdings keine Modifikatoren *<modifiers>* möglich sind. Normalerweise bestimmt die Operation die Art der zurückgegebenen Ergebnisse. Durch einen *Action*-Parameter kann dies aber gesteuert werden um so z.B. nur die Anzahl der gefundenen „Matches“ zu liefern, eine Filterung zu erreichen oder um die Werte der *<pattern>*-Variablen zu erhalten.

Folgende Aktionen mit entsprechenden Auswertungen sind möglich:

Tabelle 1: *Aktionen, Rückgabetypen und Auswertungen der regex-Funktion*

Aktion	Rückgabetyp	Auswertung
regex_eval	*boolean* *string* *list[string]*	Je nach Operation wird entweder zurückgeliefert, ob wenigsten einer der Strings dem Regulären Ausdruck entspricht (Matching-Operation *true* oder *false*). Oder im Substitution-Fall werden die ersetzten String(s) zurückgeliefert, falls das Muster übereinstimmt oder eben der Orginal-String.

Aktion	Rückgabetyp	Auswertung
regex_ match	*boolean list[string]*	Führt nur eine Prüfung des Musters durch und liefert *true* wenn das Muster übereinstimmt, ansonsten *false*. Bei einer String-Liste werden in die Resultatsliste nur Strings aufgenommen, welche das Muster erfüllen.
regex_ unmatch	*boolean list[string]*	Führt nur eine Prüfung des Musters durch und liefert *false* wenn das Muster übereinstimmt, ansonsten *true*. Bei einer String-Liste werden in die Resultatsliste nur Strings aufgenommen, welche das Muster **nicht** erfüllen
regex_ count	*integer*	Zählt die Anzahl gefundener Muster im String bzw. der String-Liste. Pro String wird maximal *+1* gezählt, sodass bei der Anwendung auf einen Einzel-String als Wert *0* oder *1* herauskommt und bei einer String-Liste ein Wert im Bereich *0 … itemcount()*.
regex_ locate	*integer list[integer]*	Liefert die Zeichenposition der ersten Übereinstimmung bei der Anwendung auf einen Einzel-String. Für eine String-Liste werden die Indexpositionen in der Liste geliefert, bei denen das Mustern erfüllt ist.

Der IDM erlaubt die Anwendung der **regex**-Funktion auf jeden beliebigen Sammlungsdatentyp (*list*, *hash*, *matrix*, *vector*). Eine generierte Rückgabeliste ist aber immer vom Typ *list* ohne Übernahme einer besonderen Indizierung der Ausgangsliste. Vor der Anwendung des Regulären Ausdrucks werden Werte, die nicht vom Datentyp *string* sind, in einen String umgewandelt, wie es auch z.B. bei *print <Value>;* geschieht.

Definition

```
anyvalue regex
(
      anyvalue StringOrList input,
      string   Pattern input
  { , string   Replace input }
  { , enum     Action  := regex_eval input }
)
```

Parameter

anyvalue *StringOrList* input

Dieser Parameter beinhaltet den String bzw. die Liste von Strings auf die der Reguläre Ausdruck angewendet wird.

string *Pattern* input

Dieser Parameter beinhaltet entweder den Regulären Ausdruck oder den Muster-String (*<pattern>*), falls der Reguläre Ausdruck in *<pattern>* und *<replacement>* aufgeteilt wird.

string ***Replace*** *input*

Dieser optionale Parameter sollte den Ersetzungs-String (*<replacement>*) beinhalten. wenn für den Funktionsaufruf der Reguläre Ausdruck in *<pattern>* und *<replacement>* aufgetrennt wird.

enum ***Action*** *:= regex_eval input*

Dieser optionale Parameter steuert die Ergebnisauswertung. Folgende Aktionen sind dabei möglich:

regex_eval (Standard)
: Evaluation des Regulären Ausdrucks („Matching“ oder „Substituition“).

regex_match
: Filterung auf Strings, die dem Muster entsprechen.

regex_unmatch
: Filterung auf Strings, die dem Muster **nicht** entsprechen.

regex_count
: Anzahl der gefundenen Muster (*+1* pro Suchstring).

regex_vars
: Liefert den Inhalt aller vom Suchmuster definierten Variablen.

regex_locate
: Liefert die String-Position bzw. Indexposition des gefundenen Musters.

Rückgabewert

Rückgabewert und -typ sind abhängig von der Auswertungsaktion und in „Tabelle 1“ (oben) beschrieben.

Beispiele

1. Auf Ziffern in einem String prüfen

```
print regex("Ist 127 eine Zahl?", "\\d+");
```

Ausgabe

```
true
```

2. Alle Dezimalzahlen durch ein „N“ ersetzen

```
print regex("42 ist größer als 10", "s/(\d+)/N/g");
```

Ausgabe

```
"N ist größer als N"
```

3. Nur Strings ausgeben auf die das Muster passt

```
print regex("127,5", "^\\d+,\\d+$", regex_match);
print regex("1275", "^\\d+,\\d+$", regex_match);
print regex(["3,7", "17,5", "0", "21,03"], "^\\d+,\\d+$", regex_match);
```

Ausgabe

```
"127,5"
""
["3,7","17,5","21,03"]
```

4. Mehrere reguläre Ausdrücke auf eine Liste anwenden
   - » nur Werte auflisten die ein Wort beinhalten
   - » die Anzahl der Wörter zählen
   - » jeden Eintrag in „>> <<“ einrahmen
   - » alle Geburtsjahre auflisten

```
variable list BirthDays := ["12-13-1973", "Ina", "1-7-1965", "Udo"];
print regex(BirthDays, "^\\w+$", regex_match);
print regex(BirthDays, "^\\w+$", regex_count);
print regex(BirthDays, "s/(.*)/>> $1 <</", regex_match);
print regex(BirthDays, "s/\\d+-\\d+-(\\d+)/$1/", regex_match);
```

Ausgabe

```
["Ina","Udo"]
2
[">> 12-13-1973 <<",">> Ina <<",">> 1-7-1965 <<",">> Udo <<"]
["1973","1965"]
```

5. Auflistung der Variablenwerte die in einem Regulären Ausdruck enthalten sind

```
print regex("+2500 Euro oder mehr", "(\\d+)\\s+(\\w+)", regex_vars);
```

Ausgabe

```
["2500 Euro","2500","Euro"]
```

6. Auflistung der Zugriffsindizes für die gefundenen Muster in einer Liste oder einem String

```
variable list Locales := [ "de_DE.UTF8", "C.UTF-8", "de_AT.utf8",
                           "en_AU.utf8", "en_ZM", "POSIX", "de_CH.uf8" ];
print regex(Locales, "/^de_/", regex_locate);
print regex("Hallo Welt", "/W/", regex_locate);
```

Ausgabe

```
[1,3,7]
7
```

7. Ausnutzung der automatischen Konvertierung von Werten in einer Liste zu *string*-Werten

```
record Rec4711 {}
print regex([123, winsys_x11, "Bond 007", opt_w2kprefsize_compat,
Rec4711],
            "s/\\d+/N/g");
```

Ausgabe

```
["N","winsys_xN","Bond N","opt_wNkprefsize_compat","RecN"]
```

19.15.3 Reguläre Ausdrücke als Format

Zusätzlich zu den bisherigen Formatbeschreibungen Leerstring, Eingabemuster, verdeckte Formatierung und numerisches Format kann nun auch ein Regulärer Ausdruck als Format angegeben werden. Erlaubt ist dabei aber nur ein Matching-Ausdruck in der Form `"/ … /"` um so die Prüfung des Inhalts bzw. der Eingabe zu erlauben.

Passt der Inhalt nicht zum Muster, so wird ein Leerstring dargestellt bzw. die Eingabe unterbunden.

Beispiele

- `format FmtKommaZahl "/^(\\d+(\\.\\d*)?)?$/";`
  Erlaubte Strings z.B. *""*, *"1"*, *"07654.321"*
- `format FmtHexZahl "/^[abcdefABCDEF0123456789]*$/";`
  Erlaubte Strings z.B. *""*, *"AF4513c"*, *"a"*, *"7EF"*
- `format FmtSpecial "/^([[:alpha:]]*\n\\d*\n[01]*)?$/";`
  Erlaubte Strings z.B. *""*, *"Stadt\n999\n01"*, *"Frankfurt\n60311\n01101110"*

19.15.4 Ergänzung der Installation

Am Anfang der Installation (mit **install.sh**) werden die Lizenzbedingungen für im IDM eingebaute Fremdprodukte bzw. Bibliotheken (z.B. PCRE) angezeigt. Erst nach Akzeptieren dieser Bedingungen erfolgt die Installation des IDM. Das explizite Anzeigen und Fragen kann bei über die zusätzliche Installationsoption **-y** (akzeptiert alle angefragten Lizenzbedingungen) unterbunden werden.

19.16 Asynchroner Methodenaufruf

Mit der neuen eingebauten Funktion **sendmethod()** und C-Schnittstellenfunktion **DM_SendMethod()** können Methodenaufrufe in die Ereignis-Warteschlange gestellt werden und asynchron aus der Ereignisschleife heraus ausgeführt werden. Die Funktionen sind damit eine einfachere Alternative zum Verschicken eines externen Ereignisses mit **sendevent()** bzw. **DM_SendEvent()** und Aufrufen der Methode in der Ereignisregel für dieses externe Ereignis.

19.16.1 Eingebaute Funktion sendmethod()

Mit dieser Funktion wird ein Methodenaufruf in die Ereignis-Warteschlange gestellt und asynchron aus der Ereignisschleife (DM_EventLoop) heraus ausgeführt. Die Funktion ist damit eine einfachere Alternative zum Verschicken eines externen Ereignisses mit **sendevent()** und Aufrufen der Methode in der Ereignisregel für dieses externe Ereignis.

sendmethod() unterstützt maximal *14* Argumente für den Methodenaufruf und kann nicht für Methoden mit `output`-Parametern verwendet werden.

Rückgabewerte von Methoden können nicht verarbeitet werden.

Definition

```
void sendmethod
(
      object   Object input,
      method   Method input
  { , anyvalue Arg1   input
    ...
    , anyvalue Arg14  input }
)
```

Parameter

object *Object* input
Objekt dessen Methode asynchron aufgerufen werden soll.

method *Method* input
Identifikator der aufzurufenden Methode.

*anyvalue **Arg1** input*
…
*anyvalue **Arg14** input*
Argumente die der Methode übergeben werden.

19.16.2 C-Schnittstellenfunktion DM_SendMethod()

Mit dieser Funktion wird ein Methodenaufruf in die Ereignis-Warteschlange gestellt und asynchron aus der Ereignisschleife (DM_EventLoop) heraus ausgeführt. Die Funktion ist damit eine einfachere Alternative zum Verschicken eines externen Ereignisses mit **DM_SendEvent()** und Aufrufen der Methode in der Ereignisregel für dieses externe Ereignis.

DM_SendMethod() unterstützt maximal *14* Argumente für den Methodenaufruf und kann nicht für Methoden mit `output`-Parametern verwendet werden.

Rückgabewerte von Methoden können nicht verarbeitet werden.

Definition

```
DM_Boolean DML_default DM_EXPORT DM_SendMethod
(
  DM_ID      object,
  DM_Method  method,
  DM_UInt    argc,
  DM_Value  *argv,
  DM_Options options
)
```

Parameter

-> DM_ID *object*

Objekt dessen Methode asynchron aufgerufen werden soll.

-> DM_Method *Method*

Identifikator der aufzurufenden Methode.

-> DM_Int argc

In diesem Parameter wird die Anzahl der Argumente für den Methodenaufruf (bis zu *14*) übergeben.

-> DM_Value *argv

Mit Hilfe dieses Parameters werden die Argumente (bis zu *14*) angegeben, die beim Methodenaufruf vom IDM mit übergeben werden sollen. Dieser Vektor muss dabei die in dem Parameter *argc* angegebene Länge haben.

-> DM_Options options

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

Rückgabewert

DM_TRUE	Methodenaufruf wurde in die Ereignis-Warteschlange gestellt.
DM_FALSE	Methodenaufruf konnte nicht in die Ereignis-Warteschlange gestellt werden.

19.17 Laden und Speichern von Konfigurationsdateien (profiles)

Die Funktionen zur Handhabung von Konfigurationsdateien (profiles) wurden vervollständigt. Mit der C-Schnittstellenfunktion **DM_SaveProfile()** können nun die aktuellen Werte von konfigurierbaren Records und Variablen in einer Konfigurationsdatei gespeichert werden. Mit den eingebauten Funktionen **loadprofile()** und **saveprofile()** ist das Laden und Speichern von Konfigurationsdateien nun auch aus der Regelsprache heraus möglich.

19.17.1 C-Schnittstellenfunktion DM_SaveProfile()

Diese Funktion schreibt die aktuellen Werte aller konfigurierbaren ***Record***-Instanzen (*.configurable = true*) und **globalen Variablen** (deklariert mit **config**) eines Dialogs oder Moduls in eine Konfigurationsdatei (profile), aus der sie mit der Funktion **DM_LoadProfile()** wieder geladen werden können.

Bei ***Records*** werden standardmäßig nur Werte, die nicht geerbt sind, in die Datei geschrieben. Um auch die geerbten Werte in die Datei zu schreiben ist der Parameter *options* auf *DMF_SaveAll* zu setzen.

Es werden nur Werte aus dem angegebenen Dialog oder Modul gespeichert. Aus anderen Modulen importierte ***Records*** und Variablen werden nicht berücksichtigt.

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

Parameter

-> DM_String filename

Dieser Parameter definiert den Dateinamen der Konfigurationsdatei. Es kann ein Dateipfad angegeben werden, der auch eine Umgebungsvariable enthalten darf.

-> DM_ID dialog

Dieser Parameter enthält den Identifikator des Dialogs oder Moduls, dessen ***Record***- und Variablenwerte in die Datei geschrieben werden sollen.

-> DM_String comment

In diesem Parameter kann ein Text angegeben werden, der als Kommentar in die Konfigurationsdatei geschrieben wird.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_SaveAll	Schreibt zusätzlich die geerbten Werte in die Konfigurationsdatei.

Rückgabewert

DM_TRUE Das Speichern der Werte in der Konfigurationsdatei war erfolgreich.

DM_FALSE	Die Werte konnten nicht gespeichert werden. Dies kann an Fehlern beim Zugriff auf die Datei oder einer ungültigen Modul-ID liegen.

19.17.2 Eingebaute Funktion loadprofile()

Diese Funktion liest die Werte der konfigurierbaren ***Record***-Instanzen (*.configurable* = *true*) und **globalen Variablen** (deklariert mit **config**) eines Dialogs oder Moduls aus einer Konfigurationsdatei (profile) ein.

Definition

```
boolean loadprofile
(
      string Filename  input
  { , object Module := null input }
)
```

Parameter

string *Filename* input

Dieser Parameter definiert den Dateinamen der Konfigurationsdatei. Es kann ein Dateipfad angegeben werden, der auch eine Umgebungsvariable enthalten darf.

object ***Module*** *:= null input*

Dieser optionale Parameter enthält den Identifikator des Dialogs oder Moduls, dessen ***Record***- und Variablenwerte aus der Datei eingelesen werden sollen.
Bei *Module* = *null* wird die Modul-ID der aktuellen Regel verwendet.

Rückgabewert

true	Das Einlesen der Werte in der Konfigurationsdatei war erfolgreich.
false	Die Werte konnten nicht eingelesen werden. Dies kann an Fehlern beim Zugriff auf die Datei oder einer ungültigen Modul-ID liegen.

19.17.3 Eingebaute Funktion saveprofile()

Diese Funktion schreibt die aktuellen Werte aller konfigurierbaren ***Record***-Instanzen (*.configurable* = *true*) und **globalen Variablen** (deklariert mit **config**) eines Dialogs oder Moduls in eine Konfigurationsdatei (profile), aus der sie mit der Funktion **loadprofile()** wieder geladen werden können.

Bei ***Records*** werden standardmäßig nur Werte, die nicht geerbt sind, in die Datei geschrieben. Um auch die geerbten Werte in die Datei zu schreiben ist der Parameter *All* auf *true* zu setzen.

Es werden nur Werte aus dem angegebenen Dialog oder Modul gespeichert. Aus anderen Modulen importierte ***Records*** und Variablen werden nicht berücksichtigt.

Definition

```
boolean saveprofile
(
      string  Filename   input
  { , object  Module  := null input }
  { , string  Comment := ""   input }
  { , boolean All :=  false   input }
)
```

Parameter

string *Filename* input

Dieser Parameter definiert den Dateinamen der Konfigurationsdatei. Es kann ein Dateipfad angegeben werden, der auch eine Umgebungsvariable enthalten darf.

*object **Module** := null input*

Dieser optionale Parameter enthält den Identifikator des Dialogs oder Moduls, dessen ***Record***- und Variablenwerte in die Datei geschrieben werden sollen.
Bei *Module* = *null* wird die Modul-ID der aktuellen Regel verwendet.

*string **Comment** := "" input*

In diesem optionalen Parameter kann ein Text angegeben werden, der als Kommentar in die Konfigurationsdatei geschrieben wird.

*boolean **All** := false input*

Wird dieser optionale Parameter auf *true* gesetzt, werden zusätzlich die geerbten Werte in die Konfigurationsdatei geschrieben. Beim Standardwert *false* werden nur nicht geerbte Werte gespeichert.

Rückgabewert

true	Das Speichern der Werte in der Konfigurationsdatei war erfolgreich.
false	Die Werte konnten nicht gespeichert werden. Dies kann an Fehlern beim Zugriff auf die Datei oder einer ungültigen Modul-ID liegen.

19.18 Neue eingebaute Funktion queryhelp()

Ohne Argument bzw. mit *null* als Argument versetzt diese Funktion den IDM in den Kontexthilfe-Modus, in dem der nächste Mausklick auf ein Objekt ein *help*-Ereignis für dieses Objekt auslöst.

Wird beim Aufruf eine Objekt-ID übergeben, dann löst die Funktion ein *help*-Ereignis für dieses Objekt aus.

Definition

```
void queryhelp
(
  { object Id := null input }
)
```

Parameter

object ***Id*** *:= null input*

In diesem optionalen Parameter kann ein Objekt angegeben werden, für das ein *help*-Ereignis ausgelöst werden soll.

19.19 Neue eingebaute Funktionen für Sammlungen und Strings

19.19.1 append()

Mit dieser Funktion kann an eine Sammlung (Datentypen *hash*, *list*, *matrix*, *refvec* und *vector*) oder einen String ein neuer Wert einmal oder mehrmals angehängt werden.

Allgemein gilt, dass der Datentyp des (optionalen) neuen Wertes ein Skalar sein muss und zum Wertetyp der Sammlung passen muss.

Die Wiederholungszahl des Anhängens (*Count*-Parameter) muss >= *0* sein.

Im Fehlerfall wird der Funktionsaufruf mit einem `fail` abgebrochen.

Besonderheiten

- *hash*
  Da dieser Sammlungsdatentyp keinen definierten Indexbereich besitzt, erfolgt das Anhängen hinter dem letzten *integer*-Index. Es wird also von einem *hash* mit einer Indizierung von *1,2,3…* ausgegangen. Der letzte gesetzte Index vom Datentyp *integer* wird ermittelt und daran anschließend angehängt.
- *matrix*
  Das Anhängen in eine Matrix erfolgt immer zeilen- oder spaltenweise. Die Steuerung erfolgt über den *Dir*-Parameter. Standardmäßig (*dir_row*) werden neue Zeilen angefügt. Bei Angabe von *dir_column* werden neue Spalten angefügt.
- *refvec*
  Das Anhängen von *null*-Objekten ist nicht erlaubt. Falls die Objekt-Id schon vorhanden ist, erfolgt ein Umpositionieren an die letzte Position.
- *string*
  Der neue Wert wird am String-Ende angefügt.

» *Default-Werte*

Um die Indexposition für die neuen Werte zu ermitteln werden auch Default-Werte beachtet, außer es handelt sich um eine *hash*-Sammlung.

Definition

```
anyalue append
(
      anyvalue ListValue  input,
      anyvalue NewValue   input
  { , integer  Count := 1 input }
  { , enum     Dir   := dir_row input }
)
```

Parameter

anyvalue *ListValue* input

In diesem Parameter wird die Sammlung bzw. der String angegeben an die ein Wert angehängt werden soll.

anyvalue *NewValue* input

Dieser Parameter definiert den neu anzufügenden Wert. Der Wert muss zum Wertetyp der Sammlung passen.

*integer **Count** := 1 input*

In diesem optionalen Parameter wird die Anzahl angegeben, wie oft ein neuer Wert (bzw. wie viele Zeilen oder Spalten) angehängt werden soll. Erlaubt sind Werte >= *0*, wobei bei *0* kein Anhängen erfolgt. Standardmäßig wird der neue Wert einmal angehängt.

*enum **Dir** := dir_row input*

In diesem optionalen Parameter wird für Sammlungen vom Datentyp *matrix* definiert, ob neue Zeilen (*dir_row*) oder neue Spalten (*dir_column*) angehängt werden.

Rückgabewert

Die Funktion gibt die veränderte Sammlung bzw. den veränderten String zurück.

Beispiele

» Mehrfaches Anhängen an einen String

```
print append("hi", " ho", 3);
```

Ausgabe

```
"hi ho ho ho"
```

» Mehrfaches Anfügen an eine *list*-Sammlung

```
variable list L := [4711, "koeln"];
print append(L, true, 2);
```

Ausgabe

```
[4711,"koeln",true,true]
```

» Anfügen von zwei Zeilen an eine Matrix

```
variable matrix M := [ [0,0]=>"?", [1,1]=>"a", [1,2]=>"b",
                       [2,2]=>"c", [3,1]=>"end" ];
print append(M, "new", 2);
```

Ausgabe

```
[[1,1]=>"a",[1,2]=>"b",[2,1]=>"?",[2,2]=>"c",[3,1]=>"end",[3,2]=>"?",
[4,1]=>"new",[4,2]=>"new",[5,1]=>"new",[5,2]=>"new"]
```

» Anfügen an einen Hash mit gemischter Indizierung

```
variable hash H := [false=>.xleft, 31=>.width, 33=>.ytop, "x"=>1];
print append(H, .height);
```

Ausgabe

```
[false=>.xleft,31=>.width,33=>.ytop,34=>.height,"x"=>1]
```

» Anhängen an eine *vector*-Liste

```
variable vector[integer] V := [9, 8, 7];
print append(V, 6);
```

Ausgabe

```
[9,8,7,6]
```

19.19.2 delete()

Mit dieser Funktion können aus einer Sammlung (Datentypen *hash*, *list*, *matrix*, *refvec* und *vector*) einer oder mehrere Werte an einer vorgegebenen Position gelöscht werden. Dahinterliegende Werte werden nach vorne geschoben. Außerdem ist der Datentyp *string* erlaubt um Zeichen aus einem String zu entfernen.

Das Löschen der Werte erfolgt dabei an der angegebenen Position. Die Positionsangabe muss >= *1* sein, d.h. ein Löschen von Default-Werten an den Indizes *[0]* bzw. *[0,0]* ist mit dieser Funktion nicht möglich. Die größtmögliche erlaubte Position entspricht dem aktuellen *countof(ListValue)*.

Die Anzahl der zu löschenden Werte wird über den *Count*-Parameter angegeben und sollte >= *0* sein.

Im Fehlerfall wird der Funktionsaufruf mit einem `fail` abgebrochen.

Als fehlerhaft wird auch angesehen, wenn versucht wird mehr Werte bzw. Zeilen oder Spalten zu löschen als überhaupt vorhanden sind.

Besonderheiten

» *hash*
Da dieser Sammlungsdatentyp keinen definierten Indexbereich besitzt, erfolgt das Löschen bezogen auf die *integer*-Indizes. Es wird also von einem *hash* mit einer Indizierung von *1,2,3…* ausgegangen. Es gibt keine Beschränkungen hinsichtlich der größtmöglichen Indexposition.

» *matrix*
Das Löschen in einer Matrix erfolgt immer zeilen- oder spaltenweise. Die Steuerung erfolgt über den *Dir*-Parameter. Standardmäßig (*dir_row*) werden Zeilen gelöscht. Bei Angabe von *dir_column* werden Spalten gelöscht. Dies bedeutet auch, dass sich die Positionsangabe im ersten Fall auf eine Zeile und im zweiten Fall auf eine Spalte bezieht. Entsprechend verkleinert sich damit auch die Matrix.

» *string*
Hier bezieht sich die Position auf die Zeichenposition im String, d.h. der String wird als Array von Zeichen behandelt.

Definition

```
anyvalue delete
(
      anyvalue ListValue input
      integer  Pos   input
  { , integer  Count := 1 input }
  { , enum     Dir   := dir_row input }
)
```

Parameter

anyvalue *ListValue* input
In diesem Parameter wird die Sammlung bzw. der String angegeben aus der Werte gelöscht werden sollen.

integer *Pos* input
In diesem Parameter wird die Indexposition angegeben, ab der einschließlich das Löschen erfolgen soll. Erlaubt sind Werte >= *1*. Bei *matrix*-Sammlungen stellt *Pos* je nach *Dir*-Parameter entweder eine Zeilenposition (*dir_row*) oder eine Spaltenposition (*dir_column*) dar.

*integer **Count** := 1 input*
In diesem optionalen Parameter wird die Anzahl der Werte (bzw. Zeilen oder Spalten bei einer Matrix) vorgegeben, die zu löschen sind. Erlaubt sind Werte >= *0*, wobei bei *0* kein Löschen erfolgt. Durch die Anzahl sollte beim Löschen der mögliche Positionsbereich nicht überschritten werden.

enum ***Dir*** *:= dir_row input*

In diesem optionalen Parameter wird die Ausrichtung der Indexposition definiert, was für eine Matrix von Bedeutung ist. Dabei bedeutet der Standardwert *dir_row*, dass in der Matrix Zeilen gelöscht werden. Bei *dir_column* werden Spalten gelöscht.

Rückgabewert

Die Funktion gibt die veränderte Sammlung bzw. den veränderten String zurück.

Beispiele

» Mehrfaches Löschen von Werten aus einer Liste

```
variable list L := [17, "x", window, .xleft, null];
print delete(L, 2, 3);
```

Ausgabe

```
[17,null]
```

» Löschen einer Spalte aus einer Matrix

```
variable matrix M := [ [0,0]=>"?", [1,1]=>"a1", [1,2]=>"b1",
                       [2,2]=>"b2", [2,3]=>"c2" ];
M := delete(M, 2, dir_column);
print M;
print countof(M);
```

Ausgabe

```
[[1,1]=>"a1",[1,2]=>"?",[2,1]=>"?",[2,2]=>"c2"]
[2,2]
```

» Entfernen von zwei Zeichen aus einem String

```
print delete("hallo", 3, 2);
```

Ausgabe

```
"hao"
```

» Löschen von Elementen aus einem Hash mit *integer*-Indizierung

```
variable hash H := [2=>.xleft, 31=>.width, 33=>.ytop];
H := delete(H, 30, 3);
print H;
```

Ausgabe

```
[2=>.xleft,30=>.ytop]
```

» Löschen aus einer *vector*-Liste

```
variable vector[boolean] V := [true, false, false, false, true];
print delete(V, 2, 3);
```

Ausgabe

```
[true,true]
```

» Löschen der ersten beiden Elemente aus einer *refvec*-Liste

```
variable refvec R := [PbApply, PbCancel, PbRevert];
print delete(R, 1, 2);
```

Ausgabe

```
[pushbutton D.PbRevert]
```

19.19.3 exchange()

Mit dieser Funktion können zwei Werte einer Sammlung (Datentypen *hash*, *list*, *matrix*, *refvec* und *vector*) vertauscht werden. Außerdem ist der Datentyp *string* erlaubt um einzelne Zeichen innerhalb eines Strings zu vertauschen.

Die beiden Positionsangaben (*Pos1* und *Pos2*) geben die Indexpositionen an, deren Werte vertauscht werden. Die Positionsangaben müssen >= *1* sein und dürfen nicht über der möglichen Anzahl von Elementen in der Sammlung liegen.

Im Fehlerfall wird der Funktionsaufruf mit einem `fail` abgebrochen.

Besonderheiten

» *hash*
Da dieser Sammlungsdatentyp keinen definierten Indexbereich besitzt, erfolgt das Austauschen bezogen auf die *integer*-Indizes. Es wird also von einem *hash* mit einer Indizierung von *1,2,3…* ausgegangen.

» *matrix*
Das Austauschen in einer Matrix erfolgt immer zeilen- oder spaltenweise. Die Steuerung erfolgt über den *Dir*-Parameter. Standardmäßig (*dir_row*) werden ganze Zeilen getauscht. Bei Angabe von *dir_column* werden ganze Spalten getauscht. Dies bedeutet auch, dass sich die Positionsangabe im ersten Fall auf eine Zeile und im zweiten Fall auf eine Spalte bezieht.

» *string*
Hier bezieht sich die Position auf die Zeichenposition im String, d.h. der String wird als Array von Zeichen behandelt.

» *Default-Werte*
Die Sammlungsdatentypen *list*, *hash* und *matrix* erlauben Default-Werte. Der Austausch solcher geerbten Werte an gültigen Positionen wird konsistent durchgeführt.

Definition

```
anyvalue exchange
(
     anyvalue ListValue input,
     integer  Pos1 input,
     integer  Pos2 input
 { , enum     Dir  := dir_row input }
)
```

Parameter

anyvalue *ListValue* input

In diesem Parameter wird die Sammlung bzw. der String angegeben in der Werte ausgetauscht werden sollen.

integer *Pos1* input

In diesem Parameter wird die erste Indexposition angegeben, deren Wert mit dem der zweiten Indexposition (*Pos2*) vertauscht wird. Erlaubt sind Werte >= *1*. Bei *matrix*-Sammlungen stellt *Pos1* je nach *Dir*-Parameter entweder eine Zeilenposition (*dir_row*) oder eine Spaltenposition (*dir_column*) dar.

integer *Pos2* input

In diesem Parameter wird die zweite Indexposition angegeben, deren Wert mit dem der ersten Indexposition (*Pos1*) vertauscht wird. Erlaubt sind Werte >= *1*. Bei *matrix*-Sammlungen stellt *Pos2* je nach *Dir*-Parameter entweder eine Zeilenposition (*dir_row*) oder eine Spaltenposition (*dir_column*) dar.

*enum **Dir** := dir_row input*

In diesem optionalen Parameter wird die Ausrichtung der Indexposition definiert, was für eine Matrix von Bedeutung ist. Dabei bedeutet der Standardwert *dir_row*, dass in der Matrix Zeilen vertauscht werden. Bei *dir_column* werden Spalten vertauscht.

Rückgabewert

Die Funktion gibt die veränderte Sammlung bzw. den veränderten String zurück.

Beispiele

» Austauschen von Werten in einer Liste

```
variable list L := ["a", "c", "b", "d"];
print exchange(L, 2, 3);
```

Ausgabe

```
["a","b","c","d"]
```

» Austauschen von zwei Zeilen einer Matrix

```
variable matrix M := [ [0,0]=>"?", [1,1]=>"a", [1,2]=>"b",
                       [2,2]=>"end", [3,1]=>"c" ];
M := exchange(M, 2, 3);
print M;
```

Ausgabe

```
[[1,1]=>"a",[1,2]=>"b",[2,1]=>"c",[2,2]=>"?",[3,1]=>"?",[3,2]=>"end"]
```

» Austausch von zwei Zeichen in einem String

```
print exchange("X-Y", 1, 3);
```

Ausgabe

```
"Y-X"
```

» Austausch von zwei Elementen in einem Hash mit *integer*-Indizierung

```
variable hash H := [2=>.xleft, 31=>.width, 33=>.ytop];
print exchange(H, 31, 32);
```

Ausgabe

```
[2=>.xleft,32=>.width,33=>.ytop]
```

» Austausch von Werten in einer *vector*-Liste

```
variable vector[boolean] V := [true, false];
print exchange(V, 1, 2);
```

Ausgabe

```
[false,true]
```

» Austausch der letzten beiden Werte einer *refvec*-Liste

```
variable refvec R := [PbApply, PbCancel, PbRevert];
print exchange(R, 2, 3);
```

Ausgabe

```
[PbApply,PbRevert,PbCancel]
```

19.19.4 insert()

Mit dieser Funktion kann in einer Sammlung (Datentypen *hash*, *list*, *matrix*, *refvec* und *vector*) oder einem String ein neuer Wert einmal oder mehrmals eingefügt werden.

Das Einfügen der neuen Werte erfolgt dabei vor der angegebenen Position. Die Positionsangabe muss >= *1* sein, d.h. ein Einfügen von Default-Werten an den Indizes *[0]* bzw. *[0,0]* ist mit dieser

Funktion nicht möglich. Die größtmögliche erlaubte Position ist um *1* höher als der aktuelle *countof (ListValue)* um das Anhängen an eine bestehende Sammlung zu ermöglichen.

Allgemein gilt, dass der Datentyp des (optionalen) neuen Wertes ein Skalar sein muss und zum Wertetyp der Sammlung passen muss.

Die Wiederholungszahl des Einfügens (*Count*-Parameter) muss >= *0* sein.

Im Fehlerfall wird der Funktionsaufruf mit einem `fail` abgebrochen.

Besonderheiten

- *hash*
  Da dieser Sammlungsdatentyp keinen definierten Indexbereich besitzt, erfolgt das Einfügen bezogen auf die *integer*-Indizes. Es wird also von einem *hash* mit einer Indizierung von *1,2,3…* ausgegangen. Es gibt keine Beschränkungen hinsichtlich der größtmöglichen Indexposition.
- *matrix*
  Das Einfügen in eine Matrix erfolgt immer zeilen- oder spaltenweise. Die Steuerung erfolgt über den *Dir*-Parameter. Standardmäßig (*dir_row*) werden neue Zeilen eingefügt. Bei Angabe von *dir_column* werden neue Spalten eingefügt. Dies bedeutet auch, dass sich die Positionsangabe im ersten Fall auf eine Zeile und im zweiten Fall auf eine Spalte bezieht.
- *refvec*
  Das Einfügen von *null*-Objekten ist nicht erlaubt. Ein mehrfaches Einfügen wird eben so wenig unterstützt, da die gleiche Objekt-Id nur einmal vorhanden sein darf. Falls die Objekt-Id schon vorhanden ist, erfolgt ein Umpositionieren.
- *string*
  Hier bezieht sich die Position auf die Zeichenposition im String, d.h. der String wird als Array von Zeichen behandelt.
- *Default-Werte*
  Grundsätzlich macht es Sinn beim Einfügen einen neuen Wert anzugeben. Wird dieser weggelassen, so hängt es vom Sammlungstyp ab, welche Werte an den eingefügten Positionen stehen. Die Sammlungstypen *list*, *hash* und *matrix* erlauben Default-Werte, sodass beim Einfügen ohne neuen Wert die Default-Werte an den eingefügten Positionen erscheinen.

Definition

```
anyalue insert
(
      anyvalue ListValue  input,
      integer  Pos   input
  { , integer  Count := 1 input }
  { , enum     Dir   := dir_row input }
  { , anyvalue NewValue   input }
)
```

Parameter

anyvalue *ListValue* input

In diesem Parameter wird die Sammlung bzw. der String angegeben in die ein Wert eingefügt werden soll.

integer *Pos* input

In diesem Parameter wird die Indexposition angegeben, vor der das Einfügen erfolgen soll. Erlaubt sind Werte >= *1*. Bei *matrix*-Sammlungen stellt *Pos* je nach *Dir*-Parameter entweder eine Zeilenposition (*dir_row*) oder eine Spaltenposition (*dir_column*) dar.

*integer **Count** := 1 input*

In diesem optionalen Parameter wird die Anzahl angegeben, wie oft ein neuer Wert (bzw. wie viele Zeilen oder Spalten) eingefügt werden soll. Erlaubt sind Werte >= *0*, wobei bei *0* kein Einfügen erfolgt.

*enum **Dir** := dir_row input*

In diesem optionalen Parameter wird die Ausrichtung der Indexposition definiert, was für eine Matrix von Bedeutung ist. Dabei bedeutet der Standardwert *dir_row*, dass in der Matrix Zeilen eingefügt werden. Bei *dir_column* werden Spalten eingefügt.

*anyvalue **NewValue** input*

Dieser optionale Parameter definiert den neu einzufügenden Wert. Der Wert muss zum Wertetyp der Sammlung passen.

Rückgabewert

Die Funktion gibt die veränderte Sammlung bzw. den veränderten String zurück.

Beispiele

» Mehrfaches einfügen des Strings *"X"* in eine Liste

```
variable list L := ["a", "b", "c"];
print insert(L, 1, 2, "X");
```

Ausgabe

```
["X","X","a","b","c"]
```

» Anhängen einer Zeile an das Ende einer Matrix

```
variable matrix M := [ [0,0]=>"?", [1,1]=>"a1", [1,2]=>"b1",
                       [2,1]=>"a2", [2,2]=>"b2" ];
M := insert(M, 3, "NEW");
print M;
```

Ausgabe

```
[[1,1]=>"a1",[1,2]=>"b1",[2,1]=>"a2",[2,2]=>"b2",[3,1]=>"NEW",
[3,2]=>"NEW"]
```

Einfügen einer neuen 1. Spalte ohne Wert

```
M := insert(M, 1, dir_column);
print M;
```

Ausgabe

```
[[1,1]=>"?", [1,2]=>"a1",[1,3]=>"b1",[2,1]=>"?",[2,2]=>"a2",[2,3]=>"b2",
[3,1]=>"?",[3,2]=>"NEW",[3,3]=>"NEW"]
```

» Einfügen von *"ll"* in einen String

```
print insert("hao", 3, 2, "l");
```

Ausgabe

```
"hallo"
```

» Einfügen in ein Hash mit *integer*-Indizierung

```
variable hash H := [2=>.xleft, 33=>.ytop];
H := insert(H, 10, .width);
print H;
```

Ausgabe

```
[2=>.xleft,10=>.width,34=>.ytop]
```

» Einfügen in eine *vector*-Liste

```
variable vector[boolean] V := [true, true];
print insert(V, 2, 3, false);
```

Ausgabe

```
[true,false,false,false,true]
```

» Einfügen eines schon vorhandenen *refvec*-Elements an einer anderen Stelle

```
variable refvec R := [PbApply, PbCancel, PbRevert];
print insert(R, 1, PbRevert);
```

Ausgabe

```
[pushbutton D.PbRevert,pushbutton D.PbApply,pushbutton D.PbCancel]
```

19.20 Unterstützung von SSH und SSL im DDM

Unterstützung von SSL

Die Netzwerkschnittstelle unterstützt nun **SSL** für eine verschlüsselte Kommunikation zwischen Darstellungsseite (Client) und Anwendungsseite (Server).

Voraussetzung ist, dass auf dem System **OpenSSL** installiert ist. Wenn die dynamische Bibliothek **libssl** (Version 1) im Suchpfad liegt, lädt der IDM diese zur Laufzeit und bietet die SSL-Unterstützung an.

SSL unter Microsoft Windows

Es wird **OpenSSL** (Version 1.1) unterstützt. Weitere Informationen sind auf **www.openssl.org** zu finden, unter anderem auch Hinweise zu fertig gebauten Binaries von OpenSSL für Windows, zum Beispiel **slproweb.com/products/Win32OpenSSL.html**.

Unterstützung von SSH

Die Netzwerkschnittstelle unterstützt nun **SSH** für einen sicheren Start der Anwendungsseite.

Voraussetzung ist, dass auf dem System **libssh** installiert ist (Windows: Version 1, Unix: ab Version 0.9). Wenn die dynamische Bibliothek **libssh** im Suchpfad liegt, lädt der IDM diese zur Laufzeit und bietet die SSH-Unterstützung an.

SSH unter Microsoft Windows

Es wird **libssh** unterstützt. Weitere Informationen sind auf **www.libssh.org** zu finden, unter anderem auch Hinweise zur Windows-Version auf der Seite „Download“. Man muss **vcpkg** installieren (ist auf der Downloadseite verlinkt), darüber kann dann **libssh** installiert werden.

Der Name der dynamischen Bibliothek ist **ssh** (ohne „lib“).

Hinweis: Dabei werden DLLs für SSL installiert, die auch vom IDM verwendet werden können.

Netzwerkprotokoll inkompatibel

Der IDM unterstützt nun SSL zur Kommunikation. Außerdem kann beim Starten der Anwendungsseite mit *.exec* nun das **SSH-Protokoll** verwendet werden.

Durch diese Erweiterungen ist das **Netzwerkprotokoll inkompatibel** geworden. Das heißt es ist keine Verbindung zwischen IDM-Version A.06.01 (und niedriger) und IDM-Version A.06.02.g (und höher) möglich.

Achtung

Verwendet die Anwendungsseite SSL und versucht eine alte Darstellungsseite (IDM-Version A.06.01 und niedriger) sich zu verbinden, dann werden beide Seite aufeinander warten („Deadlock“).

19.20.1 Schlüssel und Zertifikate

Um SSH bzw. SSL zu verwenden, werden Schlüssel und Zertifikate benötigt. Eine Anleitung zum richtigen Aufsetzen der Umgebung würde über den Rahmen dieser Dokumentation weit hinausgehen.

Unter **www.openssl.ord/docs** wird auf entsprechende Literatur verwiesen.

Für einfache Tests kann folgendes Kommando verwendet werden:

```
openssl req -x509 -newkey rsa:4096 -keyout private.pem
  -out cert.pem -days 365 -nodes
```

Dies erzeugt die Zertifikatsdateien **cert.pem** und **private.pem** ohne „Pass Phrase“. Wenn beide im Verzeichnis der IDM-Anwendungsseite liegen, werden keine weiteren Startoptionen benötigt.

19.20.2 Erweiterungen am Application-Objekt

19.20.2.1 Erweiterungen bei vorhandenen Attributen

19.20.2.1.1 .connect

Um die Verbindung über **SSL** durchzuführen kann bei der Spezifikation der Verbindung über *.connect* das Schema *"ssl://"* vorangestellt werden (Beispiel `.connect "ssl://myserver:0815";`).

Anmerkung

Auch beim Attribut *.transport* kann das Schema *"ssl://"* angegeben werden. Wenn bei beiden Attributen ein Schema angegeben ist, muss es identisch sein. Ein einmal angegebenes Schema *"ssl://"* lässt sich nicht mehr abstellen.

Beispiel

Ohne SSL	Mit SSL
`application Appl {` `  .connect "localhost:4711";` `}`	`application Appl {` `  .connect "ssl://localhost:4711";` `}`

19.20.2.1.2 .transport

Um die Verbindung über **SSL** durchzuführen kann bei der Spezifikation des Protokolls über *.transport* das Schema *"ssl://"* vorangestellt werden (Beispiel `.transport "ssl://tcpip";`). Wird kein Protokoll vorgegeben kann „://“ entfallen (also `.transport "ssl“;`).

Anmerkung

Auch beim Attribut *.connect* kann das Schema *"ssl://"* angegeben werden. Wenn bei beiden Attributen ein Schema angegeben ist, muss es identisch sein. Ein einmal angegebenes Schema *"ssl://"* lässt sich nicht mehr abstellen.

Beispiel

Ohne SSL	Mit SSL
`application Appl1 {` `  .connect "localhost:4711";` `}` `application Appl2 {` `  .transport "tcpip-winsock";` `  .connect "localhost:4711";` `}`	`application Appl1 {` `  .transport "ssl";` `  .connect "localhost:4711";` `}` `application Appl2 {` `  .transport "ssl://tcpip-winsock";` `  .connect "localhost:4711";` `}`

19.20.2.1.3 .exec

Um die Anwendungsseite über **SSH** an Stelle von RSH zu starten, kann bei der Spezifikation des Kommandos das Schema *"ssh://"* vorangestellt werden (Beispiel `.exec "ssh://myserver:list";`).

Standardmäßig wird das RSH-Protokoll verwendet, wenn kein Sicherheitsschema beim *.transport*-Attribut angegeben wurde. Andernfalls wird das SSH-Protokoll verwendet. Soll von diesem Standard abgewichen werden, muss das gewünschte Schema beim *.exec*-Attribut angegeben werden. Hierbei wird mit *"ssh://"* das SSH-Protokoll und mit *"rsh://"* das RSH-Protokoll ausgewählt.

Anmerkungen

- Die für SSL notwendigen zusätzlichen Startoptionen müssen beim Kommando angegeben werden. Nur das Sicherheitsschema „ssl“ wird – falls verwendet – dem Kommando als zusätzliche Startoption `-IDMtransport ssl` hinzugefügt.
- Die Kommandozeile erhält automatisch die Startoption `-IDMtellport`. Wird als Kommando ein Skript oder ähnliches an Stelle einer IDM-Serveranwendung angegeben, dann muss von der Ausgabe der IDM-Serveranwendung zumindest die Zeile, die mit `-IDMport` beginnt, komplett (inklusive führender und folgender Zeilenumbrüche „\n“) weitergeleitet werden.

Beispiel

Ohne SSH	Mit SSH
`application Appl1 {` `  .exec"host%account%passwd:list";` `}`	`application Appl1 {` `  .exec "ssl://host%account%passwd:list";` `}`

Ohne SSH	Mit SSH und SSL
`application Appl2 {` `  .exec "host%account%passwd:list";` `}`	`application Appl2 {` `  .transport "ssl";` `  .exec "host%account%passwd:list";` `}`

19.20.2.2 Neue Attribute

19.20.2.2.1 .certificatefile

In diesem Attribut wird die Zertifikatsdatei definiert, die für eine SSL-Verbindung verwendet wird.

	Bezeichner		Datentyp
Regelsprache	**.certificatefile**		string
C	AT_certificatefile		DT_string
COBOL	AT-certificatefile		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	application		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Standardmäßig wird die Datei **cert.pem** aus dem Installationsverzeichnis der Anwendung verwendet.

19.20.2.2.2 .password

In diesem Attribut kann das Passwort definiert werden, das zum Starten der Anwendungsseite beim RSH- oder SSH-Protokoll verwendet wird.

	Bezeichner	Datentyp
Regelsprache	**.password**	string
C	AT_password	DT_string
COBOL	AT-password	DT-string
Klassifizierung	Objektspezifisches Attribut	
Objekte	application	

	Bezeichner		Datentyp
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Das Attribut wird ausgewertet, wenn beim Kommando im *.exec*-Attribut kein Passwort angegeben ist. Das Attribut stellt eine Alternative dar, wenn es Probleme bereitet, das Kommando im *.exec*-Attribut korrekt aufzubauen.

19.20.2.2.3 .username

In diesem Attribut kann der Benutzername definiert werden, das zum Starten der Anwendungsseite beim RSH- oder SSH-Protokoll verwendet wird.

	Bezeichner		Datentyp
Regelsprache	**.username**		string
C	AT_username		DT_string
COBOL	AT-username		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	application		
Zugriff	get, set	Standardwert	–
changed-Ereignis	ja	Vererbung	ja

Das Attribut wird ausgewertet, wenn beim Kommando im *.exec*-Attribut kein Benutzername angegeben ist. Das Attribut stellt eine Alternative dar, wenn es Probleme bereitet, das Kommando im *.exec*-Attribut korrekt aufzubauen.

19.20.3 Änderungen bei Startoptionen

-IDMtransport

(erweitert)

Um die Verbindung über SSL durchzuführen kann bei der Spezifikation des Protokolls über **-IDMtransport** das Schema *"ssl://"* vorangestellt werden (Beispiel `-IDMtransport "ssl://tcpip"`). Wird kein Protokoll vorgegeben kann „://“ entfallen (also `-IDMtransport "ssl"`).

Da die Darstellungsseite die beim *.exec*-Attribut angegebene Verschlüsselung mit **-IDMtransport** an die Kommandozeile anhängt, kann diese Option beim Starten der Anwendungsseite zweimal vorkommen.

-IDMconnect

(veraltet)

Die Startoption **-IDMconnect** wird nicht mehr unterstützt, da sich die Anwendungsseite nicht mehr zur Darstellungsseite verbindet. Diese Änderung wurde notwendig, um eine einheitliche Richtung des Verbindungsaufbaus von der Darstellungsseite zur Anwendungsseite zu erreichen.

-IDMtellport

(neu, intern)

Die Startoption **-IDMtellport** veranlasst die Anwendungsseite einen freien Port zu bestimmen und diesen auf STDOUT auszugeben.

Diese Startoption wird intern verwendet um der Darstellungsseite den Port mitzuteilen, auf den sie sich verbinden kann wenn die Anwendungsseite mit Hilfe des *.exec*-Attributs gestartet wird.

Diese Startoption ist nur auf der Anwendungsseite erlaubt.

Wichtig

Wird beim *.exec*-Attribut als Kommando ein Skript oder ähnliches an Stelle einer IDM-Serveranwendung angegeben, dann muss von der Ausgabe der IDM-Serveranwendung zumindest die Zeile, die mit `-IDMport` beginnt, komplett (inklusive führender und folgender Zeilenumbrüche „\n“) weitergeleitet werden.

-IDMssl_certfile <Dateiname>

(neu)

Mit der Startoption **-IDMssl_certfile <Dateiname>** wird der Anwendungsseite die Zertifikatsdatei mitgeteilt, welche für eine SSL-Verbindung verwendet wird.

Standardmäßig wird die Datei **cert.pem** aus dem Installationsverzeichnis der Anwendung verwendet.

Diese Startoption ist nur auf der Anwendungsseite erlaubt.

-IDMssl_privkeyfile <Dateiname>

(neu)

Mit der Startoption **-IDMssl_privkeyfile <Dateiname>** wird der Anwendungsseite der private Schlüssel mitgeteilt, welcher für eine SSL-Verbindung verwendet wird.

Standardmäßig wird die Datei **private.pem** aus dem Installationsverzeichnis der Anwendung verwendet.

Diese Startoption ist nur auf der Anwendungsseite erlaubt.

20 A.06.02.a – f

Mit diesen Versionsnummern gab es keine allgemein veröffentlichten A-Versionen sondern nur Vorab-Versionen (V-Versionen).

Alle Erweiterungen, Änderungen und Korrekturen dieser Versionen sind in den Release Notes zu Version A.06.02.g zusammengefasst.

21 A.06.01.h

21.1 Windows

IDM-12923: ***Acceleratoren***, die am Kontextmenü eines ***Image***-Objekts mit *.style = menubox* definiert sind, werden nun fensterweit ausgelöst, da ein solches ***Image***-Objekt dazu dient, ein Fenstermenü zu simulieren. Bisher wurden die ***Acceleratoren*** – wie bei den anderen Stilen – nur ausgelöst, wenn das ***Image***-Objekt den Fokus besaß.

IDM-12900 und IDM-12846: Es kommt nicht mehr zu Positionierungsfehlern des Kindobjekts einer ***Layoutbox***, wenn die Größe eines unten angebundenen Kindobjekts geändert (IDM-12846) oder ein Kindobjekt nachträglich sichtbar geschaltet wird (IDM-12900). Die Positionierungsfehler korrigierten sich auch bisher schon nach erneuter Berechnung der Kindgeometrien von alleine.

IDM-12898: Von sehr langen Einträgen war im ***Poptext*** nur der Anfang zu sehen. Über die Option *.options[opt_hscoll]* kann nun eine optionale horizontale Scrollbar aktiviert werden.

Neue Option am Poptext-Objekt unter Windows

Mit der Option *.options[opt_hscroll]* kann eine optionale horizontale Scrollbar an der Liste des ***Poptext***-Objekts aktiviert werden. Ist diese Option gesetzt, dann wird eine horizontale Scrollbar angezeigt, sobald es Einträge gibt, die nicht komplett sichtbar sind.

Hinweis

Die horizontale Scrollbar erscheint innerhalb des Darstellungsbereichs und überdeckt somit die untersten Zeilen. Es wird dafür dann auch eine vertikale Scrollbar eingeblendet. Da die Bedienung von Scrollbars in einer geöffneten Liste für den Anwender nicht einfach ist, sollte diese Option nur verwendet werden, wenn es keine andere Lösung gibt.

IDM-12891: Wenn ein ***Poptext*** mit *.style = edittext* einen leeren Eintrag in seiner Liste enthält, dann ist nicht mehr *thisevent.index = nothing* bei *select-* bzw. *activate*-Ereignissen, die durch die Maus ausgelöst werden. Das Problem lag daran, dass nach Leerstrings nicht gesucht werden kann. Dieser Fall wird jetzt speziell behandelt.

Behoben: Es kommt nicht mehr zu einem Abbruch *„Assfail in Module: object/slot Line: 3560“*, wenn ein *.statushelp*-Text in der ***Statusbar*** angezeigt wird und UI Automation aktiviert ist. Das Problem lag daran, dass am ***Statusbar***-Objekt die *.statushelp*-Information als Text gespeichert ist, während für alle anderen Bereiche die IDM Objekt-ID gespeichert war. Außerdem änderte das ***Statusbar***-Objekt die Indizierung der Bereiche dynamisch, je nachdem ob aktuell ein *.statushelp*-Text angezeigt wurde oder nicht. Dies ist aber für die UI Automation ungünstig. Jetzt bleibt die Indizierung statisch. Die Bereiche werden stattdessen sichtbar bzw. unsichtbar gesetzt.

21.2 Qt

IDM-12855: Bei einer ***Groupbox***, die eine **virtuelle Größe** besitzt, aber selbst größer ist als die virtuelle Größe, erzeugt ein Klick außerhalb des virtuellen Bereichs nun richtigerweise ein *select-*

Ereignis für die ***Groupbox*** und nicht für das Vaterobjekt. Analog zum Windows-Toolkit zählt der Rahmen nicht zum klickbaren Bereich dazu. Deshalb erfolgt nun eine Korrektur der Mauskoordinaten ausgehend vom Bereich in dem die Kindobjekte dargestellt werden. Es kommt somit zu keiner Überschreitung der Pixelkoordinaten in *thisevent.x* und *thisevent.y* bezüglich der realen Pixelgröße des Objekts.

IDM-12847: Die Behandlung der Tastatureingaben mit und ohne Formatbehandlung bei ***Edittext*** und ***Poptext*** wurde umgestellt. Ein Einfügen („paste“) über das Popup-Menü sollte nun den angezeigten Text nicht mehr ohne Formatprüfung einfach einfügen.

Hinweis

Im IDM FÜR QT erfolgen die Formatvalidierung und die *.maxchars*-Prüfung auf einem „Plain Text“, was bei langen Textinhalten oder HTML-Inhalten zu Leistungseinbußen führen kann. Wir empfehlen ohnehin für mehrzeilige ***Edittexte*** mit beliebig großen Inhalten von der Verwendung eines Formats abzusehen.

Achtung

Grundsätzlich wird für HTML-Inhalte (*.options[opt_html]* = *true*) von der Benutzung von Formaten dringend abgeraten! Da bei der HTML-Darstellung der formatierte String nicht erhalten bleibt (z.B. durch Wegoptimierung von Leerzeichen) wird damit die Konsistenz von Cursor-Positionen verletzt.

21.3 Unix

IDM-12884: Die Funktionen **DM_QueueExtevent()** und **DM_SendEvent()** sind nun auch unter UNIX/LINUX „thread safe“. Die notwendige interne Synchronisierung erfolgt mit einem Mutex der **pthread**-Bibliothek, die beim Linken nun erforderlich ist. Die **pthread**-Bibliothek wurde im SYSLIBS-Define in **MakeDefs** aufgenommen.

21.4 Kern

IDM-12929: Ein Speicherleck der eingebauten Funktionen **tolower()** und **toupper()** unter MICROSOFT WINDOWS ist nun behoben.

IDM-12916: Die dynamische Umsetzung von *.selstyle* an ***Listbox*** und ***Tablefield*** wird nun wieder vom WSI ausgeführt und erzeugt keine Warnung für das *.multisel*-Attribut mehr.

IDM-12859: Ein Problem wurde behoben, durch das am ***Notepage***-Objekt sporadisch Initialisierungsfehler des *.index*-Wertes beim *changed*-Ereignis für *.active* auftraten.

PLUGIN-12330: Das Schlüsselwort **reexport** ist nun in der Sprachdefinitionsdatei des IDM zur Verwendung im IDM ECLIPSE PLUGIN enthalten.

21.5 DDM (Netz)

IDM-12856: Es kommt nicht mehr zu Protokollfehlern, wenn Funktionen sowohl vom Server als auch lokal angemeldet werden. Ursache des Problems war, dass der IDM eine Funktion entsprechend dem Anwendungsobjekt, zu dem die Funktion gehört, aufruft. Es wurde hierbei nicht berücksichtigt wie die Funktion am IDM angemeldet wurde. Jetzt wird an der Funktion gespeichert, wie sie angemeldet wurde. Gibt es hierbei eine Diskrepanz zu dem Anwendungsobjekt, wird die Funktion nicht aufgerufen. In der Trace-Datei ist in einem solchen Fall eine zusätzliche Fehlermeldung zu sehen.

21.6 Debugger

IDM-12290: Auf allen Plattformen wird nun eine Combobox für die Expression-Eingabe und das Suchfenster benutzt.

IDM-8551: Der IDM Debugger stoppt nicht mehr nach jeder Aktualisierung des IDM Profilers, wenn dieser im automatischen Modus läuft. Die Profiler-Daten werden jetzt aktualisiert, ohne dass der Debugger stoppt.

Außerdem hat man nun beim Export der Profiler-Daten zusätzlich die Formatierungsoptionen, dass Felder durch Kommas getrennt werden oder eine feste Breite erhalten.

Im Menü *„Profiler“* kann nun auch das Ausgabeformat für das Speichern der Profiler-Daten gewählt werden. Folgende Formate stehen zur Verfügung:

- *„Export tab separated“*
  Jede Spalte wird durch ein Tabulator-Zeichen getrennt (bisheriges Format).
- *„Export comma separated“*
  Jede Spalte wird durch ein Komma getrennt.
  Falls der Text einer Spalte Leerzeichen oder Kommas enthält, wird er in " (Anführungszeichen) eingeschlossen. Im Text vorhandene " werden dann verdoppelt.
- *„Export with fixed width“*
  Die Spaltenbreite ergibt sich aus dem jeweils längsten Text in der Spalte plus zwei Leerzeichen.

22 A.06.01.g

22.1 Unix

Neu: Als neue Plattform wird AIX 7.1 für PowerPC mit Compiler GCC 4.8 und Toolkit Motif 2.1 unterstützt.

22.2 Windows

IDM-12854: Bei aktivierter **UI Automation** konnte es zu einem Abbruch (Assfail in *„Module: object/slot Line: 3560“*) kommen, wenn ein IDM-Objekt zerstört wurde, für das UI Automation aktiviert war. Im vorliegenden Fall wurde das zum IDM-Objekt gehörende UI Automation Objekt erst sehr viel später freigegeben, wodurch ein Zugriff auf das zerstörte IDM-Objekt versucht wurde.

Als **Workaround** lässt sich die UI Automation Unterstützung deaktivieren (siehe C-Funktionen DM_Control und DM_ControlEx bzw. COBOL-Funktion DMcob_Control).

IDM-12842: Ein ***Image***-Objekt mit einer leeren ***Tile***-Ressource führt in der Debug-Version des IDM nicht mehr zu einer Meldung *„ASSERTION FAILED in "..\wsimsw\itile.c" line 388:…“*. Nun wird lediglich eine Fehlermeldung ausgegeben, wenn eine ***Tile***-Ressource ohne Daten verwendet wird.

Außerdem wurden weitere die ***Tile***-Ressource betreffende Fehlermeldungen überarbeitet, sodass sie mindestens den Dateinamen und meist auch das Label ausgeben.

IDM-12837: Das ***Image***-Objekt und der ***Statictext*** aktualisieren nun die Schriftart, auch wenn lediglich das Attribut *.font* geändert wird. Bisher war eine weitere Aktion erforderlich, die ein Neuzeichnen des Objekts auslöste.

IDM-12777: Am ***Treeview*** wird das Attribut *.firstchar* jetzt auch aktualisiert, wenn am sichtbaren Objekt zu große Werte gesetzt werden. Der aktuelle Wert kann nun abgefragt werden (wie bei *.topitem*).

IDM-12751: Bei einem leeren ***Edittext*** konnte kein **Drag & Drop** ausgeführt werden. Jetzt kann Drag & Drop ausgeführt werden, wenn sich die Maus entweder über markiertem Text oder neben Text befindet. Wenn sich die Maus über (unmarkierten) Text befindet, dann wird kein Drag & Drop gestartet, sondern der Text markiert.

IDM-12749: Bei Änderung des Attributs *.level* eines ***Treeview***-Eintrags im sichtbaren Zustand wurde unter Umständen fälschlicherweise das Attribut *.activeitem* geändert und es kam zu einem *select-activate*-Ereignis. Die Ursache war, dass WINDOWS beim Löschen des gesamten ***Treeview***-Inhalts Benachrichtigungen verschickt, dass sich der selektierte Eintrag geändert hätte, und solche Benachrichtigungen zur Unterstützung der „UI Automation“ ausgewertet wurden. Jetzt erfolgt keine Auswertung, wenn der ***Treeview*** neu gefüllt wird.

Anmerkung

Auch andere Attributänderungen konnten zu dem Fehlverhalten führen.

IDM-12669 (keine Änderung): Wenn eine Änderung des ***Treeview*** implizit auch *.activeitem* veränderte, wurde der Wert Attributs nicht aktualisiert, wodurch Anzeige und Attributwert inkonsistent waren. Dies passierte zum Beispiel, wenn sich *.activeitem* in einem Bereich befand, der mit der **:delete**-Methode gelöscht wurde.

Anmerkung

Das Problem wurde durch die Behebung eines anderen Fehlers beseitigt.

IDM-12635: Der ***Treeview*** scrollt nun auch in der klassischen Ansicht von WINDOWS 7 (32-Bit) nicht mehr horizontal, wenn ein Eintrag derart geändert wird, dass der ***Treeview*** dadurch eine horizontale Scrollbar benötigt.

IDM-10564: Mit der Startoption **-writeole** wird unter anderem eine **reg**-Datei zur Registrierung des **OLE-Controls** erzeugt. Das Format dieser Datei war veraltet, sodass keine Registrierung erfolgte. Jetzt wird das aktuelle Format „Windows Registry Editor Version 5.00“ verwendet. Außerdem erfolgt die Registrierung nicht mehr unter HKEY_CLASSES_ROOT sondern unter HKEY_LOCAL_MACHINE.

Hinweis

Das Windows-Werkzeug **regedit.exe** setzt bei einem Fehler meist keinen Fehlerstatus. Wenn es Probleme mit Start eines OLE-Controls gibt, sollte die Registry überprüft werden, ob die Registrierungsdaten überhaupt geladen wurden. Änderungen der Windows-Registry sollten mit Vorsicht erfolgen, da Windows bei Fehlern unter Umständen nicht mehr gestartet werden kann.

Neue Option -userregistry

Um ein OLE-Control nur für den aktuellen Benutzer zu registrieren, kann bei **-writeole** zusätzlich die Option **-userregistry** angegeben werden. Dann werden die Registrierungsdaten unter HKEY_CURRENT_USER in die Windows-Registry geschrieben.

22.3 Qt

IDM-12844: Sonderzeichen, die in Kombination mit der `AltGr`-Taste eingegeben werden, erzeugen bei einem ***Edittext* ohne Format** nun ein *charinput*-Ereignis und werden bei einem ***Edittext* mit Format** nun nicht nur angezeigt sondern auch übernommen.

IDM-12333: Das Zuweisen eines sichtbaren ***Edittextes*** an ein ***Tablefield*** (Attribut *.edittext*) zeigt nun das erwartete Verhalten. Probleme gab es, weil der ***Edittext*** bereits als falsche Widget-Klasse angelegt war. Jetzt wird er gegebenenfalls unsichtbar gemacht, um die falsche Widget-Klasse zu zerstören. Außerdem verschwand der ***Edittext*** bei einem Fokuswechsel, wenn sein Attribut *.visible* und das Attribut *.editpos* des ***Tablefields*** auf *false* gesetzt waren.

22.4 Kern

IDM-12832: Eine **Gültigkeitsverletzung** wird nun auch bei der **Rückgabe von Sammlungen** geprüft. Außerdem werden Sammlungen bei der Rückgabe automatisch (entsprechend der Dokumentation) in den korrekten Sammlungsdatentyp gewandelt, ohne dabei die Elemente (z.B. beim Datentyp *vector* mit Gültigkeitsbereich) zu konvertieren. Es erfolgt nun auch keine Rückgabe der Regel-ID bei der Rückkehr aus Regeln mit dem Datentyp *void*.

Achtung Verhaltensänderung

Bei Verlassen einer Regel wird nun der Rückgabewert (`return`-Anweisung) auf den korrekten Rückgabetyp und dessen Gültigkeitsbereich überprüft. Eine Typverletzung, das Fehlen eines Rückgabewertes oder eine Rückgabe bei einer *void*-Regel werden mit der Fehlermeldung *„bad type of return value“* angezeigt, eine Gültigkeitsverletzung mit der Fehlermeldung *„scope violation of the return value“*. Wird ein fehlerhafter Rückgabewert erkannt, so wird stattdessen *nothing* zurückgegeben.

IDM-12781: Ein Absturz beim Zugriff auf das Attribut *.ixmldomdocument2* eines ***Doccursors***, der durch den Zugriff auf ein Strukturelement ohne vorherige Prüfung des Zeigers verursacht wurde, tritt nun nicht mehr auf.

IDM-12778: Ein Aufruf der Methode **:select** des ***Doccursors*** mit dem Argument *select_last* positioniert den ***Doccursor*** nun wieder auf den letzten Knoten, der dem Suchmuster entspricht.

IDM-12759: Bei der interaktiven Zahleneingabe – genauer dem Ersetzen eines selektierten Bereichs – kam es in besonderen Konstellationen bei den **numerischen Formaten** „f“ und „t“ zur Verschiebung des Zahlenwertes (Auffüllen mit Nullen im Vorkommateil). Dieser Fehler wurde behoben.

IDM-12689: **Statische Variablen** werden bei **rekursiven** Regel- oder Methodenaufrufen nicht mehr erneut initialisiert.

22.5 Editor

IDM-10154: Es wird nun vermieden, dass ein ***Fenster*** unter MICROSOFT WINDOWS allein durch das Anzeigen verkleinert wird. Das Problem lag daran, dass bei einem Fenster ohne maximale Größe die maximale „Tracking Size“ implizit auf die Größe des Vaters gesetzt wurde, wodurch es beim Sichtbarmachen auf diese Größe verkleinert wurde.

Hinweis

Ein zu großes Fenster kann nicht mit der Maus verkleinert werden. Entweder muss in den Geometrie-Eigenschaften der gewünschte Wert per Tastatur eingegeben oder die Größe des Darstellungsbereichs entsprechend angepasst werden.

22.6 Debugger

IDM-12840: Ein Ausführungsschritt (Step) über das Ende einer benannten Regel oder Methode führt nicht mehr zum Fehler *„ACCESSING ILLEGAL YAny“*.

IDM-12782: Abstürze des Debuggers, die beim Zerstören von debuggten Objekten auftreten konnten, wurden behoben.

22.7 COBOL

IDM-12747: Bei falscher Verwendung des Vektorindex kommt es nicht mehr zu einem „Assfail“. Jetzt wird ein Fehlercode gesetzt und eine Fehlermeldung in die Log-Datei geschrieben, sodass die Anwendung weiterläuft. Das Problem trat zum Beispiel auf, wenn der Fehlercode einer COBOL-Schnittstellenfunktion zur Bearbeitung von Vektoren nicht abgefragt wurde bevor die nächste dieser Funktionen aufgerufen wurde.

IDM-12656: Der Aufruf von **DMcob_CreateObject** mit einem Modell-Identifikator führt zum Fehler *„Specified object-ID is not valid“*, weil der Parameter keine *DM-ID* aufnehmen kann. Deshalb wurde eine neue Funktion **DMcob_CreateFromModel** eingeführt, die dieses Problem löst.

22.7.1 Neue Schnittstellenfunktion DMcob_CreateFromModel

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

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

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

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

Parameter

-> DM-Options of DM-StdArgs

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

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

<- DM-objectid

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

-> DM-parentid

Dieser Parameter definiert den Vater des neu erzeugten Objekts.

-> DM-modelid

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

Rückgabewert

DM-status of DM-StdArgs

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

23 A.06.01.f3

23.1 Windows

IDM-12749: Wenn im sichtbaren Zustand das Attribut *.level* eines ***Treeview***-Eintrags geändert wurde, dann wurde unter Umständen das Attribut *.activeitem* fälschlicherweise geändert und es kam zu einem *select-activate*-Ereignis. Das Problem lag daran, dass beim Löschen des gesamten ***Treeview***-Inhalts von Windows Benachrichtigungen geschickt wurden, dass sich der selektierte Eintrag geändert hätte. Diese Benachrichtigungen wurden für die Unterstützung der UI Automation ausgewertet. Jetzt wird die Auswertung vermieden, wenn der ***Treeview*** neu gefüllt wird.

Anmerkung

Auch andere Attributänderungen konnten zu dem Fehlverhalten führen.

23.2 Qt

IDM-12760: Obwohl ein Mausklick abgearbeitet und ein *select*-Ereignis ausgelöst wurde, wurde der Klick an das Vaterobjekt weitergeleitet und löste dort auch ein *select*-Ereignis aus. Betroffen waren das ***Canvas*** und die **Gruppierungsobjekte**.

IDM-12750: Am ***Tablefield*** fehlte die Tastatursteuerung für `Space` und `Return`. Nun werden die Ereignisse *select* und *dbselect* auch über Tastatur ausgelöst. Außerdem kann mit `Return` zum nächsten Feld navigiert werden.

Hinweis

Am ***Tablefield*** kann der angebundene ***Edittext*** mit der `F2`-Taste aktiviert werden.

23.3 Kern

IDM-12755: Die indizierte Zuweisung eines listenartigen Wertes an eine lokale Variable ist wie in der Dokumentation beschrieben verboten und erzeugt eine Fehlermeldung. Listenartige Variablen dürfen nur skalare Werte enthalten.

IDM-12754: Die indizierte Zuweisung auf eine *refvec*-Liste mit einem Index außerhalb der Listengröße erzeugt nun auch einen Fehler wenn der Wert schon im *refvec* vorhanden ist und vermeidet einen Assfail. Auch das Ersetzen einer schon vorhandenen ID zum Umsortieren (an die letzte Stelle) funktioniert nun.

IDM-12745: Wird in einer Objekt-Regel bzw. Methode das Objekt mit **:destroy()** zerstört, kommt es beim Verlassen der Regel nicht mehr zu einem Assfail, wenn es sich bei der übergeordneten Regel ebenfalls um eine Methode des zerstörten Objekts handelt.

23.4 Debugger

IDM-12744: Es kommt nicht mehr zu Abstürzen, wenn Objekte innerhalb ihrer eigenen Regeln gelöscht werden.

24 A.06.01.f2

24.1 COBOL

IDM-12746: Bei den Funktionen **DMcob_SetVector**, **DMcob_LSetVector**, **DMcob_GetVector** und **DMcob_LGetVector** kommt es nicht mehr zu einem Abbruch, wenn in den Parametern *DM-ENDCOL* und *DM-ENDROW* der Wert *0* übergeben wird. Der Wert wird nun wieder korrekt zu *void* gewandelt und nicht mehr fälschlich als Index *[0,0]* behandelt.

25 A.06.01.f

25.1 Kompatibilität

Das Binärformat hat sich geändert. Binärdateien müssen neu generiert werden.

25.2 Windows

12346: Der Windows **UI-State** wurde manchmal nicht weitergereicht. Dadurch wurde der **Fokusrahmen** nicht entsprechend dem UI-State gezeichnet. Jetzt wird zentral der UI-State für alle Objekte initialisiert. Außerdem wird der UI-State nicht mehr zwischengespeichert, sondern aktuell von Windows erfragt.

Hinweis zum UI-State unter Windows

Beim Öffnen eines Fensters wird der UI-State entsprechend dem letzten Eingabeereignis gesetzt. War das letzte Eingabeereignis ein Mausereignis, dann werden keine Fokusrahmen dargestellt. Die Darstellung von Fokusrahmen wird eingeschaltet, sobald eine Navigationstaste gedrückt wird.

25.3 Qt

12733: Bei *select* auf einen leeren Bereich im ***Tablefield*** wird nun der korrekte Index *nothing* zurückgegeben.

12718, 12717, 12716: Bei den Objekten ***Edittext***, ***Poptext*** und ***Tablefield*** werden formatierte Strings jetzt vor der Darstellung wieder in die korrekte Codepage gewandelt.

12685: Die ***Groupbox*** gibt nun für *.real_height* und *.real_width* nicht mehr die virtuellen Größenwerte zurück.

12672: In bestimmten Situationen konnte es dazu kommen, dass ein gesetztes *.firstchar* wieder auf *1* zurückgesetzt und der ***Treeview*** weggescrollt wurde. Jetzt sollte die *.firstchar*-Position im ***Treeview*** immer korrekt beibehalten bzw. wiederhergestellt werden.

12671 (keine Änderung):

Hinweis zum Treeview unter Qt

In den **Qt-Versionen 4.x** gibt es beim ***Treeview***-Objekt einen Darstellungsfehler (**QTBUG-9352**), der unter Umständen bewirkt, dass Einträge entweder abgeschnitten oder überhaupt nicht dargestellt werden. Dies liegt daran, dass Qt die Spaltenbreite für die Anzeige der ***Treeview***-Einträge nur anhand der Einträge im sichtbaren Bereich (+/- einem Offset) berechnet. Das heißt, dass dieser Effekt in einem ***Treeview*** mit besonders vielen Einträgen, deren Länge sich teilweise deutlich unterscheidet, zu tragen kommen kann. Ebenso betroffen sind ***Treeviews***, die besonders viele Einträge mit vielen Hierarchiestufen enthalten.

Workaround

Als Workaround sollte man in diesem Fall den längsten Eintrag (im Sinne von benötigtem Darstellungsbereich, also Einrückung und Text) vor dem initialen Sichtbarmachen des ***Treeviews*** in den angezeigten Bereich bringen (z.B. über *.topitem*).

25.4 Kern

12729: Eine fehlerhafte, optimierende Speicherallokierung wurde korrigiert. Diese sollte für die maximale Ausnutzung des 64k-Blocks bei Regel-Instruktionen sorgen, hatte aber unter Umständen eine Zugriffsverletzung zur Folge.

25.5 COBOL

12726: Ein **dynamisches Linken** der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL war nicht möglich, da einige Symbole der dynamischen IDM-Bibliotheken nicht exportiert wurden.

26 A.06.01.e

26.1 Kompatibilität

Das Binärformat hat sich geändert. Binärdateien müssen neu generiert werden.

26.2 Windows

12701: Beim Schließen eines ***Fensters*** mit überschriebener **:set**-Methode kommt es nicht mehr zu einem Assfail. Das Problem trat auf, wenn die **:set**-Methode beim Sichtbarmachen des Fensters aufgerufen wurde, beim Unsichtbarmachen aber nicht (zum Beispiel durch Klick auf das Schließen-Symbol in der Titelleiste).

12668 (keine Änderung): Das ***Treeview***-Control unter Windows stellt maximal *256* Hierarchiestufen dar.

Hinweis zum Treeview-Objekt und .level-Attribut

Das MICROSOFT WINDOWS Treeview-Control rückt maximal *256* Hierarchiestufen ein. Die 257. Stufe erscheint dann wieder ganz vorne im ***Treeview***-Objekt. Dies wiederholt sich alle 256 Stufen. Die Funktionalität ist davon nicht betroffen, lediglich die Darstellung.

Das Problem wurde in allen Windows-Versionen (einschließlich WINDOWS 10) festgestellt.

26.3 Qt

12712: Die Option *opt_auto_close* am ***Fenster*** wird nun auch für Qt unterstützt. Beim Schließen des Fensters wird nur das *close*-Ereignis erzeugt ohne das Fenster unsichtbar zu schalten.

26.4 Kern

12694: Es kommt zu keiner fehlerhaften **Parameterprüfung** mehr, wenn man Regeln aufruft unter Ausnutzung von Defaultwerten, bei denen der Rückgabewert wiederum Parameter eines erneuten Regelaufrufs ist. Zusätzlich wird nun die **Gültigkeitsprüfung** des Defaultwertes direkt beim Laden ausgeführt.

12691: Beim **Löschen eines Attributs** wird nun nicht mehr über allokierte Speicherbereiche hinaus geschrieben und kein ungültiger Heap bzw. ungültige Speicherverwaltung hinterlassen.

12687: Beim Binärschreiben eines ***Tablefields*** mit gesetztem *.rowheader* oder *.colheader* kommt es nicht mehr zum Überschreiben von *.content[I,J]*-Strings des Headers mit Strings nach dem Header (*.field[I,J]*).

12678: Wenn ein Attribut an einer Instanz gelöscht wird, werden bei der Berechnung von *.itemorder* nun geerbte Attribute berücksichtigt. Auf lokal gesetzte Attribute wird nun geachtet, um das Attribut an der richtigen Stelle der Ordnung wegzunehmen.

12676: Der **IDM Builder Prozess** bricht nun auch im zweiten Lauf bei Fehlern konsistent ab.

12637: Bei Regeln unterhalb von exportierten Objekten kommt es nicht mehr zu Fehlern bei der statischen Überprüfung des **Gültigkeitsbereichs**.

26.5 Unterstützung von Microsoft UI Automation

Verfügbarkeit

Nur IDM FÜR WINDOWS

UI Automation (im Folgenden meist mit UIA abgekürzt) ist ein Framework von MICROSOFT um die Barrierefreiheit (Accessibility) und das automatisierte Testen von Benutzeroberflächen zu ermöglichen.

Der Fokus bei der Erweiterung des IDM liegt dabei auf der Bereitstellung der Infrastruktur für das automatisierte Testen.

Es macht dabei die Oberflächenelemente (UIA Elements) in einer Baumstruktur zugänglich. Wobei ein UIA Element Eigenschaften (Properties) aufweist, abhängig auch von seinem Typ (UIA Control Type). UIA Elemente können außerdem mehrere Steuerungsmuster (UIA Control Patterns) aufweisen, welche zusätzliche Eigenschaften oder Interaktionsmöglichkeiten (UIA Control Pattern Methods) bieten. Ereignisse zeigen dabei an, ob sich an einem UIA Element Änderungen ergeben haben.

Diese Dokumentation dient dazu, die Funktionalität des IDM bezüglich der UI Automation Unterstützung kenntlich zu machen und ist weitestgehend nur für Entwickler von UI Automation Clients interessant. Es stellt keine „Programmierdokumentation“ dar, sondern dokumentiert lediglich die Unterstützung durch den IDM. Für die Nutzung von UI Automation sei auf die Microsoft-Dokumentation verwiesen.

26.5.1 Zuordnung von IDM-Objektklassen zu UIA Control Types

Der IDM hat als ein Designprinzip die Verwendung der vom System bzw. Toolkit bereitgestellten Objektklassen (Controls, Widgets) um die Oberflächenobjekte zu implementieren und damit Bedienung und Aussehen möglichst konform zum System zu halten.

Eine Vielzahl von IDM-Objektklassen werden durch den „UI Automation Support for Standard Controls“ von MICROSOFT abgedeckt. Der IDM ergänzt wenn möglich und erforderlich noch die notwendigen Eigenschaften.

Im Hinblick auf die UI Automation Unterstützung im IDM lassen sich folgende Kategorien unterschieden:

D Besondere Objektklassen können nicht mit einer zusätzlichen UIA-Unterstützung durch den IDM ausgestattet werden, verfügen aber durch die Microsoft-Default-Implementierung über eine ausreichende Unterstützung. Dazu zählen z.B. ***menubox***, ***menuitem***, ***menusep***, ***messagebox*** und ***filereq***.

E Die meisten dieser Objektklassen erhalten ergänzende UIA-Unterstützung durch den IDM, insbesondere um deren Identifikation zu erleichtern.

X Vom IDM eigenimplementierte Objektklassen bzw. Objektklassen mit unzureichender Microsoft-Default-Implementierung erhalten eine eigene UIA-Unterstützung. Das sind hauptsächlich die Objektklassen ***image***, ***tablefield***, ***splitbox***, ***layoutbox***, ***progressbar***, ***poptext*** und ***toolbar***.

U Für besondere Objektklassen wie ***control***, ***canvas*** und USW-Objekte obliegt die UIA-Unterstützung dem Anwendungsprogrammierer bzw. Objekt-Implemetierer.

Die folgende Tabelle soll einen Anhaltspunkt für die Zuordnung der IDM-Objektklassen zu UIA Control Types liefern sowie der bereitgestellten Funktionalität durch die Pattern-Zugehörigkeit:

Kategorie	IDM-Objektklasse	UIA Control Type	Unterstützte UIA Control Patterns
U	***canvas***	Pane	Ist vom Canvas-Programmierer zu implementieren
E	***checkbox***	CheckBox	Toggle, Invoke
U	***control***	Pane	Ist vom OLE/Control-Objekt abhängig
E	***edittext***	Edit	Systemabhängig – typischerweise Value, Text, Text2 Multiline-Texte mit Scrollbars haben meist noch das Scroll-Pattern
E	***edittext*** mit *.options[opt_rf] = true*	Document	Systemabhängig – typischerweise Value, Text, Text2
D	***filereq***	Window	Systemabhängig
X	***groupbox***	Pane Kind: Group	Invoke, Scroll (bei virtuellen Bereichen)

Kategorie	IDM-Objekt-klasse	UIA Control Type	Unterstützte UIA Control Patterns
X	***image***	Button Kinder: Text, Image (wenn vor-handen)	Invoke, Toggle
X	***layoutbox***	Pane	Scroll (bei virtuellen Bereichen)
E	***listbox***	List Kinder: ListItem	Scroll, Selection, Invoke
D	***menubox***	Menu	ExpandCollapse
D	***menuitem***	MenuItem	Toggle, Invoke, ExpandCollapse, Selec-tionItem – abhängig vom *.style*-Attribut
D	***menusep***	Separator	–
D	***messagebox***	Window	Systemabhängig
E	***notebook***	Tab	Selection
E	***notepage***	TabItem/Pane	SelectionItem
X	***poptext***	ComboBox Kinder: Edit, List, Text (entsprechend dem *.style*)	ExpandCollapse, Value sowie Selection – abhängig vom *.style*-Attribut
X	***progressbar***	ProgressBar	RangeValue
X	***pushbutton***	Button	Invoke
E	***radiobutton***	RadioButton	SelectionItem
X	***rectangle***	Button	Invoke
X	***scrollbar***	ScrollBar	RangeValue
X	***scrollbar*** (Slider)	Slider	RangeValue
X	***spinbox***	Spinner	RangeValue, Value, Selection – abhängig vom *.style*-Attribut

Kategorie	IDM-Objekt-klasse	UIA Control Type	Unterstützte UIA Control Patterns
X	***splitbox***	Pane Kinder: Pane	Sichtbare Split-Bereiche sind Sub-Panes mit Transform-Pattern
X	***statictext*** (insensitiv)	Text	Text
X	***statictext*** (sensitiv)	Button	Invoke
E	***statusbar***	StatusBar	–
X	***tablefield***	Table Kinder: ListItem	Grid, Table, GridItem, TableItem, Invoke, Scroll, Selection
X	***toolbar***	ToolBar	Dock, Transform, Window (abhängig vom *.docking*-Attribut), Invoke
E	***treeview***	Tree	Scroll, Selection
E	***window***	Window	Window, Transform

Änderungen an der UI Automation Unterstützung können durch MICROSOFT jederzeit erfolgen, sodass bezüglich den Zuordnungen zu UIA-ControlTypes, Patterns, Properties und Events jederzeit Änderungen möglich sind.

26.5.2 Unterstützte UIA Properties und Control Pattern Methoden

Grundsätzlich sind die Properties, Methoden und Funktionalität durch die „UI Automation Specification“ vorgegeben. Dieses Kapitel hat also eher einen informativen Charakter um den Zusammenhang mit dem IDM herzustellen.

UIA Property/Method	IDM-Attribut	Bemerkungen
AcceleratorKey	*.accelerator*	Die Tastenkombination des ***Accelerators*** wird geliefert.
AccessKey	*.text*	Der **Mnemonic**-Buchstabe im Text wird zurückgeliefert, beispielsweise *"Alt+B"* für *.text "&Bearbeiten"*.
AutomationId	*.label*, *.acc_label*	*.acc_label* genießt, wenn gesetzt, Vorrang vor *.label*. Der *.label* wird unter Umständen ergänzt durch die Nummerierung „:[<I>]“ um eine eindeutige Identifikation auf Geschwisterebene zu erreichen.

UIA Property/Method	IDM-Attribut	Bemerkungen
BoundingRectangle	*.x*, *.y* *.width*, *.height* *.xauto*, *.yauto*	Position und Größe des Objekts.
ClassName	–	**Achtung** Systemnamen oder IDM-spezifische Namen die sich von Version zu Version ändern können und keine 1:1-Korrelation mit der IDM-Objektklasse besitzen.
ClickablePoint	–	Punkt innerhalb des BoundingRectangle zur Ausführung eines Mausklicks.
ControlType	*.class* (und anderen Attributen wie *.options*, *.style*, *.arrows*)	Die Umsetzung auf einen UIA Control Type hängt nicht nur von der Objektklasse ab, sondern unter Umständen von zusätzlichen Objektattributen.
FrameworkId	–	Wird von der UIA-Implementierung von MICROSOFT geliefert.
HasKeyboardFocus	*.focus*	Gibt wieder, ob das Objekt den Tastaturfokus besitzt.
HelpText	*.toolhelp* bzw. *.statushelp*	Falls kein Hilfetext in *.toolhelp* gesetzt wurde, wird *.statushelp* zurückgeliefert.
IsEnabled	*.sensitive* bzw. *.real_sensitive*	Liefert die Bedienbarkeit bzw. Sensitivität des UIA Elements.
IsKeyboardFocusable	–	Objekt ist sensitiv und kann den Tastaturfokus erhalten.
IsOffscreen		UIA Element ist sichtbar, kann aber durch darüberliegende Fenster oder Elemente verdeckt sein.
IsPassword	*.format*	Ein Format mit verdeckter Formatierung (Formatstring beginnt mit „S“) liefert für diese Property *true* zurück.
LocalizedControlType	–	Wird von der UIA-Implementierung von MICROSOFT geliefert.

UIA Property/Method	IDM-Attribut	Bemerkungen
LabeledBy	–	Ist dem IDM-Objekt ein insensitiver ***Statictext*** vorangestellt, wird das zugehörige UIA Element geliefert.
Name	*.text*, *.title*, *.acc_text* Bei Unterstrukturen (***Listbox***, ***Treeview***, Content) oft auch *.content*, *.content[I]*, *.content[R,C]*	Der Name des UIA Elements. Kann durch das Attribut *.acc_text* überschrieben werden.
NativeWindowHandle	*AT_WinHandle*	Entspricht meist dem HWND den **DM_GetToolkitData()** auf *AT_WinHandle* zurückliefert. Usually the same as the HWND that **DM_GetToolkitData()** returns for *AT_WinHandle*.
Orientation	*.direction*, *.docking*	Für IDM-Objektklassen wie ***splitbox***, ***scrollbar***, ***toolbar*** und ***tablefield*** wird hier die korrespondierende Ausrichtung geliefert.
OrientationType_Horizontal	*.direction = 2*	
OrientationType_Vertical	*.direction = 1*	
OrientationType_Horizontal	*.docking = dock_<up|down|window>*	
OrientationType_Vertical	*.docking = dock_<left|right>*	
ProcessId	–	Wird von der UIA-Implementierung von MICROSOFT geliefert.
ProviderDescription	–	Wird von der UIA-Implementierung von MICROSOFT geliefert.
RuntimeId	–	Wird meist von der UIA-Implementierung von MICROSOFT geliefert.

26.5.2.1 Value Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ValueIsReadOnly	*.editable* *.editable[R,C]*	Eingabefeld oder Tabellenzelle ist editierbar. Eine ***Listbox*** liefert hier immer *true*.
Value	*.content* *.content[l]* *.content[R,C]*	Liefert den (formatierten) Inhalt eines Eingabefelds, ***Poptext***-, ***Treeview***- oder ***Listbox***-Elements oder einer Tabellenzelle.
SetValue()	*.content* *.content[l]* *.content[R,C]*	Verändert den Wert des Eingabefelds oder der Tabellenzelle.

26.5.2.2 RangeValue Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ValueIsReadOnly	–	Range-Wert ist über SetValue() veränderbar.
Value	*.curvalue*	Range-Wert einer ***Spinbox*** oder ***Scrollbar***.
Minimum	*.minvalue*	Untere Range-Grenze.
Maximum	*.maxvalue*	Obere Range-Grenze.
SetValue()	–	Verändert den Range-Wert.

26.5.2.3 Scroll Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
HorizontallyScrollable	–	Horizontales Scrollen ist möglich.
HorizonalScrollPercent	–	Errechneter Wert aus *.vwidth* und realer Breite in %.
HorizontalViewSize	–	Errechneter Wert aus *.vwidth* und realer Breite in %.
VerticallyScrollable	–	Vertikales Scrollen ist möglich.
VericalScrollPercent	–	Errechneter Wert aus *.vheight* und realer Höhe in %.

UIA Property/Method	IDM-Attribut	Bemerkungen
VericalViewSize	–	Errechneter Wert aus *.vheight* und realer Höhe in %.
Scroll()	–	Scrollt relativ.
SetScrollPercent()	–	Scrollt zu einer bestimmten %-Position.

26.5.2.4 ScrollItem Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ScrollIntoView()	–	Scrollt das Element in den sichtbaren Bereich.

26.5.2.5 Transform Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
CanMove	*.moveable*	***Fenster*** bzw. ***Toolbar*** kann verschoben werden.
CanResize	*.sizeable*	***Fenster*** bzw. ***Toolbar*** kann verkleinert und vergrößert werden. Oder der ***Splibox***-Bereich kann verkleinert und vergrößert werden.
CanRotate	–	Nicht vom IDM abgedeckt.
Move()	–	Erlaubt die Verschiebung eines ***Fensters*** oder einer ***Toolbar***.
Resize()	–	Erlaubt die Verkleinerung und Vergrößerung eines ***Fensters***, ***Toolbar*** oder ***Splitbox***-Bereichs.
Rotate()	–	–

26.5.2.6 Grid Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ColumnCount	*.colcount*	Liefert die Anzahl der sichtbaren Spalten einer Tabelle.

UIA Property/Method	IDM-Attribut	Bemerkungen
RowCount	*.rowcount*	Liefert die Anzahl der sichtbaren Zeilen einer Tabelle.
GetItem()	–	Liefert das UIA Element einer Zelle.

26.5.2.7 GridItem Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
GridItemColumn	–	Spaltenindex (ab *0* und ohne unsichtbare Spalten).
GridItemColumnSpan	–	Ist immer *1* da im IDM keine Zellen zusammengefasst werden können.
GridItemRowCount	–	Zeilenindex (ab *0* und ohne unsichtbare Zeilen).
GridItemRowSpan	–	Ist immer *1* da im IDM keine Zellen zusammengefasst werden können.

26.5.2.8 Selection Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
SelectionCanSelectMultiple	*.multisel* *.selstyle* *.selection[]*	***Listboxen*** und Tabellen können eine Mehrfachselektion erlauben.
SelectionIsRequired	–	Ist eine Selektion erforderlich?
GetSelection()	–	Liefert die selektierten UIA Elemente.

26.5.2.9 SelectionItem Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
SelectionItemIsSelected	.active .activeitem	Liefert *true* für ein aktives Element.
AddToSelection()	–	Fügt ein Element zu einer Mehrfachselektion hinzu.
RemoveFromSelect()	–	Entfernt ein selektiertes Element aus einer Mehrfachselektion.

UIA Property/Method	IDM-Attribut	Bemerkungen
Select()	–	Selektiert dieses Element.

26.5.2.10 Toggle Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ToggleState	*.active* *.state*	Aktivierung bzw. Zustand einer ***Checkbox***, ***Image***-Objekts oder ***Menuitem*** im *checkbox*-Stil.
Toggle()	–	Schaltet die Zustände zwischen *checked*, *unchecked* und *indeterminate* um.

26.5.2.11 ExpandCollapse Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
ExpandCollapseState	*.open* (***Treeview***)	–
Collapse()	–	Schließen eines Unterbaums.
Expand()	–	Öffnen eines Unterbaums.

26.5.2.12 Window Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
CanMinimize	*.iconifyable*	***Fenster*** kann minimiert werden.
CanMaximize	*.maximizable*	***Fenster*** kann maximiert werden.
IsTopmost	*.top_most*	***Fenster*** ist immer vor allen anderen Fenstern.
WindowIsModal	*.dialogbox*	***Fenster*** mit *.dialogbox* = *true* sind modale Fenster.
WindowInteractionState	–	–
WindowVisualState	–	Zustand des ***Fensters*** (minimiert, maximiert usw.).
Close()	–	Schließt das ***Fenster***.

UIA Property/Method	IDM-Attribut	Bemerkungen
SetWindowVisualState()	–	Erlaubt es, das ***Fenster*** zu minimieren oder zu maximieren.
WaitForInputIdle()	–	–

26.5.2.13 Dock Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
DockDockPosition	*.docking*	Liefert die Docking-Position einer ***Toolbar***.
SetDockPosition()	–	Kann zum Umdocken einer ***Toolbar*** verwendet werden.

26.5.2.14 Table Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
TableRowOrColumnMajor	*.direction*	Zeigt die Ausrichtung der Tabelle an.
GetColumnHeaders()	–	Liefert die UIA Elemente der sichtbaren Spalten-Header-Zellen.
GetRowHeaders()	–	Liefert die UIA Elemente der sichtbaren Zeilen-Header-Zellen.

26.5.2.15 TableItem Pattern

UIA Property/Method	IDM-Attribut	Bemerkungen
GetColumnHeaderItems()	–	Liefert die zugehörigen UIA Elemente im Header-Bereich *[row,1] … [row,.colheader]* der Tabelle.
GetRowHeaderItems()	–	Liefert die zugehörigen UIA Elemente im Header-Bereich *[1,col] … [.rowheader,col]* der Tabelle.

26.5.3 UIA Objektidentifikation

Die von UI Automation definierten Schlüssel-Properties für die Identifikation von UIA Elementen sind AutomationId, Name, ControlType und RuntimeId.

Um eine möglichst konsistente und sprachunabhängige Identifikation zu ermöglichen, wird deshalb bei den meisten IDM-Objekten der IDM Objekt-Label als AutomationId vergeben. Bei Mehrdeutigkeiten erfolgen bei geerbten Bezeichnern noch die IDM-typische Indizierung „:[<Nr>]“ nach dem Label für *Nr >= 2*. Über das *.acc_label*-Attribut kann die AutomationId überschrieben werden. Dann hat allerdings der Anwendungsprogrammierer für die Eindeutigkeit zu sorgen.

Oft wird auch die Name-Property herangezogen, welche typischerweise den sichtbaren statischen Text-String (also nicht dem dynamischen Inhalt) widerspiegelt. Auch dies ist vom Anwendungsprogrammierer überschreibbar durch das *.acc_text*-Attribut.

In einem UI Automation Element Tree sind durch diese Properties damit Objekte meist eindeutig identifizierbar.

Die RuntimeId basiert im allgemeinen auf dem Window-Handle und kann sich somit auch bei Attributänderungen ändern. Sie eignet sich somit weniger für eine stabile Objektidentifikation.

26.5.4 Besonderheiten

26.5.4.1 Edittext

Für die Änderung des Textes ist die Methode SetValue() des Value Pattern zu benutzen. Dies führt zu einem *charinput*-Ereignis im IDM. Das *modified*-Ereignis um eine Änderung zu signalisieren wird allerdings erst versendet, wenn der ***Edittext*** den Fokus verliert. Konsistenter mit einer normalen Bedienung ist die „Simulation“ von Tastatureingaben. Generell sollte der Fokus über die SetFocus()-Methode von UIA auf das Eingabefeld gelenkt werden.

26.5.4.2 Poptext

Neben der üblichen Methodik, die Selektion über die Select()-Methode des SelectionItem Patterns zu verändern, kann ebenso die Methode SetValue() des Value Patterns benutzt werden. Handelt es sich dabei um einen editierbaren ***Poptext***, so wird, falls der Wert in der Poptext-Liste aufgeführt ist, eine Selektion vorgenommen und die entsprechenden *select*- und *activate*-Ereignisse erzeugt. Andernfalls wird der Inhalt des Editierbereichs umgesetzt und die Ereignisse *charinput* und *modified* ausgelöst.

26.5.4.3 Formatierte Eingabe

Wird bei einem Eingabefeld ein **Format** verwendet, so liefern Text und Value immer den Darstellungstext inklusive Formatierungszeichen. Ein Zugriff auf den Inhalt von Passwortfeldern bei einem verdeckten Format ist somit nicht möglich. Eine vollständige Änderung sollte ebenso über die SetValue()-Methode erfolgen und darf keine Formatierungszeichen beinhalten. Ansonsten sollten Testwerkzeuge die „Simulation“ der Tastatureingaben durchführen.

26.5.4.4 Client-Bereich

IDM-Objektklassen können aus mehreren UIA Elementen bestehen. Gruppierungsobjekte haben typischerweise ein übergeordnetes Element und einen Client-Bereich, welcher den Suffix „#client“ als AutomationId bekommt.

26.5.4.5 Virtuelle Bereiche

Gruppierungsobjekte erhalten einen virtuellem Bereich und optionale Scrollbars durch Setzen der Attribute *.vwidth* und *.vheight*. Das Client UIA Element erhält dann zusätzlich das Control Pattern Scroll und ScrollItem an den Kind-Objekten um so die Änderung des sichtbaren Bereichs und das Scrolling eines Kindes in den sichtbaren Bereich zu ermöglichen.

Durch die UIA-Methoden Scroll() bzw. ScrollIntoView() ist die Kontrolle über den angezeigten Ausschnitt möglich. Eine Scrolling-Kontrolle über die innen liegenden ScrollBar UIA Elemente ist nicht möglich.

26.5.4.6 Splitbox

Die einzelnen sichtbaren Bereiche sind als UIA-Kind-Elemente vom Typ Pane mit Namenssuffix „#client[1]“ … „#client[n]“ erreichbar und können durch die Resize() Control Pattern Methode in ihrer Größe geändert werden.

26.5.4.7 Menüs

Menüs in der Menüleiste wie auch Popup-Menüs sind nicht vom IDM überdefiniert sondern durch die Standard-Implementierung von Windows abgedeckt. Typischerweise sind die UIA Elemente der Menüs, Untermenüs und Menüeinträge erst beim Öffnen zugänglich. Kontextmenüs sind dabei unterhalb des Desktop UIA Elements eingeordnet. Die Menüs von Menüleisten liegen nach dem Öffnen direkt unter dem UIA Window Element.

26.5.4.8 Messagebox und Filereq

Beim **querybox()**-Aufruf werden diese durch Standard-Controls aufgebaut und sind typischerweise als UIA Control Type Dialog oder Window mit dem gesetzten Fenstertitel zu finden.

26.5.4.9 Image-Objekt

Um eine Erkennung des dargestellten Bildes zu erlauben, wird das ***Image***-Objekt in UI Automation durch das UI Element Button mit den Kind-Elementen Image und Text repräsentiert. Das Image-Element erhält, wenn vorhanden, den Alternativtext der verwendeten ***Tile***-Ressource in der UIA Name Property.

26.5.4.10 Toolbar-Objekt

Eine eingedockte ***Toolbar*** besitzt kein Transform Pattern, kann also nicht in Größe und Position variiert werden.

26.5.4.11 Treeview-Objekt

Eine erneute Selektion des gleichen Elements, über die Select()-Methode des SelectionItem Pattern, erzeugt kein erneutes *select*-Ereignis.

26.5.4.12 Benutzerinteraktion vs. Automatisierung

Die Funktionalität zur Automatisierung einer Benutzeroberfläche, die durch UI Automation möglich ist, deckt nicht alle Möglichkeiten ab, die einem Benutzer durch die Bedienung mit Maus und Tastatur gegeben sind.

- Tastatureingaben wie auch der Aufruf von ***Acceleratoren*** oder **Mnemonics** sind nicht direkt vorgesehen. Dies bedeutet aber auch, dass Aktionen bzw. Ereignisregeln, die z.B. auf `Enter` (`Return`), `Esc`, Cursortasten, usw. oder einer Funktionstaste basieren, nicht direkt durch UIA-Methoden angestoßen werden können. Es empfiehlt sich unter Umständen also die Simulation von speziellen Tasten um z.B. das Auslösen von *deselect_enter*-Ereignisregeln zu ermöglichen.
- Die Interaktionsmuster der Maus (Bewegen, Drücken und Loslassen einer Maustaste) sind ebenso kein Bestandteil der UIA-Methoden. Eine Interaktion in der Form „Poptext anklicken – mit gedrückter Maustaste nach unten fahren – über dem 3. Eintrag loslassen“ ist z.B. schlecht oder gar nicht simulierbar. Ein Mausklick entspricht meist der UIA-Methode Invoke() oder Select(), unter Umständen aber auch einem Expand() oder Collapse() und vieles mehr. Die Benutzerinteraktion ist also vom Control Type abhängig.

Grundsätzlich ist zu sagen, dass sich Benutzerinteraktion und Automatisierung bei den möglichen Aktionen, der Atomarität wie auch den Zustandsabhängigkeiten unterscheiden.

26.5.5 Hinweise zu HP UFT 12.52

Das Testwerkzeug HP UNIFIED FUNCTIONAL TESTING (UFT) bietet ab Version 12.52 eine Unterstützung für das Testen von UI Automation. Dabei weist UFT aber keine vollständige Unterstützung für alle UIA Control Types auf, sondern beschränkt sich auf die wichtigsten Control Types und Control Patterns. Dieses Kapitel ist also lediglich informativer Art und sollte keinen Blick in die UFT-Dokumentation ersetzen.

Die Unterstützung von UIA (UI Automation) durch UFT lässt sich wie folgt beschreiben:

- Das „UI Automation“ Add-In muss aktiv sein.
- Zur Identifikation über den Objekt-Spion muss der „UI Automation Modus“ aktiv sein.
- Die Objektidentifikation erfolgt über die UIA Properties Name und AutomationId.

» Es werden folgende UIA Control Types erkannt und in ein entsprechendes UFT-Objektmodell überführt:

UIA Control Type	UFT-Objektmodell
Button	UIAButton
Calendar	UIACalendar
CheckBox	UIACheckBox
ComboBox	UIAComboBox
Edit	UIAEdit
HyperLink	UIAHyperLink
List	UIAList
RadioButton	UIARadioButton
Slider	UIASlider
Tab	UIATab
DataGrid	UIATable
SplitButton	UIASplitButton
Window	UIAWindow
Table	UIATable
... andere ...	UIAObject

» Folgende UIA Control Patterns werden durch UFT über spezielle Methoden an den UIA-Objekten unterstützt:

UIA Control Pattern	UFT-Methoden
ExpandCollapse	.Expand .Collapse
Grid	.GetItem
Invoke	.Click
RangeValue	.Decrement .Increment .SetValue

UIA Control Pattern	UFT-Methoden
Scroll	.Scroll .ScrollDown .ScrollUp .ScrollLeft .ScrollRight .SetScrollPercent
ScrollItem	.ScrollIntoView
Selection	.Select .AddToSelection .RemoveFromSelection .GetSelection
SelectionItem	.Select .AddToSelection .RemoveFromSelection
Table	.GetRowHeaders .GetColumnHeaders
TableItem	.GetRowHeaderItems .GetColumnHeaderItems
Transform	.Move .Resize .Rotate
Toggle	.Set
Value	.SetValue
Window	.Maximize .Minimize .Restore .Close

Dies bedeutet, dass nicht alle IDM-Objektklassen „unterscheidbar“ als Testobjekte mit eigener UIA-Klasse von UFT erkannt werden. Dazu gehören die IDM-Objektklassen:

- ***toolbar***
- ***statusbar***
- ***scrollbar***
- ***splitbox***
- ***menubox***, ***menuitem***, ***menusep***

- ***spinbox***
- ***progressbar***
- ***image***
- ***rectangle***
- ***control***
- ***layoutbox***, ***groupbox***, ***splitbox***
- ***edittext*** (RTF)

Allerdings sollten diese von UFT als allgemeines UIAObject erkannt werden und durch die unterstützten Pattern-Methoden hinreichend test- und manipulierbar sein.

Es sei angemerkt, dass Microsoft mit UI Automation zwar die Bereiche „Accessiblity“ und „Test Automation“ abdeckt, allerdings keine vollständige und korrekte Erfassung von Benutzerinteraktion ermöglicht. Insofern ist ein „Recording“ der Benutzerinteraktion durch UFT im UI Automation Modus oft unvollständig und unter Umständen auch fehlerbehaftet.

Dies zeigt sich z.B. beim „Recording“ einer Menüinteraktion auf einem Untermenü. UFT generiert das folgende, nicht funktionsfähige Skript, welche keinen „Click“ auf das Menü enthält:

```
UIAWindow("Hauptfenster").UIAMenu("Anwendungsmenü").UIAObject("Datei").Expand
UIAWindow("Hauptfenster").UIAMenu("Datei").UIAObject("Untermenü").Expand
```

Ein funktionierendes Skript würde hingegen wie folgt aussehen:

```
UIAWindow("Hauptfenster").UIAMenu("Anwendungsmenü").UIAObject("Datei").Click
UIAWindow("Hauptfenster").UIAMenu("Datei").UIAObject("Untermenü").Click
UIAWindow("Hauptfenster").UIAMenu("Datei").Select "Untermenü;Menü A"
```

Beim „Recording“ im Windows-Modus erhält man etwa folgendes funktionstüchtiges Skript:

```
Window("Hauptfenster").WinMenu("Datei").Select "Datei;Untermenü;Menü A"
```

26.5.6 Erweiterung zur C-Schnittstelle

Die Funktionen **DM_Control()** und **DM_ControlEx()** wurden um eine zusätzliche Aktion ergänzt. Standardmäßig ist die Unterstützung von UI Automation aktiv. Mit dieser Aktion kann die spezielle Unterstützung des IDM für seine besonderen Objekte abgeschaltet werden. Weiterhin aktiv bleibt aber die UI Automation Unterstützung von Microsoft für die Standard Controls.

Die Umschaltung muss dabei vor dem Aufruf von **DM_Initialize()** und nach dem Bootstrapping erfolgen. Ein erfolgreiche Umsetzung wir mit der Rückgabe von *DM_TRUE* angezeigt.

Aktion	Argument
DMF_UIAutomationMode	*0* deaktiviert oder *1* aktiviert den UI Automation Support des IDM

26.5.7 Attribut .acc_text

	Bezeichner		Datentyp
Regelsprache	**.acc_text**		object, string
C	AT_acc_text		DT_Text, DT_String
COBOL	AT-acc-text		DT-Text, DT-String
Klassifizierung	Standardattribut		
Zugriff	get, set	Standardwert	*null*
changed-Ereignis	ja	Vererbung	ja

Mit diesem Attribut kann der Automation-Name überschieben werden, der für ein Oberflächenobjekt über die MICROSOFT UIA-Schnittstelle vom IDM erfragt wird. Bei einem Wert von *null* wird der Name normalerweise vom Windows Control bzw. durch die UIA-Unterstützung im IDM sinnvoll vorgegeben.

Unterstützung des Attributs durch Objekte

Objekt	Unterstützung des Attributs
filereq, ***messagebox***	Attribut hat **keine** Auswirkung
menubox, ***menuitem***, ***menusep***	
canvas, ***spinbox***, ***statusbar***, ***tablefield***	Attribut wird unterstützt
groupbox, ***notebook***, ***notepage***, ***splitbox***	
image, ***layoutbox***, ***window***	
rectangle, ***scrollbar***	
checkbox, ***pushbutton***, ***radiobutton***	
edittext, ***poptext***, ***statictext***	
control, ***listbox***, ***treeview***	
andere Objektklassen	Attribut wird **nicht** unterstützt

Anmerkung

Dieses Attribut ist nur für die automatisierte Fremdsteuerung mit aktiver MICROSOFT UIA-Unterstützung relevant. Unter QT und MOTIF hat dieses Attribut keine Funktion.

26.5.8 Attribut .acc_label

	Bezeichner		Datentyp
Regelsprache	**.acc_label**		object, string
C	AT_acc_label		DT_Text, DT_String
COBOL	AT-acc-label		DT-Text, DT-String
Klassifizierung	Standardattribut		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Mit diesem Attribut kann der Automation-Identifikator überschieben werden, der für ein Oberflächenobjekt über die MICROSOFT UIA-Schnittstelle vom IDM erfragt wird. Bei einem leeren String wird der Automation-Identifikator normalerweise vom Windows Control bzw. durch die UIA-Unterstützung im IDM sinnvoll vorgegeben.

Unterstützung des Attributs durch Objekte

Objekt	Unterstützung des Attributs
filereq, ***messagebox***	Attribut hat **keine** Auswirkung
menubox, ***menuitem***, ***menusep***	
canvas, ***spinbox***, ***statusbar***, ***tablefield***	Attribut wird unterstützt
groupbox, ***notebook***, ***notepage***, ***splitbox***	
image, ***layoutbox***, ***window***	
rectangle, ***scrollbar***	
checkbox, ***pushbutton***, ***radiobutton***	
edittext, ***poptext***, ***statictext***	
control, ***listbox***, ***treeview***	
andere Objektklassen	Attribut wird **nicht** unterstützt

Anmerkung

Dieses Attribut ist nur für die automatisierte Fremdsteuerung mit aktiver MICROSOFT UIA-Unterstützung relevant. Unter QT und MOTIF hat dieses Attribut keine Funktion.

Bei einer Überschreibung sollten die Regeln beachtet werden, welche in der MICROSOFT UI Automation Dokumentation für die AutomationId vorgegeben sind.

26.5.9 Erweiterung der Tile-Ressource

Für den Zugang zu IDM-Oberflächenobjekten über die UI-Automation Schnittstelle unter MICROSOFT WINDOWS wurde die ***Tile***-Ressource ergänzt. ***Image***-Objekte, welche eine ***Tile***-Ressource als Bild (Muster) nutzen, erhalten einen Alternativtext um die Automatisierung und Zugänglichkeit („Accessibility“) zu erleichtern.

Dieser Alternativtext in kann dabei wie folgt angegeben werden:

```
{ export | reexport } tile  <Bezeichner>  <x>, <y>,
      <Tilestring1>
  [ , <Tilestring> ]
{ scale } { text(<Alternativtext>) };

{ export | reexport } tile  <Bezeichner>  <Dateiname>
  { scale } { text(<Alternativtext>) };

<Alternativtext> := ( <String> | <Text-Objektpfad> )
```

Beispiele

```
text TxPause "Pause"
{
  2: "Break";
}
tile TiCoffeeBreak "kaffeetasse.gif" text("Pause");
tile TiCoffeeBreak "kaffeetasse.gif" text(TxPause);
```

26.5.10 Erweiterung des Tracing

Um eine Trace-Ausgabe der UIA-Schnittstelle des IDM FÜR WINDOWS zu aktivieren, muss das UI Automation Tracing explizit (zum Beispiel in der `on dialog start` Regel) eingeschaltet werden:

```
setup.tracing["AU"] ::= true;  /* "AU" = Automation */
```

27 A.06.01.d

27.1 Windows

12496: Bei **editierbaren Objekten** ohne Format wird bei Änderung von *.maxchars* im sichtbaren Zustand kein falscher Inhalt mehr dargestellt.

27.2 Qt

12651: **Gruppierungsobjekte** mit Größenraster berechnen ihre Größe nun wieder korrekt auf Basis des Rasters ihres Vaterobjekts.

12649: Verschiedene **einzeilige Objekte** mit Höhe *0* und *.yauto* = *-1* werden nun wieder korrekt in einer Rasterzeile vertikal zentriert.

12644: Bei einer editierbaren ***Spinbox*** mit Format kam es in bestimmten Situationen vor, dass das Format nicht angewendet wurde. Dies ist nun behoben.

12642: Eine ***Splitbox*** mit Größenraster speichert die Größen ihrer Splitbereiche in *.size[integer]* nun wieder in Rastereinheiten.

12640: Bei einer editierbaren ***Spinbox*** mit Format werden nun nach Fokusverlust der Inhaltsstring sowie *.curvalue* aktualisiert und nicht mehr auf den letzten Wert zurückgesetzt.

12634: Wenn bei einem ***Fenster*** mit Raster nur eines der Attribute *.minheight* und *.minwidth* gesetzt ist, werden nun korrekt Rasterwerte statt Pixelwerte verwendet. Die fehlerhafte Verwendung von Pixelwerten führte unter Umständen dazu, dass das Fenster die komplette Höhe oder Breite des Bildschirms ausfüllte.

12495: Die Performance des ***Treeviews*** wurde deutlich verbessert. Außerdem wurden Probleme hinsichtlich des initialen Setzens von *.topitem* korrigiert.

12319: Die Minimal- und Maximalgrößen der ***Splitbox***-Bereiche können durch interaktives Verkleinern bzw. Vergrößern nun nicht mehr unter- bzw. überschritten werden.

27.3 Kern

12654: Das Löschen von Attributen und Kindern wird nun korrekt im Attribut *.itemorder* widergespiegelt, unter Berücksichtigung von Vererbungseigenschaften.

Hinweis

Das Attribut *.itemorder* ist dafür vorgesehen, die statische Definitionsreihenfolge der ***Record***-Kinder zu liefern. Es ist nicht garantiert, dass es jederzeit und für jegliche dynamische Änderung von Attributen und Sub-Records die korrekte Reihenfolge der Kinder eines ***Records*** wiedergibt.

12637: Die Gültigkeitsprüfung der `return`-Anweisung prüft nun den vom Attributzugriff korrekt weitergegebenen Gültigkeitswert.

12628: Bei der **automatischen Konvertierung** von Werten in einen Ziel-Datentyp, wie sie z.B. bei Argumenten oder Variablenzuweisungen erfolgt, wird eine *Null-ID* nun wieder korrekt (und konform zum IDM 5) in einen Leerstring gewandelt.

12601: Bei einer Zuweisung an das Attribut *.state* einer ***Tristate-Checkbox*** wird der Zustand der Checkbox nun korrekt angezeigt. Betroffen waren die Plattformen QT und MICROSOFT WINDOWS.

27.4 COBOL

Behoben: Die Funktionen **DMcob_FatalAppError** und **DMcob_PutArgString** beachten nun *DM-setsep*.

Behoben: Die Funktion **DMcob_FatalAppError** steht jetzt auch unter MICROSOFT WINDOWS zur Verfügung.

Änderung: Die Einträge *DM-value-first* und *DM-value-second* gibt es jetzt auch als *DM-value-long-first* und *DM-value-long-second*, die als *PIC 9(9)* definiert sind, sodass auch Werte bis *65.535* möglich sind. Größere Werte werden vom IDM nicht unterstützt und führen zu einem Fehler.

Hinweis

Intern werden nun **DM_GetValueIndex** und **DM_SetValueIndex** an Stelle von **DM_GetValue** und **DM_SetValue** verwendet. Hierdurch ergeben sich leicht geänderte Trace-Ausgaben.

Neu: Wenn **BindThruLoader** zur Übergabe der Funktionsadressen benutzt wird, dann wird **gencobfx** nicht mehr benötigt und der Aufruf von **BindFuncs** kann entfallen (siehe Kapitel „BindThruLoader-Funktionalität“ im Handbuch „COBOL-Schnittstelle“).

Neu: Struktur **DM-ValueIndex**.

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

Kompatibilität

Alle COBOL-Module, die eine der **IDM Copy-Strecken** verwenden, müssen **neu kompiliert** werden.

27.5 Neue COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL

Es gibt eine neue COBOL-Schnittstelle für den IDM, die MICRO FOCUS VISUAL COBOL unterstützt. Für diesen COBOL-Compiler wird nun auch Unicode (UTF-16) voll unterstützt. Dadurch haben sich Änderungen an den Copy-Strecken ergeben, von denen alle COBOL-Varianten betroffen sind. Deshalb müssen alle COBOL-Module, die eine der IDM Copy-Strecken verwenden, neu kompiliert werden.

Die neuen Funktionen der COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL und die Unterschiede zur weiterhin erhältlichen, bisherigen COBOL-Schnittstelle des IDM sind im Kapitel „COBOL-Schnittstelle für Micro Focus Visual COBOL“ des Handbuchs „COBOL-Schnittstelle“ beschrieben.

27.6 Dynamisierte Funktionsaufrufe

27.6.1 Dynamische Anbindung von Record-Funktionen

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

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

Beispiel

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

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

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

on dialog start {
  if not Appl.active orelse not ApplCobol.active then
    print "Applikation(en) nicht korrekt angebunden!";
  elseif not Search("Udo", RecAdr) then
    print "Udo nicht gefunden, f\374ge ihn hinzu";
    RecAdr.Name := "Udo";
    RecAdr.Strasse := "Meisenweg 7";
```

```
    RecAdr.Ort := "Wolkenstein";
    RecAdr.PLZ := 71723;
    New(RecAdr);
  else
    print "Udo gefunden, wohnt in: "+RecAdr.Ort;
  endif
  exit(();
}
```

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

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

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

27.6.2 Codepage bei Anwendungsfunktionen

Die Vorgabe der genutzten Codepage für die Weitergabe und Verarbeitung von Strings in Anwendungsfunktionen bzw. für die darin enthaltenen Aufrufe von DM-Schnittstellenfunktionen, kann noch feiner gesteuert werden.

Über das Attribut *.codepage* am ***Application***-Objekt kann die Codepage-Vorgabe für alle Anwendungsfunktionen unabhängig von anderen Applikationen oder der globalen Vorgabe der Applikationscodepage definiert werden. Alternativ kann dies auch über die IDM-Schnittstellenfunktionen DM_Control bzw. DMcob_Control und DM_ControlEx mit der Aktion *DMF_SetCodePage* bzw. *DMF-SetCodePage* geschehen.

Außerdem kann durch eine zusätzliche Definition in einem **Funktionsalias** eine funktionsspezifische Codepage redefiniert werden.

27.6.3 Siehe auch

- Kapitel „Alias-Name mit optionaler Codepage-Definition“ im Handbuch „Regelsprache“
- Startoption +writeheader im Handbuch „Entwicklungsumgebung“
- Attribut codepage in der „Attributreferenz“

28 A.06.01.c

28.1 Kompatibilität

Das Binärformat hat sich geändert. Die Binärdateien müssen neu generiert werden.

28.2 Windows

12601: Das ***Checkbox***-Objekt reagiert nun auf eine Änderung von *.state* im sichtbaren Zustand.

12527: Monochrome Icons können nun angezeigt werden.

12418: Ein **inaktives *Fenster***, das ein ***Treeview***-Objekt enthält, konnte in den Vordergrund kommen, wenn der Mauszeiger über das Treeview-Objekt bewegt wurde. Die Ursache liegt in einem Problem des Microsoft Treeview-Controls, das einen Tooltip öffnet, um den Text eines Eintrags anzuzeigen, wenn dieser Text abgeschnitten ist. Normalerweise wird der Tooltip nicht geöffnet, wenn das Treeview-Objekt sich in einem inaktiven Fenster befindet, aber ab und zu passiert dies versehentlich. Das Problem kann der IDM nicht verhindern, es wird nun lediglich verhindert, dass das Fenster nach vorne kommt. Der Tooltip öffnet sich weiterhin.

12390: Zum Teil wurde nicht das am besten passende Bild aus einer **Icon-Datei** angezeigt. Es wurde stattdessen das erste Bild verwendet und dieses vergrößert bzw. verkleinert, wodurch es zu einem unschönen Aussehen kommen konnte. Beobachtet wurde das Problem zum Beispiel beim ***Image***-Objekt, wenn die ***Tile***-Ressource das Attribut *.scale* gesetzt hatte.

12384: Wenn der Mauszeiger über dem rechten bzw. unteren Rand eines ***Splitbox***-Objekts positioniert wurde, dann wurde fälschlicherweise der Verschiebezeiger angezeigt. Das Problem trat nur auf, wenn keine „Visual Styles“ aktiv waren (zum Beispiel in der klassischen Ansicht). Außerdem konnte man virtuell eine Splitbar verschieben, es erfolgte aber keine Änderung der Splitbereiche.

12340: Die großen IDM-Icons wurden zum Teil nach rechts rotiert (rechter Rand erscheint links) dargestellt. Dies lag daran, dass wohl das PNG-Format innerhalb einer Icon-Datei nicht wirklich unterstützt wird.

12338: Eine ***Tile***-Ressource, die als Muster definiert ist, wurde in einem Kindfenster falsch dargestellt, wenn sie als **Fenstericon** verwendet wurde. Aufgefallen ist das Problem im IDM-Editor.

Hinweis

Das Verkleinern bzw. Vergrößern einer **internen Bitmap** lieferte unterschiedliche Ergebnisse, je nachdem in welchem Objekt die interne Bitmap angezeigt wurde. Dies wurde jetzt angeglichen, hierdurch kann sich die Darstellung der internen Bitmap innerhalb von Menüeinträgen ändern. Es wird jetzt die Vordergrundfarbe bevorzugt.

Der interne Typ einer ***Tile***-Ressource, die als Muster definiert ist, musste sich ändern, sodass **Fenstericons** richtig dargestellt werden. Jetzt liefert **DM_GetToolkitData** ein Icon-Handle

(HICON) an Stelle eines Bitmap Handles, wenn das Attribut *AT_XTile* bzw. *AT_wsidata* an der ***Tile***-Ressource abgefragt wird.

28.3 Motif

12472: **Mnemonics** ignorieren die Groß- und Kleinschreibung nun korrekt, eine `Shift`-Betätigung für große deutsche Umlaute ist nicht nötig. Ein großgeschriebener deutscher Umlaut wird nun ausgelöst, auch wenn die Taste mit dem kleingeschriebenen Umlaut betätigt wird.

28.4 Qt

12529: *AT_open* am ***Treeview*** ist jetzt konsistent mit dem sichtbaren Zustand.

12478: Der ***Edittext*** berücksichtigt jetzt die vom Format vorgegebene Breite, wenn *.width = 0* gesetzt ist.

12467: Wenn ein ***Tile***, welches über ein Muster definiert wurde, einem ***Treeview*** oder einer ***Listbox*** zugewiesen wurde, so wurde das Bild nur im Normalzustand angezeigt. Bei Insensitivität und aktivem Zustand des jeweiligen Eintrags war das Bild verschwunden. Dies wurde nun behoben, sodass das Muster-***Tile*** in jedem Zustand angezeigt wird.

12465: Wenn an einem ***Tablefield*** initial alle Zellen unsichtbar gesetzt waren, kam es trotzdem zu einer falschen Fokussetzung, welche in einer Fehlermeldung (Dumpstate) des IDM resultierte. Ist die betreffende Zelle unsichtbar wird nun nicht mehr versucht initial den Fokus zu setzen. Außerdem wurde beim *focus*-Ereignis nicht immer die aktuell fokussierte Zelle mitgegeben. Dieser Umstand wurde auch behoben.

12461: Die ***Listbox*** hat jetzt die Option *opt_scroll_pixels* neu hinzubekommen. Diese Option wird nur von QT unterstützt. Standardmäßig ist diese Option deaktiviert. Bei aktiviertem *opt_scroll_pixels* wechselt die ***Listbox*** in den „Pixel-Scrollmodus“. Das heißt, es wird nicht mehr Item-weise gescrollt, sondern immer der nächste nicht sichtbare Bereich in den „Viewport“ geholt. Lange Items, die über den sichtbaren Bereich hinausgehen, können somit schrittweise gescrollt werden. Dabei beträgt laut QT-Regeln ein „PageStep“ nun die Pixel-Höhe des „Viewports“, ein „SingleStep“ beträgt immer die Pixel-Höhe des ersten Items.

12460: Bei der Selektion einzelner Tabellenzellen, bei denen *.fgc* bzw. *.bgc* gesetzt waren, gingen diese Farbinformationen verloren. Bei keinen gesetzten Farben erfolgt die Darstellung der Selektion weiterhin wie von System und QT-Stil vorgesehen. Sind Farben an einer Zelle gesetzt, so werden diese erhalten und getauscht. Ist nur eine Farbe nicht gesetzt, so wird hier die System- bzw. Stil-Farbe beibehalten.

Anmerkung zur Verwendung von Farben

Wenn keine Farben am ***Tablefield***-Objekt gesetzt sind, dann verwendet der IDM die Systemfarben, die dem Zustand zugeordnet sind.

Sind Farben gesetzt, werden diese verwendet. Bei einem aktiven, selektierten Feld bestimmt *.fgc* die Hintergrundfarbe und *.bgc* die Vordergrundfarbe. Ist nur eine Farbe gesetzt (*.fgc* oder *.bgc*) wird für die nicht gesetzte Farbe die entsprechende Systemfarbe verwendet

Bei einem aktiven Feld werden somit nicht Vordergrund- und Hintergrundfarbe getauscht, sondern die gesetzte Farbe anders zugeordnet. Es ist daher davon abzuraten, nur eine Farbe (*.fgc* oder *.bgc*) zu setzen. Die besten Ergebnisse erhält man, wenn entweder beide (*.fgc* und *.bgc*) oder gar keine Farben gesetzt werden.

Es kann unter Umständen bei einigen QT-Stilen zu Abweichungen vom oben beschriebenen Verhalten kommen. Auffällig ist hier zum Beispiel der Stil „GTK+“, wo die Farbgebung vom Stil vorgeschrieben ist.

12453: Die dynamische Umsetzung der Attribute *.style[style_lines]* und *.style[style_buttons]* am ***Treeview*** wirkt sich jetzt sofort aus.

12452: Das *select*-Ereignis an der ***Notepage*** kam nicht immer. Dies wurde nun behoben und das Verhalten ist nun analog zu MOTIF.

12423: Bei nur 2 Kindern innerhalb eines Gruppierungsobjekts wurde beim Umsetzen der Sichtbarkeit die Stackorder nicht korrekt wieder hergestellt, sodass das untere Kind danach über dem oberen lag.

12419: Das ***Listbox***-Objekt hat nach *.itemcount*-Änderung oder **:delete**-Aufruf den Listeninhalt nicht aktualisiert bzw. einen falschen Inhalt dargestellt. Der Inhalt wird nun aktualisiert.

12407: Das ***Image***-Objekt hat keine Warnmeldung ausgegeben, wenn dem *.picture*-Attribut ein String (Dateipfad) zugewiesen wurde. String-Werte für das *.picture*-Attribut sind unter QT obsolet. Es erfolgt jetzt eine Warnung mit Verweis auf die ***Tile***-Ressourcen.

Das ***Image***-Objekt unter QT unterstützt keine Strings (Dateipfade) am *.picture*-Attribut. Stattdessen sollten ***Tile***-Ressourcen verwendet werden.

12344: Das ***Tablefield*** hatte initial, beim Sichtbarschalten, nicht die eingestellte oberste Zeile bzw. Spalte dargestellt. Es wurde immer die 1. Zeile bzw. Spalte als oberste angezeigt. Da aber die Header an die richtige Position verschoben wurden, erschienen diese irgendwo in der Mitte.

Behoben: Beim Setzen von *AT_topitem* wurde das „Topitem“ nicht immer an die oberste Position korrigiert sondern teilweise nur irgendwo in den sichtbaren Bereich gebracht. Das „Topitem“ wird nun – sofern möglich – immer ganz oben angezeigt.

Behoben: Toplevel-Objekte berücksichtigen jetzt die im **Editor** gesetzte Arbeitsgröße und werden jetzt korrekt entsprechend dieser Arbeitsgröße positioniert.

Behoben: Der **Editor** kann jetzt die Rahmengrenzen für die Bildschirmauflösungen auch ohne aktiviertes Raster zeichnen. Außerdem sind die Beschriftungen mit den Größen jetzt besser lesbar wenn auch das Raster angeschaltet ist.

Behoben: Das „Topitem“ beim ***Treeview*** wird jetzt auch korrekt berechnet, wenn vor ihm einige Knoten geschlossen sind.

Behoben: Wurde am ***Treeview*** der Content über ein **Datenmodell** gesetzt, so wurden andere indizierte Attribute nicht korrekt gesetzt bzw. wiederhergestellt.

Behoben: Kein Scrolling mehr, wenn am ***Treeview*** ein *setval* auf einen indizierten *.content* erfolgt.

Behoben: Der ***Treeview*** hat in gewissen Konstellationen eine falsche Baumstruktur aufgebaut, was sich zum Teil auch auf Selektionen und das *.open*-Attribut ausgewirkt hat. Der Baumaufbau erfolgt jetzt analog zum IDM FÜR WINDOWS und IDM FÜR MOTIF. Außerdem wurde die Performance beim Befüllen des ***Treeviews*** deutlich verbessert.

Behoben: ***Treeview***-Abstürze bei **:delete()** und Key-Input beseitigt. Außerdem Performance bei **:delete()** sowie Wiederherstellung des „Topitem“ verbessert.

Änderung: Der ***Poptext*** im *listbox*-Stil nutzt jetzt die korrekte „PrefSize“ unter Berücksichtigung von *.itemcount* und *.showitem*. Außerdem wird die Breite jetzt wieder dem längsten Inhalt angepasst.

Die QT-Stile „Mac“ und „GTK+“ ignorieren *.showitem* für nicht-editierbare Poptexte (siehe Qt-Dokumentation).

Änderung: Der ***Poptext*** geht jetzt korrekt mit der Darstellung von **Mnemonics** um. Die Option *opt_mnemonic* wird nun beachtet.

28.5 Kern

12533: Zusätzliche, fehlerhafte Codepage-Konvertierungen von Werten, die über **DM_CallRule()** in den IDM kommen, werden nun vermieden.

12490: Dialoge, welche **Regel- oder Methodenargumente** mit einer Vorwärtsreferenz auf ein Objekt oder Modell als **Gültigkeitsbereich** besitzen, werden nun ohne Assfail gelesen und ausgeführt.

12488: Dialoge, welche **lokale Variablen** mit einer Vorwärtsreferenz auf ein Objekt oder Modell als **Gültigkeitsbereich** besitzen, werden nun ohne Assfail gelesen und ausgeführt.

12369 und 12370: Speicherleak beim Entfernen von benutzerdefinierten Attributen (assoziative Arrays oder Felder) beseitigt.

Neu (12356 und 12357): Um Konsistenz bei den Funktionen **find()**, **toupper()** und **tolower()** zu gewährleisten wurde eine **Locale-Unterstützung** eingebaut (siehe Kapitel „Locale-Unterstützung und die Auswirkungen auf String- und Sortierfunktionen“).

12296: Rechtschreibfehler „to many“ statt „too many“ in Regel-Interpreter-Fehlermeldung *„too many levels of recursion“* behoben.

12224: **Werteauflösung von Pfaden**, welche Attribute, Methoden oder exportierte Objekte aus anderen Modulen einführen, führen nun nicht mehr zu Fehlern beim **Binärschreiben**. Diese Korrektur führt dazu, dass Binärdateien neu erzeugt werden müssen, wegen einer inkompatiblen Änderung des Binärformats.

28.6 Editor

12464: Änderungen an ***Farbressourcen*** sollten nun vom Editor wieder umgesetzt und korrekt gespeichert werden.

12463: Im Editor wird nun keine ***Farbe*** mit Label „(-?-)“ angelegt. Ressourcen werden nun im Scope „OS_object“ nicht mehr angelegt.

12456: **Hilfe** ist vom Editor aus mehr als nur einmal unter WINDOWS startbar.

Änderung: Beim Anlegen von Objekten über den Dialog *„Create“* wird nun bei einer Klassen- oder Modellauswahl nur der schon eingegebene Bezeichner überschrieben, wenn dieser als geerbt markiert ist.

28.7 Debugger

12351: Darstellung in der *„Objektansicht“* für Hashes und Arrays sowohl bei Standard- als auch bei User-Attributen ist nun konsistent. Arrays mit fester Länge werden mit „[]“ bzw. mit „[,]“ dargestellt, Hashes immer mit Datentyp.

12350: Werte mit Objektreferenzen werden in der *„Objektansicht“* jetzt nicht mehr bezogen auf die aktuell debuggte Regel sondern bezogen auf das beobachtete Objekt, zu dem das Attribut mit der Referenz gehört, dargestellt. Außerdem werden jetzt auch bei der Wertermittlung alle Dialoge untersucht.

12343: Wenn man nacheinander Breaks in verschiedenen Regeln derselben Stacktiefe vollzogen hat, so wurden die **Informationen in der Statusbar** bezüglich Modul, Regel und Tiefe nicht aktualisiert und zeigten weiterhin veraltete Daten an. Informationen zur aktuell debuggten Regel werden nun korrekt aktualisiert und nicht nur auf Basis der Stacktiefe ermittelt.

12214: Unter UNIX wurden beim Beenden die debuggten Dialoge teils in unnutzbarem Zustand hinterlassen, abhängig davon ob man sich noch im Debugging-Modus befand oder im „Continue“ war. Nun wird der Dialog entweder mit dem Debugger geschlossen oder in benutzbarem Zustand hinterlassen.

8480: Wird der Debugger aus dem Editor heraus gestartet, so wird die gewählte Sprachoption jetzt mit an den Debugger übergeben. Somit laufen Editor und Debugger in derselben Sprache.

-IDMDBGargs "<Auflistung von IDM-Parametern>"

Über diesen optionalen Parameter kann dem Debugger beim automatischen Start eine Reihe von IDM-Parametern mitgegeben werden. Die IDM-Parameter werden aufeinanderfolgend als ein String angegeben und wirken sich nur auf den Debugger aus.

Beispiel

```
idm dialog.dlg -IDMdbg -IDMDBGargs "-IDMlanguage 2"
```

Startet den Debugger auf deutsch.

Neu: Die *„Objektansicht“* kann nun Werte scannen, welche Pfadausdrücke auf indizierte geerbte Kinder beinhalten, zum Beispiel *Wi.Gb.MPb:[3]*.

28.8 Beispiele

12318: Der Build der Qt-Demos brach mit Fehlern ab, wenn man nicht die USW-Option besitzt. Der Build wird nun ohne Fehler durchgeführt. Außerdem gibt es jetzt in der Readme-Datei einen Hinweis, wie man das QtDemo-Beispiel ohne USW-Option und ohne den USW-Teil bauen kann, um sich Qt-Eigenheiten wie Farbverläufe und die spezifischen Objekte anzuschauen.

28.9 Locale-Unterstützung und die Auswirkungen auf String- und Sortierfunktionen

Ab der Version A.06.01.c erfolgt in der Regelsprache die Umwandlung und Behandlung von Groß- und Kleinbuchstaben in Strings durch Systemfunktionen welche das „Locale“ (Gebietsschema, Zeichencodierung) der Anwendung berücksichtigen. Dadurch können z.B. deutsche Umlaute besser bei String-Vergleichen oder der Sortierung berücksichtigt werden.

Dies betrifft die folgenden Funktionen, Methoden und Ressourcen:

- **strcmp()**
- **toupper()**
- **tolower()**
- **sort()**
- **find()**
- **:find()**
- **applyfomat()**, ***format*** (über die Erweiterung des Muster-Formats)
- **DM_ValueChange()** (Sortierung von Strings sowie linguistische Sortierung)

28.9.1 Standardverhalten und sicherer Fallback

Der IDM hatte unter UNIX/LINUX schon immer implizit durch das verwendete Toolkit (MOTIF, QT) bzw. explizit für die DDM-Serverseite das Locale einer Anwendung über die Systemfunktion *setlocale(LC_ALL, "")* gesetzt, selbst aber aus Konsistenzgründen meist nur Systemfunktionen für die String-Behandlung im ASCII-Bereich verwendet. Ein expliziter oder inkompatibler Aufruf von **setlocale()** im Anwendungscode sollte besser vermieden werden.

Ab Version A.06.01.c erfolgt nun eine Aktivierung von Locale-spezifischen Systemfunktionen für die oben genannten IDM-Funktionalitäten standardmäßig über den **DM_Initialize()**-Aufruf. Bei Locale-Einstellungen die zu Problemen führen, erfolgt außerdem standardmäßig ein Warnhinweis und der Fallback auf die ASCII-konformen Systemfunktionen.

28.9.2 Empfehlungen

Es wird auf UNIX/LINUX-Plattformen empfohlen, die IDM-Anwendungen im UTF8-Locale zu betreiben. Da die WINDOWS-Plattform allgemein zugängliche UTF16-Systemfunktionen besitzt, ist hier vom

Anwendungsprogrammierer nichts weiteres zu beachten.

28.9.3 Erzwungene Locale-Benutzung

Unter UNIX/LINUX ist bei anderem Locale eine Konvertierung der internen Unicode-Strings in die jeweilige Zeichencodierung des Locale nötig. Nicht konvertierbare Zeichen werden dabei durch „?“ ersetzt. In einem UTF8-Locale entfällt diese Umcodierung. Die Umcodierung entfällt ebenso unter WINDOWS, da hier UTF16-Systemfunktionen vorhanden sind und verwendet werden.

Beispiel

Dialogdatei

```
// UTF8
dialog D
on dialog start
{
  print toupper("öl");
  print strcmp("ähm", "Ähm", true);
  print applyformat("EULA", "äÉX");
  exit();
}
```

Ergibt nun als Ausgabe:

```
"ÖL"
0
"ÄéX"
```

Lässt man den Dialog nun im C-Locale (*LC_ALL = C*) laufen, führt dies zur Anwendung des Fallbacks (siehe Kapitel „Standardverhalten und sicherer Fallback“):

```
[E: IDM APPLICATION REQUIRES UNICODE COMPATIBLE CHAR ENCODING - DISABLE
LOCALE TO AVOID UNDEF CHARS MAYBE REPLACED WITH ?]
"öL"
1
"X"
```

Achtung

Lässt man diesen Dialog nun im C-Locale (*LC_ALL = C*) mit einer selbstgebauten Anwendung und einer erzwungenen Locale-Benutzung („Erzwungene Locale-Benutzung“, explizit durch **DM_Initialize()** mit Option *DMF_EnforceLocale*) laufen, führt dies zum Ersetzen von Zeichen, die in der Zeichenkodierung des Locales ungültig sind, zum Fehlerzeichen „?“ und zu zusätzlichen Fehlermeldungen:

```
[E: Yiconv() can't convert char 0x0x00F6 to the system codepage]
"?L"
```

```
[E: Yiconv() can't convert char 0x0x00C4 to the system codepage]
[E: Yiconv() can't convert char 0x0x00E4 to the system codepage]
0
"X"
```

Wichtig

Das standardmäßige Setzen des Locale durch **DM_Initialize()** kann Auswirkungen auf die Anwendung haben, wenn Systemfunktionen verwendet werden die Locale-Abhängigkeiten aufweisen! Für die Regelsprache entstehen eher Vorteile da die String-Funktionen konsistenter mit Umlauten umgehen.

Für eine vollständige Rückwärtskompatibilität des IDM zu Vorgängerversionen kann die Option *DMF_InhibitLocale* bei Aufruf der **DM_Initialize()**-Routine angewendet werden.

Ebenso sollte bedacht werden, dass die Umsetzung von Groß- und Kleinbuchstaben bei aktivierter Locale-Unterstützung nicht vom IDM durchgeführt wird sondern von den Systemfunktionen. Unterschiede bei den Zeichenklassen wie auch der Behandlung von Groß- und Kleinbuchstaben zwischen verschiedenen UNIX/LINUX-Varianten wie auch zu WINDOWS-Systemen sind im Detail unvermeidlich.

Wenn Zeichen zwischen der internen Unicode-Codepage durch die Locale-Unterstützung nicht konvertiert werden können, meldet der IDM dies durch eine *„E:…“* Fehlermeldung in der Log-Datei. Durch die unvermeidliche Umsetzung auf das Fehlerzeichen „?“ entstehen unerwartete aber erklärbare Ergebnisse bei Benutzung der oben aufgelisteten Funktionen.

Zum Beispiel würde dann ein *toupper("abcÄÖÜß")* plötzlich *„ABC????“* liefern, wenn man den IDM im C-Locale betreiben würde, bei dem nur ASCII-Zeichen (7 Bit) erlaubt sind.

28.9.4 Ergänzungen der Funktion DM_Initialize()

Folgende Optionen sind erlaubt um die Locale-Unterstützung zu konfigurieren:

Option	Beschreibung
DMF_EnforceLocale	Es erfolgt die erzwungene Verwendung von Locale-spezifischen Systemfunktionen für IDM-Funktionalitäten welche Case-spezifische Operationen durchführen (siehe auch „Standardverhalten und sicherer Fallback“). Unter MICROSOFT WINDOWS wird mit dieser Optionen explizit *setlocale (LC_ALL, "")* aufgerufen, sodass konsistent zu UNIX/LINUX dann das Locale der C-Bibliothek gesetzt ist.
DMF_InhibitLocale	Es erfolgt kompatibel zu den Vorgängerversionen der A.06.01.c keine Nutzung des Locale, sondern die Verwendung von ASCII-Systemfunktionen. In Kombination mit *DMF_EnforceLocale* wird die Nutzung der Locale-Systemfunktionen deaktiviert wenn kein Unicode-Locale genutzt wird.

Der Aufruf der **DM_Initialize**-Funktion ohne eine der beiden Optionen aktiviert nun standardmäßig das Locale der Anwendung (dieses wird unter UNIX/LINUX typischerweise über die Umgebungsvariablen `LC_ALL`, `LC_CTYPE` usw. definiert und innerhalb der Applikation durch den Aufruf der Systemfunktion *setlocale(LC_ALL, "")*) für die Nutzung in String-Funktionen des IDM. Ist dieses Locale nicht Unicode-fähig, so erfolgt eine Warnmeldung in der Art:

```
IDM APPLICATION REQUIRES UNICODE COMPATIBLE CHAR ENCODING - DISABLE LOCALE TO
AVOID UNDEF CHARS MAYBE REPLACED WITH ?
```

Daraufhin wird sicherheitshalber die Locale-Verwendung im IDM nicht aktiviert. Der IDM funktioniert dann bezüglich den String- und Sortierfunktionen wie im C-Lokale (ASCII), also analog zur Vorgängerversion der A.06.01.c.

Für IDM-Applikationen, die konsequent für die Nutzung unter beliebiger Locale geschrieben wurden, kann über die Option *DMF_EnforceLocale* die Nutzung des Locale ohne obige Meldung verhindert werden. Dies hat Auswirkungen auf die Behandlung und Erkennung von Groß- und Kleinbuchstaben bei String-Funktionen oder bei der Sortierung mit dem *sort_linguistic*-Parameter. Unter Umständen hat dies auch den Verlust bzw. die Ersetzung von nicht kodierbaren Zeichen durch das Fehlerzeichen „?“ zufolge. Für einen sicheren Fallback empfiehlt sich die Kombination mit *DMF_InhibitLocale*.

Achtung

Das Setzen und Benutzen des Locale durch **DM_Initialize()** hat Auswirkungen auf Systemfunktionen, die Locale-spezifisch sind und hat damit Auswirkungen auf die Gesamtanwendung.

28.9.5 Ergänzungen der Format-Ressource

Eingabemuster

Die folgenden zusätzlichen Zeichen stehen innerhalb des Formatstrings zusätzlich zur Verfügung:

L	Alphabetische Zeichen, nur Kleinbuchstaben
W	Alphabetische und numerische Zeichen
P	Alle druckbaren Zeichen, also Buchstaben, numerische Zeichen, graphische Zeichen wie auch Zwischenraumzeichen und Satzzeichen
I	Identifikator-Zeichen – entspricht „W“ und zusätzlich dem Unterstrich „_“
J	Identifikator-Startzeichen – entspricht „A“ und zusätzlich dem Unterstrich „_“
Y	Identifikator-Startzeichen – entspricht „U“ und zusätzlich dem Unterstrich „_“

Erweiterte Zeichenklassenerkennung

Steht vor dem Eingabemuster das Erweiterungszeichen „E“, so erfolgt die Umsetzung in Groß- und Kleinbuchstaben sowie die Zeichenklassenerkennung als Unicode-Zeichen. Das heißt, dass bei aktivierter Locale-Unterstützung die Eingabemuster systembedingt deutlich mehr Zeichen in den diversen Zeichenklassen erkennen.

So erlaubt das Format *"EUAAA"* z.B. die Eingabe von vier Buchstaben, welche z.B. Umlaute oder Buchstaben aus anderen Sprachen sein können. Dabei muss der erste Buchstabe ein Großbuchstabe sein. Möchte man beliebige alphanumerische Zeichen bei der **verdeckten Eingabe** verwenden, so muss das „E“ hinter dem Spezialzeichen angegeben werden wie z.B. bei *"S*EWWWWWWWW"*.

28.9.6 Ergänzungen am Escaping von Unicode-Zeichen

Der IDM erlaubt nun die Ersatzdarstellung von Unicode-Zeichen, die über die 16-Bit-Kodierung von „\uXXXX“ hinausgehen. 32-Bit Unicodezeichen (UTF-32) sind durch die Form „\UXXXXXXXX“ möglich (dabei steht der Platzhalter „X“ für eine beliebige hexadezimale Ziffer zwischen *0 – 9* bzw. *A – F*). Die bisherige optionale Verwendung von „\UXXXX“ für UTF-16 Zeichen ist nur noch erlaubt, wenn dahinter keine hexadezimale Ziffer folgt.

28.10 Dokumentation

Die Dokumentation wird nun in den Formaten HTML und PDF ausgeliefert.

28.10.1 Systemvoraussetzungen

Die HTML-Dokumentation verwendet HTML5, CSS3 und JavaScript (ECMA-Script 5). Sie benötigt einen der folgenden Webbrowser:

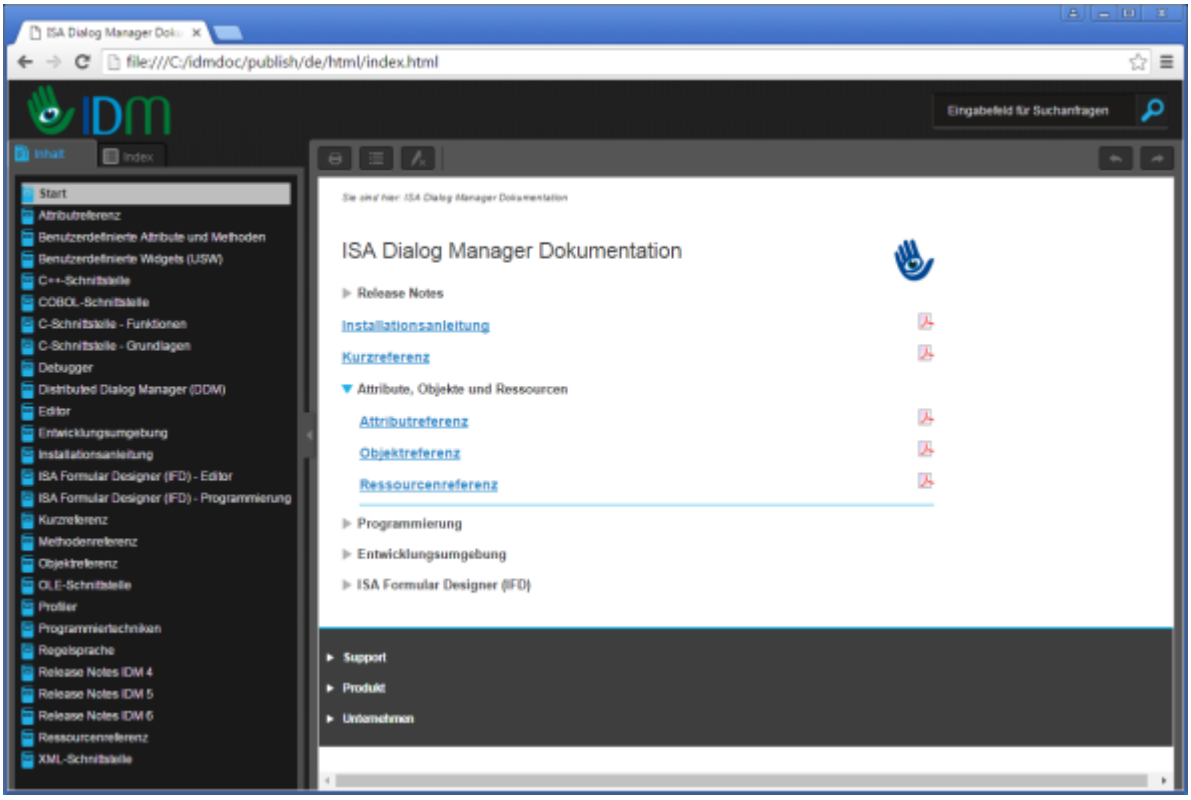
- Microsoft Internet Explorer ab Version 10
- Microsoft Edge
- Mozilla Firefox für Microsoft Windows und Unix/Linux ab Version 26
- Google Chrome für Microsoft Windows und Unix/Linux ab Version 29
- Opera für Microsoft Windows ab Version 15

Die PDF-Dokumente benötigen ein PDF-Anzeigeprogramm, das PDF-Version 1.4 oder höher darstellen kann.

Weiterhin gelten die Systemvoraussetzungen des IDM 6 auf der jeweiligen Plattform (siehe „Installationsanleitung“).

28.10.2 Bedienung der HTML-Dokumentation

Die HTML-Dokumentation wird durch Öffnen der Datei **index.html** im Verzeichnis *<IDM-Installationsverzeichnis>/doc/html* gestartet. Dies öffnet die Startseite der Dokumentation.



Auf der linken Seite befinden sich das Inhaltsverzeichnis und der durchsuchbare Index.Durch Klicken auf einen Eintrag wird das jeweilige Buch geöffnet und die entsprechende Hilfeseite angezeigt.

Rechts oben befindet sich ein Eingabefeld für Suchanfragen, mit dem die Dokumentation nach Begriffen durchsucht werden kann. Bei der Suche kann nach Teilwörtern mit einer Länge von mindestens 3 Buchstaben gesucht werden.

Die Schaltflächen über der angezeigten Hilfeseite haben folgende Funktionen:



Druckt die aktuell angezeigte Seite.



Erweitert oder minimiert alle auf- und zuklappbaren Bereiche auf der aktuellen Seite.
Diese Bereiche sind links durch einen Pfeil gekennzeichnet.



Löscht die Markierungen von Suchtreffern.
Diese Markierungen werden angezeigt, wenn man eine Ergebnisseite aus der Liste der Suchergebnisse öffnet.



Springt zum vorherigen Thema.
Dies kann eine Überschrift auf der aktuellen Seite oder eine andere Hilfeseite sein.



Springt zum nächsten Thema.
Dies kann eine Überschrift auf der aktuellen Seite oder eine andere Hilfeseite sein.

Hinweis

Wenn der Anzeigebereich schmaler als 800 Pixel ist, schaltet die Anzeige in eine für Tablets optimierte Darstellung und bei einer Breite kleiner als 480 Pixel in eine für Smartphones optimierte Darstellung.

28.10.3 Kontextsensitive Hilfe

Die HTML-Hilfe kann aus den Hilfemenüs von Editor, Debugger und Profiler aufgerufen werden.

Im Editor steht außerdem eine kontextsensitive Hilfe zu Objekten, Ressourcen, Attributen, Methoden, eingebauten Funktionen und vielen Anweisungen der Regelsprache zur Verfügung.

Die kontextsensitive Hilfe kann beispielsweise wie folgt aufgerufen werden:

- Durch Auswahl eines Elements und Drücken der Hilfe-Schaltfläche im Fenster *„Regelsprachschatz“*.
- Durch Auswahl eines Attributs und Drücken der Hilfetaste (`F1`) im *„Eigenschaftenbereich“*.
- Durch Auswahl eines Objekts, einer Ressource oder einer Applikation in der *„Werkzeugkiste“* und Drücken der Hilfetaste (`F1`).

Hinweis

In der englischen Dokumentation steht die kontextsensitive Hilfe derzeit nur für Objekte, Ressourcen, Attribute, Methoden und eingebaute Funktionen zur Verfügung.

29 A.06.01.b3

29.1 Kern

12422: Beim Laden von Binärdateien (Auflösung von Pfaden) kommt es nicht mehr zu Zugriffen auf uninitialisierten Speicher. Solche Zugriffe konnten zu Abstürzen beim Start des IDM-Editors und beim Erzeugen von Interfaces aus Binärmodulen führen.

30 A.06.01.b2

30.1 Windows

12432: Während der Installation kommt es beim Ersetzen einer älteren IDM-Version nicht mehr zu einem Abbruch mit der Fehlermeldung *„Cannot migrate Feature States“*. Der Fehler trat auf, wenn in der älteren Version Funktionen installiert waren, die es in der neuen Version nicht mehr gibt.

12400: Die Installation entschlüsselt jetzt alle Dateien, sodass keine Dateien mehr als Testversionen für „ISA“ installiert werden.

Hinweis

Für die Installation des IDM FÜR WINDOWS sind Administrationsrechte erforderlich.

12387: Probleme bei der Registrierung von IDM-Symbolen für Dateierweiterungen bei der Installation für alle Benutzer wurden behoben. Der Status der entsprechenden Checkbox stimmt nun mit der tatsächlichen Auswahl überein. Außerdem wird im Abschlussdialog der Installation ein Hinweis angezeigt, wenn die Registrierung der Dateierweiterungen möglicherweise durch Einstellungen des aktuellen Benutzers überschrieben werden.

12386: Das Erkennen und Ersetzen von Vorgängerversionen bei der Installation wurde verbessert. Die Ersetzung wird nun automatisch ausgewählt, wenn die neu zu installierende Version höher ist als die bereits installierte Version, die Architektur beider Versionen übereinstimmt und **kein** Wechsel auf ein höheres Major Release vorliegt. In allen anderen Fällen wird eine Ersetzung nicht unterstützt. Die Ersetzung wird dann nicht ausgewählt und ein entsprechender Hinweis wird angezeigt. Eine A- oder V-Versionen ist immer höher als eine T-Version. Eine T-Version ist immer höher als eine X-Version.

Hinweis

Die neue Version darf nicht im selben Verzeichnis wie die Vorgängerversion installiert werden. Wird dies gewünscht, sollte vor der Installation der neuen IDM-Version die Vorgängerversion deinstalliert werden.

12385: Das Ändern des Startmenüeintrags bei der Installation funktioniert nun.

12382: Im Fenster *„Installationsart auswählen“* der deutschen Installation ist die Beschriftung „vollständig“ nun auch lesbar, wenn die klassische Desktop-Darstellung eingestellt ist.

12379: Die Installation zeigt im Abschlussdialog jetzt eine Warnung an, wenn die Umgebungsvariable `IDM_HOMEDIR` gesetzt ist.

Achtung

Seit Version A.06.01.a setzt die Installation des IDM FÜR WINDOWS die Umgebungsvariable `IDM_HOMEDIR` nicht mehr. Eine eventuell noch vorhandene Umgebungsvariable `IDM_HOMEDIR` sollte entfernt werden.

Prinzipiell ist bei Verwendung der Umgebungsvariablen `IDM_HOMEDIR` darauf zu achten, dass sie auf die eingesetzte IDM-Version zeigt. Ansonsten kann es zu Fehlfunktionen bis hin zu Abstürzen kommen.

12378: Die Installation legt nun wieder einen Startmenüeintrag *„IDM-Version“* an.

12377: In der deutschen Installation wurden noch vorhandene englische Beschriftungen durch entsprechende deutsche Beschriftungen ersetzt.

12376: In der Datei **readme.txt** im Installationsverzeichnis der DVD sind die Installationspakete nun korrekt mit der Dateiendung „msi“ aufgeführt und die neue S-Lizenz (Server-Lizenz) wurde ergänzt.

30.2 Motif

12427: Das mehrfache Anzeigen durchgezogener Carets wird nun vermieden. Hierzu wurde die Reihenfolge der Funktionsaufrufe zum Setzen von Cursorposition und Selektion umgestellt.

30.3 Kern

12412: Die Performance des dynamischen **destroy()** für lokale Objekte ohne große Väterhierarchie und Objekte aus anderen Modulen wurde verbessert.

12398: Beim Einlesen von Dialogen oder Modulen mit Objekten, die eine **geerbte Callback-Funktion** besitzen, kommt es nicht mehr zu Fehlern, die zu einem Objekt- oder Speicherleck führen konnten.

12375: In der Datei **IDMuser.h** ist jetzt wieder der korrekte DM_VERSION_STRING enthalten. Dadurch wird bei der Windows-Installation wieder die richtige Version angezeigt.

30.4 USW

12394: Unter MICROSOFT WINDOWS wird jetzt überprüft, ob die USW-Option lizenziert ist oder ob es sich um eine Testlizenz handelt. Eine nicht zur IDM-Version gehörende USW-Bibliothek wird deshalb zurückgewiesen und die USW-Option in diesem Fall nicht aktiviert. Damit wird ein Absturz vermieden, der bei dem Versuch auftrat, eine unpassende DLL der USW-Option zu laden. Dies konnte zum Beispiel bei einer falsch gesetzten Umgebungsvariable `IDM_HOMEDIR` passieren.

Hinweis für Microsoft Windows

Bei der USW-Option kann es zu Fehlermeldungen kommen, wenn versucht wird, eine falsche C-Laufzeitumgebung zu laden. Dies tritt auf, wenn die ausgewählte USW-Option zu einer anderen IDM-Version gehört und eine andere C-Laufzeitumgebung verwendet.

Der IDM kann diese Fehlermeldung nicht abfangen. Zur Vermeidung des Fehlers muss sichergestellt sein, dass die zu einer IDM-Version gehörende **idmusw.dll** verwendet wird. Dabei ist die Reihenfolge der Verzeichnisse zu berücksichtigen, in denen der IDM nach der USW-Bibliothek

sucht. Unter anderem ist darauf zu achten, dass die Umgebungsvariable `IDM_HOMEDIR` entweder undefiniert oder richtig gesetzt ist.

31 A.06.01.b

31.1 Windows

12257: Die **XML-Schnittstelle** unternimmt nun weitere Versuche ein MSXML-Objekt zu laden, nachdem das Laden einer anderen Version des MSXML-Objekts fehlgeschlagen ist.

Hinweis

Bei jedem erfolglosen Ladeversuch wird nun eine Fehlermeldung ausgegeben, dass MSXML nicht geladen werden konnte.

12253: Beim ***Tablefield*** wird das unnötige Neuzeichnen von Feldern vermieden, welches zu einem Flackern der Felder führen konnte, wenn man die Maus über das ***Tablefield*** bewegte. Das Problem wurde bei deaktiviertem „Aero Style“ unter WINDOWS 7 beobachtet.

12240: Die Kindobjekte einer ***Layoutbox*** mit *.wrap = false* werden beim Scrollen nicht mehr neu angeordnet.

Bei Attributänderungen der ***Layoutbox***, die Einfluss auf das Layout haben (z. B. *.direction*, *.xmargin*, *.ymargin*), werden die Kindobjekte unabhängig von *.wrap* neu angeordnet. Außerdem werden die Kinder neu angeordnet, wenn die Sichtbarkeit oder Größe von Kindern geändert wird. Dies stellt sicher, dass die Kindobjekte sich nicht überlappen und dass sich keine Inkonsistenzen ergeben, wenn die Layoutbox unsichtbar und wieder sichtbar geschaltet wird.

12192: Ein **mehrzeiliger *Edittext*** (*.multiline = true*) bricht jetzt den dargestellten Text neu um, wenn das Attribut *.hsb_visible* im sichtbaren Zustand geändert wird.

12159: Ein ***Scrollbar***-Objekt mit *.arrows = false* reagiert nun sofort auf das dynamische Ändern der Hintergrundfarbe.

Anmerkung

Scrollbar-Objekte mit *.arrows = false* verwenden die Hintergrundfarbe ihres Vaterobjekts und nicht ihre eigene.

12152: Drag-and-Drop beim ***Edittext*** funktioniert wieder korrekt. Es konnte vorkommen, dass beim Drag-and-Drop der Text übertragen wurde, der bei Fokuserhalt des ***Edittexts*** markiert war. Auch konnte es vorkommen, dass gar kein Drag-and-Drop initialisiert wurde (*.startsel = .endsel*) oder dass sich mit der Maus die Markierung nicht ändern ließ. Diese Probleme sind behoben.

12091 und 12012: Bei einem maximierten und sichtbaren ***Fenster*** ist die Anordnung der Kindobjekte nun auch dann korrekt, wenn alle Positions- und Größenattribute des Fensters unmittelbar nacheinander geändert werden. In bestimmten Situationen kam es vor, dass die Kinder auch im maximierten Fenster entsprechend der Geometrie des nicht maximierten Fensters angeordnet wurden.

12059: ***Pushbuttons***, die ein Popup-Menü haben und dynamisch sichtbar gemacht werden, beachten nun den gesetzten Font.

12053: Der Fokus verbleibt nun auf demselben Kindobjekt wenn ein Gruppierungsobjekt (***Groupbox***, ***Layoutbox***, ***Notepage***…) wegen einer Attributänderung intern neu angelegt werden muss.

12035: Bei einem ***Poptext*** mit *.style = edittext* wird die Scrollposition des Inhalts jetzt korrekt aktualisiert, wenn das Objekt vergrößert wird.

12019: Der angebundene ***Edittext*** wird vom ***Tablefield*** nun auch dann angezeigt, wenn die Zuweisung an das *.edittext*-Attribut erfolgt, nachdem der ***Edittext*** unsichtbar gemacht wurde.

12013: Ein ***MDI-Kindfenster*** mit *.mapped = false* wird jetzt auch dann nicht angezeigt, wenn beim Anlegen ein anderes MDI-Kindfenster maximiert ist. Das Problem entstand, weil Fenster mit *.mapped = false* aktiviert wurden, was im Allgemeinen aber kein Sichtbarmachen des Fensters bewirkte.

Änderung (11868): **Drag-and-Drop** startete zu schnell, weil der Systemwert „DragDelay“, der bisher die Startverzögerung definierte, ab WINDOWS 7 nicht mehr existiert. Nun wird die Doppelklickgeschwindigkeit als Verzögerungswert herangezogen, falls „DragDelay“ nicht vorhanden ist.

11769: Bei einem ***Popup-Menü*** gehen nun auch dann keine *select*-Ereignisse mehr verloren, wenn das Vaterobjekt direkt nach dem Schließen des Popup-Menüs unsichtbar gemacht wird. Das Problem konnte beobachtet werden, wenn das Vaterobjekt direkt nach dem Aufruf der Funktion **openpopup()** unsichtbar gemacht wurde.

31.1.1 Installation

31.1.1.1 Registrierung von Dateierweiterungen

Bei der **Installation** des IDM FÜR WINDOWS werden nun Dateisymbole für Dateien mit den Erweiterungen „.dlg“, „.idm“ und „.mod“ registriert.

Im Fenster *„Funktionen auswählen“* der Installation kann eingestellt werden, ob die Dateierweiterungen registriert werden sollen. Man erreicht dieses Fenster, wenn man im Fenster *„Installationsart auswählen“* die Option *„Anpassen“* auswählt.

Es wird ein Hinweis angezeigt, falls hierbei Registrierungen anderer Anwendungen überschrieben werden. Die Registrierung der Dateierweiterungen kann nicht rückgängig gemacht werden.

Um die Dateierweiterungen für alle Anwender zu registrieren, muss die Installation von einem Administrator ausgeführt werden.

31.1.1.2 Merge-Module für die Redistribution entfallen

Es werden keine Merge-Module für die Redistribution von IDM-Anwendungen mehr ausgeliefert. Die für die Redistribution benötigten Komponenten können ohne besondere Vorkehrungen direkt aus dem Installationsbaum verwendet werden.

31.2 Unix

Behoben: Der Desktop-Eintrag für die IDM-Hilfe ist nun korrekt mit dem Skript **idmhelp** verknüpft.

31.3 Motif

12275: Minimale und maximale Größen eines ***Fensters*** werden nun unter Berücksichtigung des Attributs *.borderraster* korrekt und konsistent gesetzt.

12198: Die IDM-Heuristik zur Korrektur der **Z-Ordnung** von Fenstern wurde verbessert, um die Überdeckung einer ***Messagebox*** oder eines ***Filerequestors*** durch andere Applikationsfenster zu unterbinden, falls Desktop oder Fenstermanager dies nicht leisten können. Beim Aufruf der Funktion **querybox()** mit einem *Parent*-Parameter wird dieser nun wieder weitergereicht.

Moderne Desktops und Fenstermanager haben immer wieder Probleme, das Vorschieben eines Applikationsfensters vor ein modales Fenster (z. B. eine ***Messagebox***) zu unterbinden, wenn mehrere Applikationsfenster vorhanden sind. Der IDM FÜR MOTIF versucht dies über die korrekte Angabe der Modalität (applikationsmodal) und durch eine Heuristik zur Korrektur der Z-Ordnung von Fenstern sicherzustellen. Allerdings kann der IDM nicht alle Situationen korrigieren, weil einige moderne Desktops und Fenstermanager in speziellen, modalen Situationen keine *VisibilityNotify*-Ereignisse weitergeben.

12135: Wenn bei einem ***Edittext*** *.startsel* und *.endsel* nicht gesetzt wurden, konnte ein Ändern von *.content* dazu führen, dass der Cursor (Caret) an mehreren Stellen durchgezogen gezeichnet wurde. Dies tritt nun nicht mehr auf.

Anmerkungen

- Das Umsetzen von *.content* oder *.navigable* an einem ***Edittext*** kann Einfluss darauf haben, ob in einem anderen Edittext der Cursor (Caret) durchgezogen gezeichnet wird. Zu beachten ist, dass bei einem Edittext mit *.navigable = false* unter Motif keine Eingaben möglich sind.
- Der durchgezogene Cursor erlaubt keinen Rückschluss auf Navigationsreihenfolge oder Fokussierung eines ***Edittexts***.

11949: Es kommt nicht mehr zu einem kurzzeitigen Stillstand der Anwendung wenn in einem ***Poptext*** mit Stil *edittext* der Cursor mit `Tab` umgesetzt wird. Außerdem blinkt nun der Cursor initial wieder.

31.4 Qt

Behoben: Beim **mehrzeiligen *Edittext*** wird nun nicht mehr das Standard-Kontextmenü von Qt sondern das im IDM definierte Kontextmenü angezeigt. Das Standard-Kontextmenü von Qt wird nur noch angezeigt, wenn im IDM für den Edittext kein Kontextmenü definiert wurde.

31.5 Kern

12331: Die eingebaute Funktion **atoi()** interpretiert die Zeichen „A“–„F“ und „a“–„f“ wieder als Buchstaben. Analog zum IDM 5 gibt die Funktion einen Fehler zurück, wenn das übergebene Argument leer ist oder Zeichen enthält, die keine Ziffern sind.

12323: Methodenaufrufe innerhalb des Index einer Variablen funktionieren nun auch auf der linken Seite eines Ausdrucks.

12223: Beim Binärschreiben von benutzerdefinierten Attributen wird kein geerbter Gültigkeitsbereich (Scope) mehr herausgeschrieben. Das Herausschreiben des Gültigkeitsbereichs führte zu Fehlermeldungen, wenn dieser nicht bekannt war, weil er in einem anderen Modul definiert war.

12070: Die eingebaute Funktion **inherited()** liefert nun auch für geerbte und an der Instanz unveränderte Defaultwerte von assoziativen Arrays den korrekten Wert.

12067: Für benutzerdefinierte, assoziative Arrays wird *.itemcount* nun auch in den besonderen Konstellationen korrekt ermittelt, bei denen der Defaultwert erneut gesetzt wird und die Vererbungskette Modelle ohne gesetzte Werte enthält. Dadurch wird ein Absturz der Methode **:index()** vermieden, der durch Zugriff auf einen Index außerhalb des Indexbereichs zustande kam.

11933: **Rekursive Aufrufe** von benannten Regeln, benutzerdefinierten oder überschriebenen Methoden erzeugen keinen ASSFAIL mehr, wenn die maximale Rekursionstiefe von 1.000 überschritten wird. Das Überschreiten der maximalen Rekursionstiefe wird nun durch ein `fail` (Evaluation Error) angezeigt.

Hinweis zum IDM für Windows

Um die maximale Rekursionstiefe ohne **Exception** zu erreichen, muss die **Stackgröße** erhöht werden. Eine Größe von 8 MByte (8.388.608 Byte) wird empfohlen.

Die Stackgröße kann mit der Linkeroption **/STACK:<size>** gesetzt werden. In VISUAL STUDIO kann man den Wert in den Projekteigenschaften unter *„Konfigurationseigenschaften“* → *„LinkerSystemStapelreservierungsgröße“* setzen.

11087: Das mehrfache Laden von binären Modulen, die sowohl von einem Dialog bzw. Modul, als auch von dessen importierten Modulen importiert werden, wird nun vermieden.

Änderung: Wird der eingebauten Funktion **join()** als erstes Argument der Datentyp *string* übergeben, dann werden die nachfolgenden Argumente in den Datentyp *string* konvertiert und anschließend verkettet. Sind die nachfolgenden Argumente Sammlungen, werden zunächst deren Werte in der natürlichen Reihenfolge des Index zu einem String verkettet.

Falls das erste Argument ein Datentyp, aber weder *string* noch ein Sammlungsdatentyp ist, gibt **join()** einen Fehler zurück.

Unverändert: Die eingebaute Funktion **join()** führt eine Mischoperation unter Verwendung vorhandener Indizes der Argumente aus, wenn Resultate mit den Datentypen *hash* und *matrix* erzeugt werden. Dadurch kommt es zum Überschreiben von Werten, wenn unter den Argumenten in mehreren Sammlungen dieselben Indexwerte verwendet werden.

Wenn mit **join()** ein *hash* erzeugt wird, dann werden beim Zusammenführen von Sammlungen deren Indexwerte in das Resultat übernommen. Elemente aus Sammlungen, die in der Argumentliste weiter hinten stehen, überschreiben Elemente mit identischen Indizes aus Sammlungen, die in der Argumentliste weiter vorne stehen.

Beim Erzeugen einer *matrix* behalten Elemente, deren Index vom Datentyp *index* ist, im Ergebnis denselben Indexwert, den sie auch im Argument haben. Sammlungen, deren Indizes nicht vom Datentyp *index* sind, werden vor dem Zusammenführen unabhängig voneinander in den Datentyp *matrix* konvertiert. In ihnen können dadurch Elemente mit denselben Indexwerten wie in anderen Argumenten entstehen. Beim Zusammenführen werden Elemente aus weiter vorne übergebenen Sammlungen durch Elemente aus weiter hinten übergebenen Sammlungen überschrieben, wenn die Elemente dieselben Indexwerte haben.

Um die Mischoperation zu vermeiden, kann man die Argumente zuerst zu einer Liste mit dem Datentyp *list* zusammenführen, bevor man einen *hash* oder eine *matrix* erzeugt.

Behoben: Bei der Methode **:calldata()** wurden die Parameter-Informationen und bei den Attributen *.colorname[]*, *.cursorname[]*, *.fontname[]* die Index-Informationen korrigiert. Sie werden im IDM ECLIPSE PLUGIN jetzt korrekt angezeigt.

31.6 Debugger

12337: Im Eingabefeld für Ausdrücke und Anweisungen funktioniert jetzt der Befehl *„Print“* aus dem Menü *„Debug“* und dem Kontextmenü. Genauso wie für eine Markierung in der Regelanzeige wertet der Befehl *„Print“* den Inhalt des Eingabefelds aus und schreibt das Ergebnis in das Ausgabefenster.

12325: Wenn die Protokoll-Angabe beim Starten des Debuggers mit der Option **-IDMDBG** fehlerhaft ist, wird jetzt eine Fehlermeldung als Dialogfeld angezeigt und ins Tracefile geschrieben. Vorher erfolgte lediglich eine Ausgabe des Fehlers ins Tracefile.

12251: Die Regelsuche (*„Find…“*) wurde umfassend überarbeitet:

- Das Fortsetzen der Suche (*„Find…“*) nach dem ersten Treffer funktioniert wieder.
- Änderungen des Suchstrings werden korrekt berücksichtigt.
- Die Suche beginnt jetzt bei der im Regelbrowser ausgewählten Regel.
- Bei Erreichen des Endes (Suchrichtung *„Down“*) bzw. Anfangs (Suchrichtung *„Up“*) des Dialogs oder Moduls wird die Suche an dessen Anfang bzw. Ende fortgesetzt.
- Das Suchen kann nun auch durch Drücken von `Return` im Eingabefeld *„Find what“* ausgelöst werden.
- Das Suchfenster bleibt nun im Vordergrund.

12250: Das Setzen von **Breakpoints** nach dem Entladen eines Dialogs oder Moduls erzeugt keine Fehlermeldung mehr. Vor der Korrektur wurden neue Breakpoints vermeintlich nicht gesetzt und nicht angezeigt, die Programmausführung wurde allerdings an ihrer Position angehalten.

Die Verwaltung der Breakpoint-Liste wurde korrigiert sodass sie nun nach dem Entladen von Dialogen und Modulen aktualisiert wird und nur noch gültige Breakpoints enthält.

Behoben: In vielen Konstellationen (besonders unter MICROSOFT WINDOWS) erschienen im IDM 6 Dialogfelder mit Fehlermeldungen nicht mehr. Diese werden nun wieder angezeigt.

31.6.1 Starten des Debuggers

Beim Start des Debuggers ist die Angabe eines Servicenamens für die DDE-Verbindung (MICROSOFT WINDOWS) bzw. eines Ports für die TCP/IP-Verbindung (UNIX) jetzt optional.

„Tabelle 2“ zeigt die Möglichkeiten, den Debugger manuell zu starten, „Tabelle 3“ zeigt die Möglichkeiten für den automatischen Start des Debuggers. Für beide Tabellen gilt, dass bei den Kommandos in der letzten Zeile der angegebene Servicename (DDE) bzw. Port (TCP/IP) verwendet wird. Bei allen anderen Kommandos wird vom IDM automatisch ein Servicename (DDE) bzw. freier Port (TCP/IP) gewählt.

Tabelle 2: *Befehle für den manuellen Start des Debuggers*

DDE (Microsoft Windows)	TCP/IP (Unix)
`idmdbg`	`idmdbg`
`idmdbg -service`	`idmdbg -port` `idmdbg -port 0`
`idmdbg -service <serviceName>`	`idmdbg -port <port>`

Tabelle 3: *Befehle für den automatischen Start des Debuggers*

DDE (Microsoft Windows)	TCP/IP (Unix)
`idm -IDMDBG DDE <dialog.dlg>`	`idm -IDMDBG TCPIP <dialog.dlg>`
`idm -IDMDBG DDE: <dialog.dlg>`	`idm -IDMDBG TCPIP: <dialog.dlg>` `idm -IDMDBG TCPIP:0 <dialog.dlg>`
`idm -IDMDBG DDE:<serviceName> <dialog.dlg>`	`idm -IDMDBG TCPIP:<port> <dialog.dlg>`

Der Debugger kann jetzt auch über die Option **-IDMdbg** (man beachte die Groß- und Kleinschreibung) ohne Angabe weiterer Parameter gestartet werden.

```
idm -IDMdbg <dialog.dlg>
```

Die Kommunikation erfolgt dabei automatisch via DDE (MICROSOFT WINDOWS) bzw. TCP/IP (UNIX). Servicename bzw. Port werden vom IDM automatisch gewählt.

Das Starten mit der Option **-IDMDBG** ist weiterhin möglich:

```
idm -IDMDBG DDE{:{<serviceName>}} <dialogl.dlg>
```

bzw.

```
idm -IDMDBG TCPIP{:{<port>}} <dialog.dlg>
```

Angaben in {…} sind optional und werden vom IDM automatisch gewählt, wenn sie fehlen.

31.6.2 Objektansicht (Object View)

In der *„Objektansicht“* (*„Object View“*) des Debuggers können Objekte mit ihren vordefinierten und benutzerdefinierten Attributen angezeigt und die Attributwerte während des Debuggens überwacht und manuell geändert werden. Die *„Objektansicht“* befindet sich im unteren Bereich des Debuggers auf einer eigenen Registerkarte *„Objects“*, neben den Registern der überwachten Ausdrücke (*„Expressions“*) und Variablen (*„Variables“*).

Die *„Objektansicht“* unterteilt sich in die *„Objektliste“* auf der linken Seite und die *„Objektanzeige“* auf der rechten Seite. Die *„Objektliste“* besteht aus einem Suchfeld oben und der Ergebnisliste darunter. Die *„Objektanzeige“* enthält für jedes überwachte Objekt eine Registerkarte mit jeweils einer Attributtabelle und einem Eingabefeld für einen Filterausdruck über der Tabelle. In der *„Objektanzeige“* können bis zu 3 Objekte gleichzeitig überwacht werden.

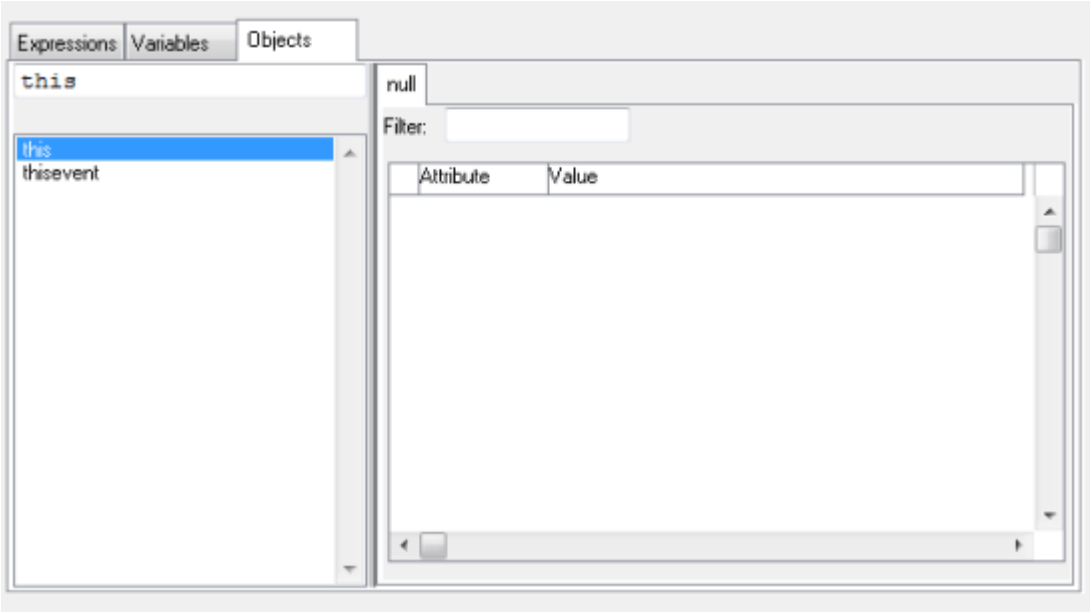


Abbildung 7: *Objektansicht im Initialzustand*

Beim Start des Debuggers ist das Suchfeld der Objektliste mit *„this“* vorbelegt. In der Ergebnisliste werden das `this`-Objekt, das `thisevent`-Objekt und eventuell weitere Objekte mit „this“ im Namen angezeigt. Die *„Objektanzeige“* ist zunächst leer, da noch kein Objekt zur Überwachung ausgewählt wurde.

31.6.2.1 Objekte suchen

Im Suchfeld der *„Objektliste“* können Objektnamen (Label), Objektpfade oder Objektklassen (***window***, ***edittext***, ***record***…) vollständig oder teilweise eingegeben werden. Objekte, die dem Suchausdruck entsprechen, werden dann in der Ergebnisliste unter dem Suchfeld angezeigt, wobei der beste Treffer markiert wird.

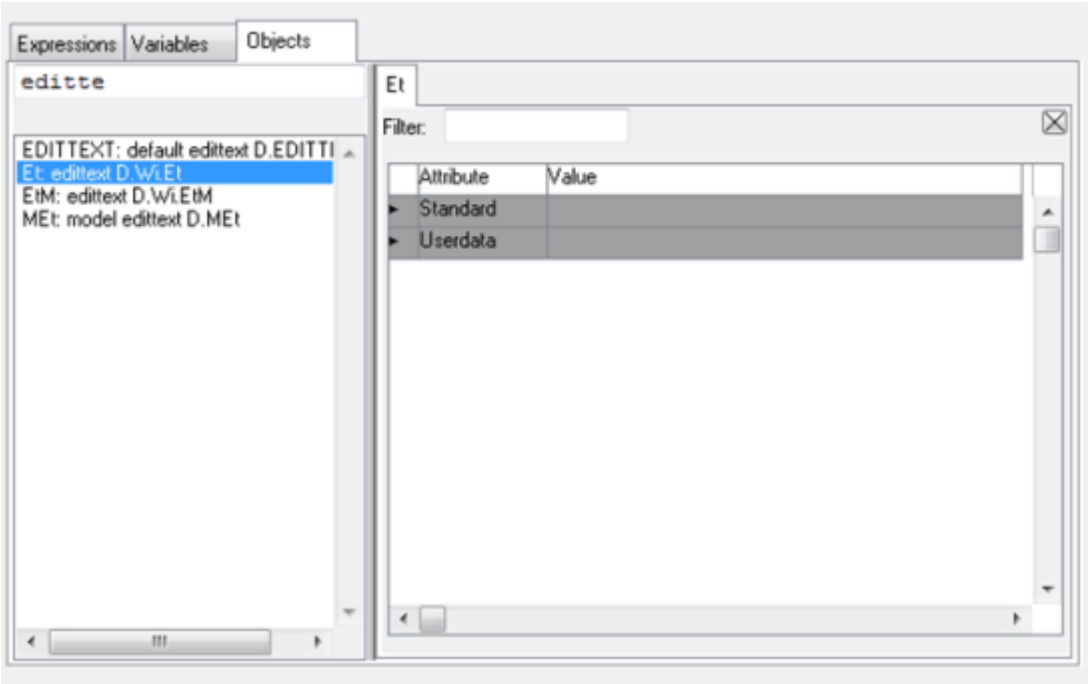


Abbildung 8: *Suche mit passenden Ergebnisobjekten*

Die Suche umfasst folgende Objektarten:

- Visuelle Objekte (***Fenster***, ***Edittext***, ***Tablefield***…)
- ***Application***, ***Record***
- ***Document***, ***Doccursor***
- Ressourcen (***Accelerator***, ***Color***, ***Font***…)
- ***Setup***-Objekt
- `this`-Objekt
- `thisevent`-Objekt
- `clipboard`-Objekt

31.6.2.2 Objekte und ihre Attribute überwachen

Ein Objekt aus der Ergebnisliste wird mit einem Doppelklick in die *„Objektanzeige“* übernommen. Nach der Übernahme in die *„Objektanzeige“* sieht man dort zunächst die zugeklappten Attributkategorien *„Standard“* mit den vordefinierten Attributen und *„Userdata“* mit den benut-

zerdefinierten Attributen (siehe „Abbildung 8“, rechter Bereich). Klappt man diese Kategorien auf, wird die Liste der jeweiligen Attribute angezeigt.

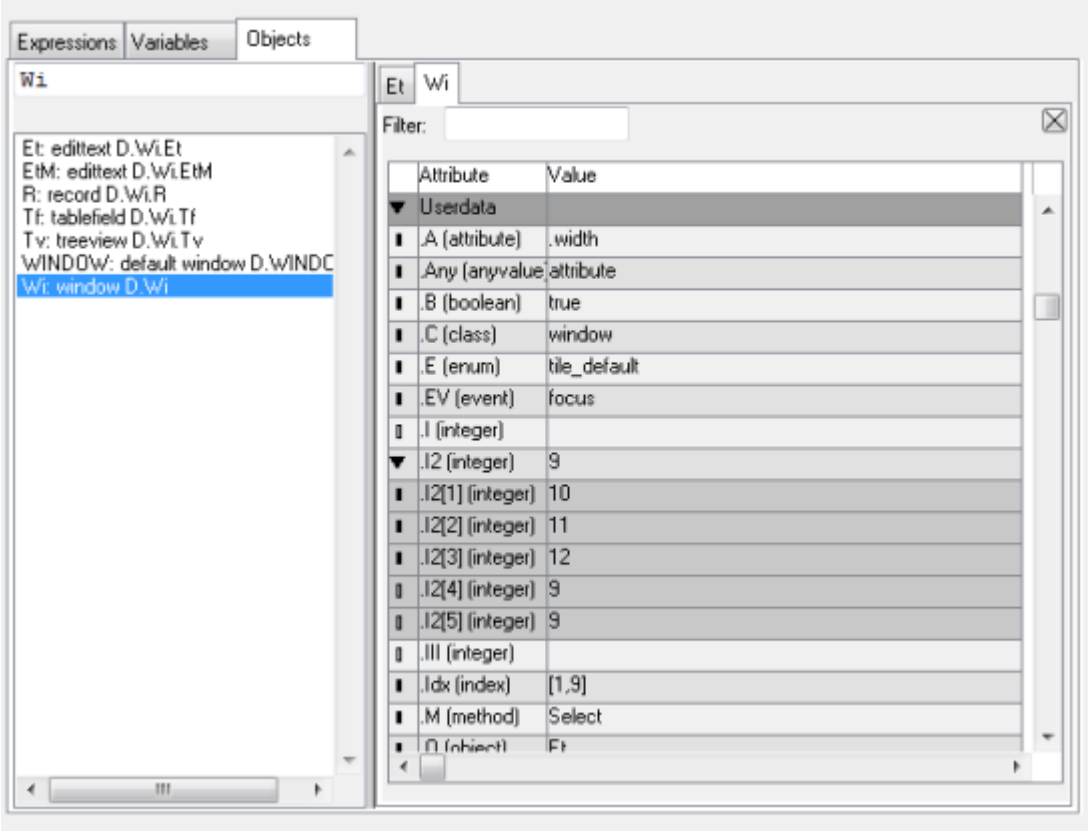


Abbildung 9: *Attribute des überwachten Objektes „Wi“*

Zu jedem Attribut werden in der zweiten Listenspalte Name und Datentyp und in der dritten Spalte der Wert angezeigt. In der ersten Spalte zeigt ein Viereck (QT, WINDOWS) bzw. Stern (MOTIF), dass der Wert des Attributs am überwachten Objekt gesetzt wurde. Bei geerbten Werten ist das Viereck nicht ausgefüllt (QT, WINDOWS) bzw. die erste Spalte leer (MOTIF).

Indizierte Attribute erkennt man an Pfeilsymbolen in der ersten Spalte. Sie zeigen nach rechts, wenn die Einzelwerte des Attributs ausgeblendet sind, und nach unten, wenn die Einzelwerte eingeblendet sind. Die Liste der Einzelwerte kann per Mausklick auf- und zugeklappt werden. Einzelwerte indizierter Attribute sind durch ein dunkleres Grau als Hintergrund und die Indexangabe beim Namen gekennzeichnet.

Wenn ein überwachtes Objekt im Verlauf des Debuggens zerstört wird, dann wird nach einer Mitteilung an den Anwender die Registerkarte des Objekts aus der *„Objektanzeige“* entfernt und die *„Objektliste“* aktualisiert.

Mit der Schaltfläche *„X“* kann ein Objekt aus der *„Objektanzeige“* entfernt werden. Wenn bereits 3 Objekte überwacht werden, muss zunächst ein Objekt aus der *„Objektanzeige“* entfernt werden, bevor man ein weiteres Objekt zur Überwachung auswählen kann.

Wird das `this`-Objekt zur Überwachung ausgewählt, dann wird das zum Zeitpunkt der Auswahl enthaltene Objekt in die *„Objektanzeige“* übernommen. Das überwachte Objekt hat anschließend keinen Bezug zum `this`-Objekt mehr.

Hinweise

- Es sollten keine Objekte mit besonders vielen Attributen oder großen indizierten Attributen überwacht werden. Bei ihnen kann es unter Umständen zu großen Zeitverzögerungen bei der Aktualisierung und Darstellung kommen.
- Ändert sich die Größe eines indizierten Attributs aus dem Dialog heraus, wird das gesamte Attribut beim nächsten Halt bzw. Schritt rot markiert.
- Der Vererbungsstatus eines Attributs lässt sich nicht immer verlässlich nachverfolgen. Deshalb kann es vorkommen, dass nicht korrekt angezeigt wird, ob ein Attributwert geerbt oder am Objekt gesetzt ist.

31.6.2.3 Attribute filtern

Wenn man nicht alle Attribute eines überwachten Objekts angezeigt bekommen möchte, kann man im Eingabefeld *„Filter“* einen Attributnamen oder einen Teil davon als Filterausdruck eingeben. Dann werden in der Tabelle nur die vordefinierten und benutzerdefinierten Attribute angezeigt, deren Name den Filterausdruck enthält. Um wieder alle Attribute anzuzeigen, löscht man den Filterausdruck und bestätigt mit `Return`.

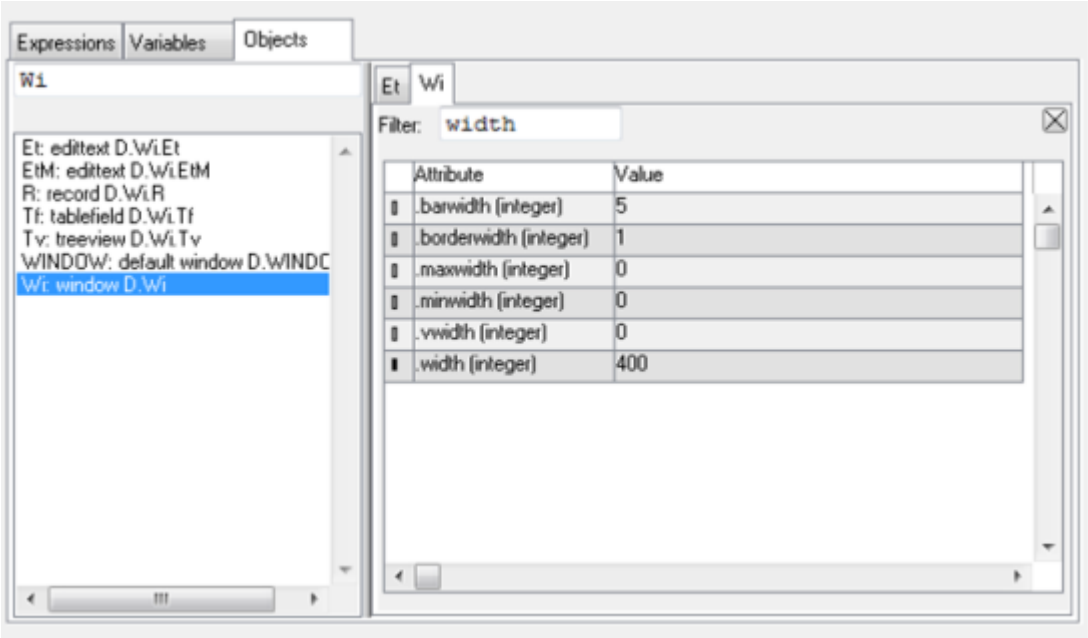


Abbildung 10: *Gefilterte Attributansicht*

31.6.2.4 Attributänderungen

Attribute, die sich im Verlauf des Debuggens ändern, werden mit ihren neuen Werten rot dargestellt (siehe „Abbildung 11“).

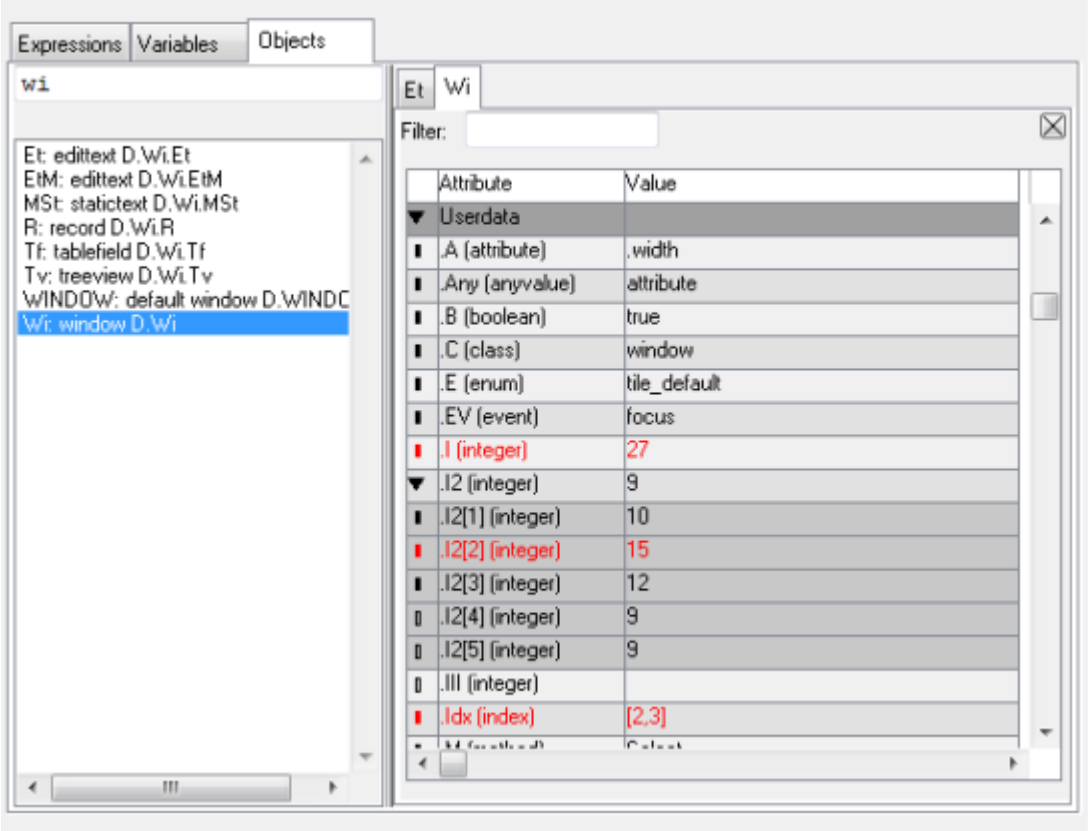


Abbildung 11: *Geänderte Attribute*

Attributwerte können manuell geändert werden, wenn die Programmausführung an einem Breakpoint angehalten wurde. Ein Attributwert lässt sich nach einem Doppelklick in das Feld mit dem Wert bearbeiten.

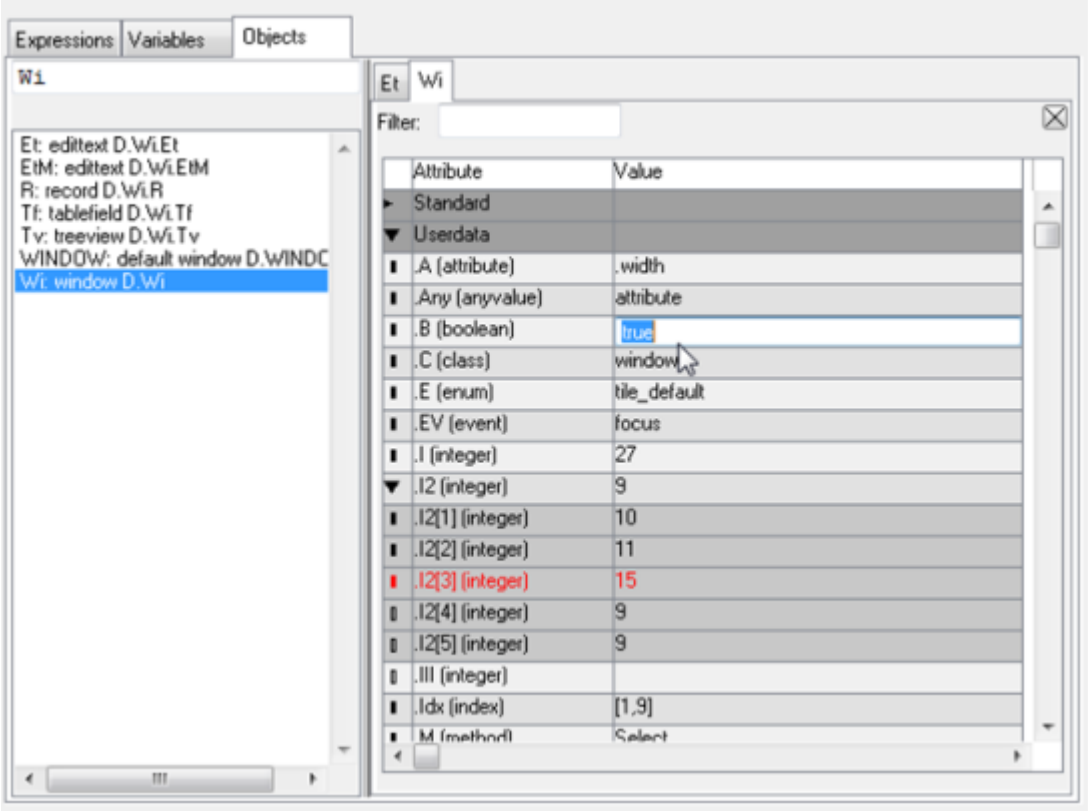


Abbildung 12: *Manuelles Ändern von Attributwerten*

Beim Ändern von Attributen mit dem Datentyp *string* muss der Wert in Anführungszeichen eingeschlossen werden (z. B. *"New Value"*).

Die Größe von indizierten Attributen und assoziativen Arrays kann manuell nicht verändert werden, das heißt es können keine Werte hinzugefügt und entfernt werden.

Hinweise

- Wird für ein Attribut dessen Datentyp sowohl Objekte als auch Strings akzeptiert (*anyvalue* oder *string*) ein Wert ohne Anführungszeichen eingegeben, der nicht als Objektbezeichner identifiziert werden kann, dann wird der Wert in einen *string* gewandelt.
- Wird bei einem *string*-Wert nur das öffnende oder schließende Anführungszeichen eingegeben, dann wird das fehlende Anführungszeichen automatisch ergänzt.
- Objekte aus noch nicht geladenen Modulen können weder von der Objektsuche gefunden noch einem Attribut als Wert zugewiesen werden.

31.7 USW

Neu (12264): Die Callback Funktion **UC_Inquire()** wurde unter MICROSOFT WINDOWS um die Abfrage *UC_I_querykey* erweitert, damit das USW Navigationstasten verarbeiten kann, die standardmäßig vom IDM interpretiert werden.

Der Datentyp *UC_InqName* wurde erweitert:

```
typedef enum {
  UC_I_undef = 0,
  ...
  UC_I_querykey  // IDM-Version A.05.02.o, ab A.06.01.b
} UC_InqName;
```

Die Struktur **UC_InqArg** wurde erweitert:

```
typedef struct {
  DM_InqName  name;
  union {
    ...
    UC_Replace   replace;
    UC_Querykey  querykey;  // IDM-Version A.05.02.o, ab A.06.01.b
  } arg;
} UC_UC_InqArg;
```

Die Struktur **UC_Querykey** wurde neu eingeführt:

```
typedef struct {
  #if defined(WIN32)
    MSG  msg;
    DM_Boolean  wantit;
  #else
    DM_Int  msg;
    DM_Boolean  wantit;
  #endif
} UC_Querykey;
```

Die Struktur **UC_Querykey** dient unter MICROSOFT WINDOWS zur Abfrage, ob eine Tastaturnachricht vom USW benötigt wird. Es bleibt dem WSI überlassen, wann und für welche Tastaturnachrichten ein Aufruf von **UC_Inquire** mit *UC_I_querykey* erfolgt.

Das Element *msg* enthält die Windows-Nachricht.

Wenn das Element *wantit* auf *DM_TRUE* gesetzt wird, dann wird die Nachricht nicht vom IDM verarbeitet. Hierdurch wird unter anderem die Tastaturnavigation ausgeschaltet.

Ein Beispiel für die Verwendung befindet sich in *demos/usw/uctext.c*. Darin werden nur die Tasten gefiltert, die der IDM standardmäßig verarbeiten würde.

Neu (11558): Das Attribut *.userdata* kann nun als Vektor- oder Matrixattribut definiert werden.

Der IDM erlaubt nun die Definition des Attributs *.userdata[]*. Dabei sind folgende Restriktionen für den Attributdeskriptor zu beachten:

1. Er sollte nur einmal vorkommen.

2. Der Datentyp muss auf *DT_anyvalue* und der Wert des Elements *indexed* muss *UCI_VECTOR* oder *UCI_MATRIX* sein. Die Elemente *rowcount* und *colcount* sollten durch ein anderes Attribut gesteuert werden.
3. Der Default-Datenwert sollte *(UC_Data) 0* sein. Da *.userdata* Attribute ausschließlich für nicht-visuelle Informationen vorbehalten sind, wird die **update**-Funktion des USW für diese Attribute **nicht** aufgerufen.

Neu: Für USWs können nun auch die Schnittstellenfunktionen **DM_SetContent()** und **DM_GetContent()** verwendet werden.

Für die vollständige Nutzbarkeit sollte das USW die Attribute

```
string   .content[]
boolean  .active[]
boolean  .sensitive[]
anyvalue .userdata[]
```

mit der gleichen Indexierung verwenden. Hierzu ist die Redefinition der Attribute *.sensitive[]*, *.active[]* und *.userdata[]* in indizierter Form (als *UCA_VECTOR* oder *UCA_MATRIX*) erlaubt.

Neu: Für Attribute eines USW kann im Attributdeskriptor nun auch der Datentyp *DT_text* verwendet werden.

Ein Aufruf von **attrGet()** der USW-Schnittstelle liefert den Text dann als String zurück.

Die Zuweisung eines Strings an ein Attribut vom Typ *DT_text*, z. B. über die API-Funktion **attrSet()**, wird vom WSI mit einem TypeClash-Fehler abgelehnt.

Dies betrifft auch die Funktionen für das Format-Handling, falls versucht wird, den neuen Content-String an ein Attribut vom Typ *DT_text* zuzuweisen. Die Format-Funktionen **fmt*()** liefern dann ein *UC_FS_FAILURE* zurück.

Änderung: In den USW-Beispielen haben nun einige Attribute den neuen Datentyp *DT_text*.

31.8 COBOL

12336: Das Modul **ufcob.obj** kann wieder fehlerfrei dynamisch mit der **dmdll.lib** bzw. **dmrtdll.lib** gelinkt werden.

11767: Die COBOL-Beispiele lassen sich wieder bauen.

Behoben: Der Aufruf von COBOL-Funktionen funktioniert nun auch auf der 64-Bit-Variante von MICROSOFT WINDOWS.

Behoben: Die Groß- und Kleinschreibung von **DMufcob_BindThruLoader** wurde an die COBOL-Erfordernisse angepasst.

31.9 Neue Stringfunktionen **strreplace()**, **concat()** und **split()**

31.9.1 strreplace()

Die Funktion **strreplace()** ersetzt Teilstrings in einem String durch einen oder mehrere Ersatzstrings. Die zu ersetzenden Teile können entweder in Form einer Position mit einer optionalen Längenangabe, einer Zeichenfolge oder einer Liste von Zeichenfolgen definiert werden.

Definition

In der ersten Form wird im Parameter *Index* die Position des Teilstrings übergeben, der innerhalb von *String* durch den Ersatzstring *Replace* ersetzt werden soll. Der optionale Parameter *Length* definiert, wie viele Zeichen ersetzt werden.

```
string strreplace
(
      string  String  input,
      integer Index   input,
      string  Replace input
  { , integer Length  input }
)
```

Bei der zweiten Form wird in *String* jedes Vorkommen von *Match* durch den Ersatzstring *Replace* ersetzt. Der optionale Parameter *IgnoreCase* bestimmt, ob beim Vergleich die Groß- und Kleinschreibung beachtet wird.

```
string strreplace
(
      string  String  input,
      string  Match   input,
      string  Replace input
  { , boolean IgnoreCase := false input }
)
```

In der dritten Form von **strreplace()** werden in *String* alle Teilstrings aus der Sammlung *MatchList* durch den einzelnen Ersatzstring in *ReplaceListOrString* bzw. den entsprechenden Ersatzstring in der Liste *ReplaceListOrString* ersetzt. Der optionale Parameter *IgnoreCase* bestimmt, ob beim Vergleich die Groß- und Kleinschreibung beachtet wird.

```
string strreplace
(
      string   String    input,
      anyvalue MatchList input,
      anyvalue ReplaceListOrString input
  { , boolean  IgnoreCase := false input }
)
```

Parameter

string *String* input

In diesem Parameter wird der String übergeben in dem Teilstrings ersetzt werden sollen.

integer *Index* input

Dieser Parameter bestimmt, an welcher Position innerhalb von *String* der Teilstring beginnt, der ersetzt werden soll. Ist *Index* größer als die Länge von *String*, wird der Ersatzstring an *String* angehängt. *Index* < *1* wird wie *Index* = *1* behandelt.

string *Match* input

In diesem Parameter wird der Teilstring übergeben, der innerhalb von *String* ersetzt werden soll. Wenn *Match* ein Leerstring ist, erfolgt keine Ersetzung.

anyvalue *MatchList* input

In diesem Parameter wird eine Liste von Teilstrings übergeben, die innerhalb von *String* ersetzt werden sollen. *MatchList* kann eine beliebige Sammlung sein, deren Werte jedoch alle den Datentyp *string* oder ***text*** haben müssen.

string *Replace* input

In diesem Parameter wird der Ersatzstring übergeben, durch den die Teilstrings in *String* ersetzt werden. Die Teilstrings werden in der Reihenfolge ihres Auftretens innerhalb von *String* ersetzt. Es erfolgt keine rekursive Ersetzung.

anyvalue *ReplaceListOrString* input

Dieser Parameter enthält einen einzelnen Ersatzstring oder eine Liste von Ersatzstrings. Die Liste kann eine beliebige Sammlung sein, deren Werte jedoch alle den Datentyp *string* oder ***text*** haben müssen und deren Länge mit der Länge von *MatchList* übereinstimmen muss.
Enthält *ReplaceListOrString* eine Sammlung, wird jedes Vorkommen des Teilstrings *MatchList[I]* durch den Ersatzstring *ReplaceListOrString[I]* ersetzt. Die Reihenfolge der Ersetzung wird durch die Reihenfolge der Teilstrings in *MatchList* bestimmt, wobei die einzelnen Teilstrings in der Reihenfolge ihres Auftretens innerhalb von *String* ersetzt werden.
Es erfolgt keine rekursive Ersetzung, das heißt jeder Teilbereich von *String* wird maximal einmal ersetzt.

*integer **Length** input*

Dieser optionale Parameter definiert die Anzahl der Zeichen, die durch den Ersatzstring ersetzt werden. *Length* kann nur in Verbindung mit *Index* verwendet werden. Wenn *Length* nicht angegeben ist, dann ersetzt *Replace* den Teilstring ab *Index* bis zum Ende von *String*. Bei *Length* <= *0* wird der Ersatzstring eingefügt ohne Zeichen zu ersetzen.

*boolean **IgnoreCase** := false input*

Dieser optionale Parameter steuert, ob beim Vergleich der Teilstrings die Groß- und Kleinschreibung berücksichtigt wird.

IgnoreCase	Bedeutung
false (Standardwert)	Beim Vergleich der Teilstrings wird die Groß- und Kleinschreibung berücksichtigt und muss übereinstimmen.
true	Die Groß- und Kleinschreibung wird beim Vergleich der Teilstrings ignoriert und muss nicht übereinstimmen.

Rückgabewert

String in dem Teilstrings durch Ersatzstrings ersetzt wurden.

Beispiele

» Mehrfaches Ersetzen des Buchstaben *"l"* durch den Buchstaben *"L"*

```
print strreplace("Hallo Welt", "l", "L");
```

Ausgabe

```
"HaLLo WeLt"
```

» Ersetzen von mehreren Teilstrings

```
print strreplace("Otto", ["Ot", "t"], ["U", "d"]);
```

Ausgabe

```
"Udo"
```

» Ersetzen von mehreren Teilstrings
Jeder Teilbereich des Strings *"abc"*wird höchstens einmal ersetzt, also *"a"* durch *"b"*, *"b"* durch *"c"* und *"c"* durch *"x"*. Es wird **nicht** rekursiv, also nicht *"a"* durch *"b"* durch *"c"* durch *"x"*, ersetzt.

```
print strreplace("abc", ["a", "b", "c"], ["b", "c", "x"]);
```

Ausgabe

```
"bcx"
```

» Ersetzen von mehreren Teilstrings durch das Zeichen *"."* ohne Berücksichtigung der Groß- und Kleinschreibung

```
print strreplace("Abrakadabra Rizinus", ["A", "r", "IZI", "u"], ".",
true);
```

Ausgabe

```
".b..k.d.b.. ..n.s"
```

» Ersetzen eines Teilbereichs durch einen anderen String

```
print strreplace("Bamjok", 3, "ngk", 2);
```

Ausgabe

```
"Bangkok"
```

» Einfügen am Anfang eines Strings

```
print strreplace("Eins", 0, "Null ", 0);
```

Ausgabe

```
"Null Eins"
```

» Anhängen am Ende eines Strings

```
print strreplace("Eins", 5, " Zwei");
```

Ausgabe

```
"Eins Zwei"
```

» Anhängen außerhalb der String-Länge

```
print strreplace("Eins", 100, "...");
```

Ausgabe

```
"Eins..."
```

31.9.2 concat()

Die Funktion **concat()** verkettet mehrere Strings zu einem String. Bei der Verkettung kann eine Trennzeichenfolge zwischen den Strings eingefügt werden oder die Strings können wiederholt aneinander gehängt werden.

Definition

Wenn der erste Parameter ein String ist, wird dieser String immer zwischen zwei zu verkettenden Strings eingefügt.

```
string concat
(
      string   Separator input,
      anyvalue Value1    input
  { , anyvalue Value2    input
  ...
  { , anyvalue Value15   input } }
)
```

Ist der erste Parameter ein *integer*-Wert, dann definiert er, wie oft die Verkettung der Strings wiederholt wird.

```
string concat
(
      integer  Repeat  input,
      anyvalue Value1  input
  { , anyvalue Value2  input
  ...
  { , anyvalue Value15 input } }
)
```

Parameter

string *Separator* input

Dieser Parameter enthält eine Trennzeichenfolge, die bei der Verkettung zwischen zwei Strings eingefügt wird.
Separator kann nicht zusammen mit *Repeat* verwendet werden.

integer *Repeat* input

Dieser Parameter definiert, wie oft die Verkettung der Strings wiederholt werden soll. Zunächst wird ein String aus den übergebenen *Value*-Parametern gebildet, anschließend wird der gebildete String so oft aneinander gehängt, wie in *Repeat* angegeben. Ist *Repeat* <= *0*, wird ein Leerstring zurückgegeben.
Repeat kann nicht zusammen mit *Separator* verwendet werden.

anyvalue *Value1* input

anyvalue ***Value2*** *input*

…

anyvalue ***Value15*** *input*

In diesen (optionalen) Parametern werden die Werte übergeben, die zu einem String verkettet werden sollen.
Skalare Werte werden für die Verkettung zu Strings gewandelt. Enthalten die Parameter Sammlungen, dann werden zunächst die darin enthaltenen Werte (ohne Defaultwerte) verkettet und anschließend die Parameter miteinander verkettet. Die Vorgehensweise entspricht einem Aufruf von *sprintf("%s", <Value>)*.

Rückgabewert

String in dem die übergebenen Werte verkettet sind.

Beispiele

» Verketten von drei Werten mit Trennzeichen

```
text TxTres "Tres";
...
```

```
print concat(",", 1, "zwei", TxTres);
```

Ausgabe

```
"1,zwei,Tres"
```

» Wiederholte Verkettung

```
print concat(3, "|", [".", ".."]);
```

Ausgabe

```
"|...|...|..."
```

31.9.3 split()

Mit der Funktion **split()** kann ein String an Trennzeichen in eine Liste von Teilstrings aufgeteilt werden. Sind keine Trennzeichen angegeben, dann wird der String zeichenweise zerlegt.

Definition

```
list split
(
  string Separators input,
  string String     input
)
```

Parameter

string *Separators* input

In diesem Parameter werden die Trennzeichen angegeben, an denen der Parameter *String* aufgeteilt wird. Die Trennzeichen sind in der Ergebnisliste nicht enthalten.
Enthält *Separators* eine Zeichenfolge, wird jedes einzelne Zeichen daraus als Trennzeichen verwendet. Folgen in *String* zwei Trennzeichen direkt aufeinander, wird an dieser Stelle ein Leerstring in die Ergebnisliste aufgenommen.
Ist *Separators* ein Leerstring, dann wird der Parameter *String* in eine Liste einzelner Zeichen zerlegt.

string *String* input

In diesem Parameter wird der String übergeben, der zerlegt werden soll.

Rückgabewert

Eine Liste von Teilstrings.

Beispiele

» Zerlegung in einzelne Zeichen

```
print split("", "Udo");
```

Ausgabe

```
["U","d","o"]
```

» Zerlegung an verschiedenen Trennzeichen

```
print split(",.;", "1,23,4.5;;6,478-9");
```

Ausgabe

```
["1","23","4","5","","6","478-9"]
```

31.10 Verwendung von XML mit dem Datenmodell

XML lässt sich als Datamodel nutzen, wobei Knotentexte und Knotenattribute die Daten – in der Regel Strings – enthalten können. Ein XML-Dokument wird mit einem ***Doccursor*** an eine View gekoppelt. Dafür werden am ***Doccursor*** Datamodel-Attribute und Selektionsmuster für Knoten des XML-Dokuments definiert. Weiterhin lässt sich festlegen, ob das Datamodel-Attribut an den Inhalt oder einen Attributwert des Knotens gekoppelt ist. Bei Operationen, die das XML-Dokument ändern, werden die Datenänderungen an alle Datamodel-Attribute weitergereicht.

Der ***Doccursor*** besitzt folgende Attribute um ihn als Datamodel zu verwenden:

Tabelle 4: *Datenmodellattribute des Doccursors*

Attribut	Bedeutung
.dataselect	Definiert ein Datamodel-Attribut mit zugehörigem Selektionsmuster.
.dataselectattr	Ordnet ein Datamodel-Attribut einem Attribut der selektierten Knoten zu.
.dataselecttype	Datentyp-Konvertierung eines Datamodel-Attributs.
.dataselectcount	Definiert die Kardinalität eines Datamodel-Attributs.
.dataoptions	Steuert das Caching über den Index *dopt_cache_data*.

Dabei ist das Attribut *.dataselect* von zentraler Bedeutung. Es definiert die Datamodel-Attribute des ***Doccursors*** sowie deren Verknüpfung mit Knoten des XML-Dokuments mithilfe von Selektionsmustern. Die Syntax der Selektionsmuster entspricht den Musterdefinitionen der **:select**-Methode. Das Selektionsmuster eines *.dataselect*-Attributs ohne Index – also ohne Datamodel-Attribut – wird **vor** den Selektionsmustern aller Datamodel-Attribute angewendet. Dies ermöglicht relative Selektionsmuster für die Datamodel-Attribute, ausgehend von den Knoten, die vom *.dataselect*-Attri-

but ohne Index vorselektiert worden sind. Gleichzeitig lässt sich dadurch der Aufwand für das Sammeln der Daten reduzieren.

Beim Sammeln der Daten für ein Datamodel-Attribut wird das Dokument anhand des Selektionsmusters in einer Schleife durchlaufen. Dabei wird entweder der Knoteninhalt über das *.text*-Attribut des ***Doccursors*** oder der Wert des über *.dataselectattr* gesetzten Knotenattributs geholt.

Für die mit *.dataselect* definierten Datamodel-Attribute können mit den Attributen *.dataselecttype* und *.dataselectcount* der Datentyp und die Kardinalität geändert werden. Standardmäßig enthalten die Datamodel-Attribute Vektoren mit String-Werten (Datentyp *vector[string]*). Mit dem Attribut *.dataselecttype* kann ein Datentyp (z. B. *integer*, *boolean*) definiert werden, in den die Werte konvertiert werden. Die Kardinalität (Vektor oder Skalar) von Datamodel-Attributen kann über das Attribut *.dataselectcount* gesteuert werden. Wenn der Datentyp des Datamodel-Attributs eine Sammlung oder seine Kardinalität der Datentyp *integer* ist, dann enthält das Datamodel-Attribut einen *vector*, ansonsten nur einen Skalar mit dem ersten Wert.

Einmal geholte Werte eines Datamodel-Attributs werden zwischengespeichert bis sich entweder das XML-Dokument oder die Datenmodellattribute des ***Doccursors*** ändern. Dieses Zwischenspeichern („Caching“) kann durch Setzen von *.dataoptions[dopt_cache_data] = false* am ***Doccursor*** unterbunden werden.

Das Speichern von Daten in einem XML-Dokument erfolgt analog mit sinngemäß umgedrehten Operationen. Allerdings können im XML-Dokument keine Knoten automatisch angelegt oder gelöscht werden.

Hinweise

- Das Sammeln der Daten kann eine „teure“ Operation sein, da im gesamten XML-Dokument nach dem Selektionsmuster gesucht werden muss. Der Aufwand lässt sich möglicherweise reduzieren, indem die Suche mithilfe eines *.dataselect*-Attributs ohne Index auf vorselektierte Teilbäume beschränkt wird.
- Ein ***Doccursor***, der als Datenmodell verwendet wird, sollte nicht für andere Zwecke genutzt werden, da er typischerweise ständig seinen Selektionspfad ändert.

31.10.1 Beispiel

Eine Liste von Nobelpreisträgern soll aus folgender XML-Datei ausgelesen und in einer Tabelle dargestellt werden:

```
<?xml version="1.0"?>
<nobelprizes>
  <category id="p">Physics</category>
  <category id="c">Chemistry</category>
  <winner year="1" category="p">Wilhelm Conrad Röntgen</winner>
  <winner year="11" category="c">Marie Curie</winner>
  <winner year="18" category="p">Max Planck</winner>
  <winner year="70" category="c">Luis Leloir</winner>
```

```
</nobelprizes>
```

Die Namen der Preisträger werden aus dem Inhalt der XML-Knoten „winner“ geholt und im Datamodel-Attribut „.Winner“ abgelegt. Die Jahreszahlen der Auszeichnung stehen im Attribut „year“ der XML-Knoten und werden in der Datei ab 1900 gezählt. Sie werden zu *integer* gewandelt und als *vector* in das Datenmodell-Attribut „.Year“ eingelesen. In der überschriebenen **:represent**-Methode werden die Jahreszahlen für die Anzeige zu vierstelligen Zahlen ergänzt.

```
dialog D

document Doc
{
  doccursor DocCur
  {
    .dataselect[.Winner]    "..winner";
    .dataselect[.Year]      "..winner";
    .dataselectattr[.Year]  "year";
    .dataselecttype[.Year]  integer;
    .dataselectcount[.Year] integer;
  }
}

window Wi
{
  .title "Nobel prize winners";
  .width 300;  .height 220;

  tablefield Tf
  {
    .xauto 0;  .yauto 0;
    .datamodel DocCur;
    .colcount 2;  .rowcount 1;
    .rowheader 1;  .colheader 1;
    .rowheight[0] 25;
    .colwidth[0] 180;  .colwidth[1] 60;
    .content[1,1] "Year";
    .content[1,2] "Winner";
    .dataget[.field]     .Winner;
    .dataget[.userdata]  .Year;
    .dataindex[.userdata] [0,1];

    :represent()
    {
      variable integer I;
      if Attribute=.userdata then
        for I := 1 to itemcount(Value) do
          Value[I] := 1900 + (Value[I] % 100);
```

```
        endfor
        setvector(this, .content, Value,[2,1],
                  [1 + itemcount(Value),1]);
        return;
      endif
      pass this:super();
    }
  }

  on close { exit(); }
}

on start
{
  Doc:load("nobelprizes.xml");
}
```

Dieser Dialog erzeugt folgendes Fenster:



Abbildung 13: *Tabelle mit XML-Daten als Datenmodel*

31.10.2 Neuer Indexwert *dopt_cache_data* des Attributs *dataoptions*

Attribut-Index	Default	Komponente	Bedeutung
dopt_cache_data	***true***	Model	Dieser Indexwert ist nur für den ***doccursor*** verfügbar. **true** Die vom ***doccursor*** selektierten Daten werden für weitere Zugriffe zwischengespeichert („Caching“). **false** Der ***doccursor*** selektiert die Daten bei jedem Zugriff neu aus dem XML-Dokument.

31.10.3 Neue Attribute des Doccursors

31.10.3.1 .dataselect

Mit diesem Attribut werden gleichzeitig das als Index angegebene Datamodel-Attribut und ein als Wert zugewiesenes Selektionsmuster für Knoten eines XML-Dokuments definiert.

	Bezeichner		Datentyp
Regelsprache	**.dataselect[attribute]**		string
C	AT_dataselect		DT_string
COBOL	AT-dataselect		DT-string
Klassifizierung	Datenmodellattribut		
Objekte	***doccursor***		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Das definierte Datamodel-Attribut kann dann zur Kopplung mit einer View verwendet werden. Datenänderungen werden über das im Index angegebene Attribut des ***Doccursors*** weitergegeben. Das Selektionsmuster beschreibt die Knoten des XML-Dokuments aus denen die Daten geholt bzw. in denen sie gespeichert werden.

Die Syntax des Selektionsmusters entspricht der Musterdefinition für die **:select**-Methode des ***Doc-cursors***.

Das Selektionsmuster des *.dataselect*-Attributs ohne Index definiert Knoten als Ausgangspunkte für die indizierten *.dataselect*-Attribute. Die Selektionsmuster der indizierten *.dataselect*-Attribute werden dann als relative IDM-Pfade interpretiert. Sie referenzieren Knoten innerhalb der Teilbäume, deren Wurzeln durch das nicht indizierte *.dataselect*-Attribut selektiert werden. Die indizierten *.dataselect*-Attribute greifen in diesem Fall also nicht auf das gesamte XML-Dokument zu, sondern nur auf die Unterknoten jener Knoten, die vom nicht indizierten *.dataselect*-Attribut selektiert worden sind.

Um eine konsistente Behandlung von optionalen XML-Elementen und -Attributen zu ermöglichen, gibt die Selektion eines Datamodel-Attributs einen Leerstring zurück, wenn ein Wert nicht vorhanden ist oder sich nicht innerhalb der vorselektierten Teilbäume befindet.

Mit dem Attribut *.dataselectattr* kann festgelegt werden, ob das Datamodel-Attribut mit dem Knoteninhalt oder einem Knotenattribut verknüpft ist. Datentyp und Kardinalität des Datamodel-Attributs können mit den Attributen *.dataselecttype* und *.dataselectcount* gesteuert werden.

Beispiel

In diesem Beispiel stammen die Daten für eine Liste mit Nobelpreisträgern aus einem XML-Dokument, in dem die Informationen zu jedem Preisträger in einem Knoten „prize“ stehen.

```
<?xml version="1.0"?>
<nobelprizes>
  <category id="p">Physics</category>
  <category id="c">Chemistry</category>
  <prize year="1" category="p">
    <merit>discovery of x-rays</merit>
    <winner>Wilhelm Conrad Röntgen</winner>
  </prize>
  <prize year="11" category="c">
    <merit>discovery of radioactivity</merit>
    <winner>Marie Curie</winner>
  </prize>
  <prize year="18" category="p">
    <merit>development of the concept of quanta</merit>
    <winner>Max Planck</winner>
  </prize>
  <prize year="70" category="c">
    <merit>discovery of sugar nucleotides and their role
      in the biosynthesis of carbohydrates</merit>
    <winner>Luis Leloir</winner>
  </prize>
</nobelprizes>
```

Die XML-Knoten können dann beispielsweise wie folgt über Datamodel-Attribute verknüpft werden:

```
...
document Doc
{
  doccursor DocCur
  {
    .dataselect "..prize";
    .dataselect[.Winner]    ".winner";
    .dataselect[.Year]      ".";
    .dataselect[.Discovery] "merit";
    .dataselectattr[.Year]  "year";
    .dataselecttype[.Year]  integer;
    .dataselectcount[.Year] integer;
  }
}
...
```

Siehe auch

Attribute .dataselectattr, .dataselectcount und .dataselecttype

31.10.3.2 .dataselectattr

Dieses Attribut definiert, mit welchem Knotenattribut das als Index angegebene Datamodel-Attribut verknüpft ist.

	Bezeichner		Datentyp
Regelsprache	**.dataselectattr[attribute]**		string
C	AT_dataselectattr		DT_string
COBOL	AT-dataselectattr		DT-string
Klassifizierung	Datenmodellattribut		
Objekte	***doccursor***		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Wenn *.dataselectattr* nicht gesetzt ist, dann ist das Datamodel-Attribut an den Knoteninhalt gekoppelt. Die Daten werden über das Attribut *.text* des ***Doccursors*** geholt und gesetzt.

Wenn *.dataselectattr* auf den Namen eines Knotenattributs gesetzt wird, dann ist das Datamodel-Attribut an die Attributwerte dieses Knotenattributs gekoppelt. Die Daten werden über das Attribut *.attribute* des ***Doccursors***, mit dem Namen des Knotenattributs als Index, geholt und gesetzt.

Siehe auch

Attribut .dataselect

31.10.3.3 .dataselecttype

Dieses Attribut definiert den Datentyp, in den die Werte des als Index angegebenen Datamodel-Attributs konvertiert werden.

	Bezeichner		Datentyp
Regelsprache	**.dataselecttype[attribute]**		datatype
C	AT_dataselecttype		DT_type
COBOL	AT-dataselecttype		DT-type
Klassifizierung	Datenmodellattribut		
Objekte	***doccursor***		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Wenn *.dataselecttype* nicht angegeben ist, dann enthält das Datamodel-Attribut einen Vektor mit Strings (Datentyp *vector[string]*).

Schlägt die Konvertierung fehl, wird das Holen der Werte mit einem Fehler abgebrochen.

Beispiel

Im folgenden Dialogausschnitt werden die drei Datamodel-Attribute „.Name“, „.Female“ und „.Name3“ definiert.

```
dialog D
...
document Doc
{
  doccursor DocCur
  {
    .dataselect[.Name] "..person";

    .dataselect[.Name3]     "..person[.birthyear=\"1978\"][3]";
    .dataselecttype[.Name3] string;

    .dataselect[.Female]      "..person";
    .dataselectattr[.Female]  "female";
    .dataselecttype[.Female]  boolean;
```

```
    .dataselectcount[.Female] integer;
  }
}
...
```

Das Attribut „.Name“ sammelt die Texte von allen „person“-Knoten in einem String-Vektor, während das Attribut „.Name3“ nur den Namen der 3. Person mit Geburtsjahr *1978* als Skalar vom Typ *string* enthält.

Beim Attribut „.Female“ werden über den Zugriff *DocCur.attribute["female"]* die entsprechenden Knotenattribute aller „person“-Knoten geholt, in *boolean* konvertiert und in einem *vector* gespeichert.

Siehe auch

Attribut .dataselectcount

31.10.3.4 .dataselectcount

Dieses Attribut definiert die Kardinalität des als Index angegebenen Datamodel-Attributs.

	Bezeichner		Datentyp
Regelsprache	**.dataselectcount[attribute]**		datatype
C	AT_dataselectcount		DT_type
COBOL	AT-dataselectcount		DT-type
Klassifizierung	Datenmodellattribut		
Objekte	***doccursor***		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Wertebereich

integer (Standard)	Das Datamodel-Attribut ist ein *vector* und enthält die Inhalte bzw. Attributwerte aller Knoten, die dem Selektionsmuster des Attributs *.dataselect* entsprechen.
void	Das Datamodel-Attribut ist ein Skalar und enthält nur den Inhalt bzw. Attributwert des ersten Knotens, der dem Selektionsmuster des Attributs *.dataselect* entspricht.

Siehe auch

Attribut .dataselecttype

31.11 Laufzeitbegrenzung von Testlizenzen

Die Laufzeit von Testlizenzen ist nun durch technische Maßnahmen begrenzt. Im Kalendermonat, in dem die Laufzeit endet, wird eine Meldung ausgegeben. Außerdem geben Testlizenzen im Tracefile eine zusätzliche Zeile mit ihrem Ablaufdatum und einem Kürzel für den Lizenzinhaber aus.

Das Ablaufdatum wird auch bei der Installation angezeigt, wenn mit einem Lizenzschlüssel für eine Testlizenz installiert wird.

Nach dem Ablaufdatum wird eine Meldung ausgegeben und die Anwendung beendet. Danach ist auch keine Installation des ISA DIALOG MANAGERS mit dem Lizenzschlüssel der Testlizenz mehr möglich.

Falls notwendig, können Sie bei uns einen neuen Lizenzschlüssel beantragen, um eine Testlizenz zu verlängern.

32 A.06.01.a2

32.1 Kern

12282: Bei der Verarbeitung von Strings im Regel-Interpreter kommt es nicht mehr zu zufälligen Codepage-Konvertierungen von bzw. zur Applikationscodepage.

33 A.06.01.a

33.1 Qt-Unterstützung

33.1.1 Unterstützte Plattformen

Der IDM FÜR QT wird als 32- und 64-Bit-Version für Qt 4.6 und Red Hat Enterprise Linux (RHEL) 5.5 sowie Qt 4.8 mit RHEL 7 angeboten. Diese Versionen sind auch auf SUSE Enterprise Linux lauffähig.

Qt selbst ist im IDM FÜR QT nicht enthalten. Die benötigten Qt-Versionen können über die Paketmanager der Systeme installiert werden. Falls mehrere Qt-Versionen auf einem System installiert sind, ist beim Bauen von Anwendungen auf das Linken der korrekten Version zu achten.

33.1.2 Fehler- und Warnmeldungen

Bei einer unpassenden Qt-Version gibt der IDM die Fehlermeldung *„WSIQT version mismatch“* aus.

Der Zugriff auf noch nicht implementierte oder nicht unterstützte Objekte, Attribute und Funktionen wird durch Meldungen wie *„…NOT-IMPLEMENTED…“* oder *„…UNSUPPORTED…“* in der Log-Datei angezeigt.

33.1.3 Besonderheiten von Qt

Qt unterstützt transparente Farben und setzt sie unter Umständen in der Standardpalette ein. Das kann dazu führen, dass sich sichtbare Objekte nicht bedienen lassen, weil ein darüber liegendes Objekt mit transparentem Hintergrund ihre Bedienung verhindert. Dieser Fall tritt zum Beispiel bei zwei sich überlappenden Statictexten mit fester Größe ein.

Da Qt verschiedene UI-Stile unterstützt, kann es dazu kommen, dass in den verschiedenen Stilen Farben (Vordergrund, Hintergrund) unterschiedlich umgesetzt oder ganz ignoriert werden. Das Erscheinungsbild kann je nach UI-Stil variieren. Neben Farben betrifft dies auch andere Darstellungsaspekte, zum Beispiel je nach UI-Stil vorhandene oder fehlende Rahmen.

Auch die Darstellung von Farbverläufen hängt sehr stark vom Objekt und vom UI-Stil ab.

Beispiel

Beim ***Poptext*** mit dem Stil *poptext* oder *edittext* und gesetztem *.bgc* wird im UI-Stil „cleanlooks“ die Hintergrundfarbe im aufgeklappten Zustand bei der Popup-Liste ignoriert.

33.1.4 Unterschiede zu anderen IDM-Versionen (Kompatibilität)

Dieses Kapitel beschreibt Unterschiede zu den Versionen des ISA DIALOG MANAGERS für andere Toolkits. Diese Unterschiede haben ihre Ursache in spezifischen Merkmalen des Qt-Toolkits, was auch

bedeuten kann, dass der IDM Möglichkeiten von Qt nutzt, die von den anderen Toolkits nicht geboten werden.

33.1.4.1 Objekte

33.1.4.1.1 Canvas

Das ***Canvas***-Objekt besitzt unter Qt die Option *opt_graphicsview*. Sie steuert, ob das Canvas mit einem ***QFrame*** (*opt_graphicsview = false*) oder einer ***QGraphicsView*** (*opt_graphicsview = true*) umgesetzt wird. Standardmäßig ist die Option ausgeschaltet. Dann kann im *Expose*-Ereignis direkt in das QWidget gezeichnet werden. Bei eingeschalteter Option wird keine *Expose*-Nachricht an die Canvas-Funktion weitergeleitet. Dieser Modus, in dem die ***QGraphicsView*** der Container einer ***QGraphicsScene*** ist, dient zum Beispiel dazu, der Szene Objekte hinzuzufügen, die von der Szene dann selbständig gezeichnet werden.

Je nach Optionswert ergeben sich Unterschiede der **DM_CanvasUserArgs**, die nun wie folgt definiert ist:

```
typedef struct {
  DM_ID  object;
  int    reason;
  #  ifdef QCOREEVENT_H
       QEvent*  event;
  #  else
       DM_Pointer  event;
  #  endif
  #  ifdef QWIDGET_H
       QWidget*  widget;  // je nach opt_graphicsview
                          // QFrame* oder QGraphicsView*
       QWidget*  window;
  #  else
       DM_Pointer  widget;
       DM_Pointer  window;
  #  endif
  #  ifdef QFONT_H
       QFont*  font;
  #  else
       DM_Pointer  font;
  #  endif
  #  ifdef QCOLOR_H
       QColor*  fgc;
       QColor*  bgc;
  #  else
       DM_Pointer  fgc;
       DM_Pointer  bgc;
  #  endif
  DM_Int2     x;
```

```
  DM_Int2     y;
  DM_UInt2    width;
  DM_UInt2    height;
  DM_Pointer  userdata;
} DM_CanvasUserArgs;
```

Zusätzlich werden *.fgc*, *.bgc* und *.font* an die Canvas-Funktion übergeben, da sie beim Zeichnen oder Hinzufügen neuer Objekte zum Teil explizit gesetzt werden müssen.

Das ***Canvas***-Objekt hat eine neue Option *opt_addevents*. Die Option ist standardmäßig ausgeschaltet. Angeschaltet bewirkt sie, dass neben den normalen Aufrufen der Canvas-Funktion auch alle *QEvents* einen Aufruf der Canvas-Funktion erwirken. So kann beispielsweise auf ein MouseMove reagiert werden. Die *reason*, die in diesem Fall der Canvas-Funktion mitgegeben wird ist *CCR_event*.

33.1.4.1.2 Edittext

Es gibt Verhaltensunterschiede zwischen einzeiligem und mehrzeiligem ***Edittext*** hinsichtlich der Darstellung einer Markierung bei Fokusverlust. Im einzeiligen Modus wird die Selektion entfernt und ebenso wie die Cusorposition nicht mehr dargestellt. Bei Wiedererlangen des Fokus stellt der IDM die Selektion wieder her. Im mehrzeiligen Modus bleibt die Selektion bei Fokusverlust erhalten.

Je nach UI-Stil kann die Darstellung des Fokusrahmens beim einzeiligen Edittext eingeschränkt sein.

Der ***Edittext*** verfügt unter Qt über die Option *.options[opt_html]*, mit der die HTML-Fähigkeit gesteuert wird.

33.1.4.1.3 Filereq

Das Attribut *.text[enum]* wird vom IDM FÜR QT nicht unterstützt, analog zum IDM FÜR WINDOWS.

Beim Anzeigen über die Funktionen **querybox** oder **DM_Querybox** wird der Datei- bzw. Verzeichnisdialog im Desktop oder Fenster – falls eins als Parameter angegeben wird – zentriert angezeigt.

Die Option *.options[opt_createprompt]* wird nicht unterstützt. Bei der Kombination *.mustexist = true* mit *.options[fro_overwriteprompt] = true* wird kein Dialog zum Bestätigen des Überschreibens angezeigt.

Eine Mehrfachselektion ist nur für existierende Dateien möglich. Bei *.multisel = true* wird *.mustexist* ignoriert.

Die Verwendung von *.bgc*, *.fgc* und *.font* ist zwar möglich, es ist aber davon abzuraten.

Zum Filtern der angezeigten Dateien kann eine Dateitypenliste angegeben werden. Die unterschiedlichen Dateitypen werden mit Tabulatoren „\t“ getrennt. Zu jedem Dateityp können mehrere Muster „*.<Endung>“ angegeben werden. Die Muster werden mit Leerzeichen oder Semikolon getrennt, die Liste der Muster ist in runde Klammern einzuschließen. Das Windows-Format der

Dateitypenliste, mit einem Tabulator zwischen der Bezeichnung des Dateityps und den zugehörigen Mustern, ohne Klammern um die Musterliste, wird ebenfalls unterstützt. Beide Varianten führen zur Anzeige der Musterliste in Klammern nach der Bezeichnung des Dateityps.

Beispiel

```
.pattern := "IDM Files (*.dlg *.idm *.mod)\tIDM Interfaces (*.if)\tAll
(*.*)";
.pattern := "IDM Files\t*.dlg;*.idm;*mod\tIDM Interfaces\t*.if\tAll\t*.*;
```

Beide Zuweisungen erzeugen folgende Auswahlliste:

```
IDM Files (*.dlg; *.idm; *.mod)
IDM Interfaces (*.if)
All (*.*)
```

33.1.4.1.4 Image

Die Farbe *.bgc* wird angewendet wenn bei intern definierten Tiles *.imagebgc* nicht angegeben ist und um bei externen Tiles transparente Bereiche einzufärben.

Bei *.borderwidth = 0* erhält das Image den Stil „flat“, das heißt es wird kein Rahmen gezeichnet. Beim Drücken der Maustaste wird jedoch ein Rahmen gezeichnet, um den Gedrückt-Zustand darzustellen.

33.1.4.1.5 Menubox, Menuitem, Menusep

Die Farben der Menüzeile eines Fensters werden durch die Attribute *.menubgc* und *.menufgc* bestimmt. Die Farben der Menüeinträge und Untermenüs werden sowohl bei Popup-Menüs als auch bei Menüzeilen durch *.bgc* und *.fgc* der ***Menubox*** bestimmt. Das bedeutet, dass Hintergrund- und Vordergrundfarbe einzelner Menüeinträge nicht verändert werden können.

Die Tastaturnavigation verhält sich je nach UI-Stil unterschiedlich. Generell kann der aktuelle Eintrag immer mit `Enter` selektiert werden. Im Hinblick auf die Selektion mit `Space` gibt es jedoch Unterschiede:

Tabelle 5: *Verhaltensunterschiede von Menüs bei unterschiedlichen UI-Stilen*

UI-Stil	Verhalten
cleanlooks, plastique, windows	Es wird auf den ersten Menüeintrag gesprungen. Es erfolgt keine Selektion, das Menü bleibt offen.
motif, cde	Für den aktuellen Menüeintrag wird ein *select*-Ereignis ausgelöst. Das Menü wird geschlossen.
gtk+	Nichts passiert, das Menü bleibt offen.

33.1.4.1.6 Messagebox

Das Attribut *.sysmodal* wird vom IDM FÜR QT nicht unterstützt.

Für das Attribut *.icon* stehen die in „Tabelle 6“ genannten Anzeigesymbole zur Verfügung, deren Aussehen durch den gewählten UI-Stil bestimmt wird.

Tabelle 6: *Icon-Werte und Anzeigesymbole*

Wert von .icon	Anzeigesymbol
icon_asterisk	**i**
icon_error	**x**
icon_exclamation	**!**
icon_hand	wie *icon_error*
icon_information	wie *icon_asterisk*
icon_query	**?**
icon_question	wie *icon_query*
icon_warning	wie *icon_exclamation*
noicon	keins

33.1.4.1.7 Poptext

Unter Qt werden standardmäßig in einem aufgeklappten ***Poptext*** maximal 10 Einträge angezeigt. Enthält die Poptext-Liste mehr als 10 Einträge, dann wird zusätzlich eine Scrollbar in der Liste angezeigt.

Das Attribut *.showitem* wird bei den UI-Stilen „mac“ und „gtk+“ ignoriert. Im UI-Stil „cleanlooks“ wird *.showitem* bei *.style = poptext* ignoriert, hier wird immer die komplette Liste angezeigt.

Bei *.style = listbox* wird *.showitem* gänzlich ignoriert.

In den Stilen *poptext* und *edittext* wird beim Navigieren durch die geöffnete Liste mit den Pfeiltasten der annavigierte Eintrag nicht automatisch selektiert. Eine endgültige Selektion muss mit `Enter` abgeschlossen werden. Dies ist Qt-Standardverhalten.

33.1.4.1.8 Progressbar

Die Option *.option[continuous]* wird nicht unterstützt. Qt zeichnet den Fortschrittsbalken abhängig vom verwendeten UI-Stil („cleanlooks“, „plastique“ usw.). Standardmäßig wird ein durchgehender Balken gezeichnet, bei dem eine Unterteilung durch Farbwechsel angedeutet wird. Sobald man *.fgc* setzt, wird der Balken jedoch durchgehend in dieser Farbe gezeichnet.

Das Attribut *.borderwidth* wird nicht unterstützt.

Das Attribut *.textfgc* wird angewendet. Sobald der Balken unter die Beschriftung läuft wird die Schrift aber in der Hintergrundfarbe *.bgc* dargestellt.

Die ***Progressbar*** zeigt bei *.value = .minvalue* links ein kleines Stück des Balkens und den Text „0%“ an. Sie wird nur dann vollkommen leer dargestellt, wenn *.value = -1* ist. Dann wird jedoch auch kein Wert in Prozent dargestellt.

33.1.4.1.9 Scrollbar

In bestimmten UI-Stilen (z. B. „cleanlooks“) wird bei *.arrows = false* ein Teil des Hintergrunds (links bzw. oberhalb des Sliders) eingefärbt.

33.1.4.1.10 Setup

Beim IDM FÜR QT hat das Attribut *.toolkit* den Enum-Wert *toolkit_qt* und *.toolkit_string* den Wert „Qt“.

Das ***Setup***-Objekt hat die neuen Attribute *.colorname[integer]*, *.cursorname[integer]* und *.fontname [integer]*, die Listen der verfügbaren Farbnamen, Cursornamen und Schriftarten enthalten (siehe Kapitel „colorname[integer]“ bis „fontname[integer]“).

Das Attribut *.toolhelp* zur Beeinflussung der Anzeigedauer wird nicht unterstützt. Außerdem kann die Darstellung der Toolhelp nicht über *.options[opt_balloon_toolhelp]* umgestellt werden.

33.1.4.1.11 Spinbox

Das Attribut *.direction* und die horizontale Ausrichtung der Pfeile wird nicht unterstützt.

33.1.4.1.12 Statusbar

Qt ignoriert die Breite von Kindern der ***Statusbar***. Abgesehen vom letzten sichtbaren Kind sind die Kindobjekte immer so breit wie für die vollständige Darstellung ihres Inhalts nötig. Deshalb hat das Attribut *.alignment* keine sichtbare Auswirkung, außer beim letzten sichtbaren Kind.

Tabulatoren um den Text von *.statushelp* zentriert oder rechtsbündig auszurichten werden nicht unterstützt.

33.1.4.1.13 Tablefield

In einem extrem kleinen ***Tablefield***, bei dem die Kopfzeilen und -spalten nicht komplett dargestellt werden können, sind Scrollbars vorhanden. Diese haben allerdings keine Wirkung.

Das ***Tablefield*** besitzt unten und rechts eine Leerzeile bzw. Leerspalte, die den freien Bereich ausfüllen, wenn ganz nach oben bzw. rechts gescrollt wird. Wenn das Tablefield so dimensioniert ist, dass die letzte Inhaltszeile bzw. Inhaltsspalte nicht komplett dargestellt werden können, dann kann der Anwender auf diese Leerzeile bzw. Leerspalte scrollen. Diese Position kann programmatisch nicht abgefragt oder eingestellt werden.

Es wird das Selektionsmodell von Qt verwendet, daher kann es leichte Unterschiede zu den anderen Fenstersystemen geben. Das Verhalten ist dadurch aber Qt-typisch. Wenn beim Attribut *.selection* von den Elementen *sel_column*, *sel_row* und *sel_single* mehrere den Wert *true* haben, dann wird *sel_single* verwendet. Das Element *sel_header* wird ignoriert, der Header wird immer mit selektiert.

Der Rahmen des ***Tablefields*** kann lediglich über *.borderwidth = 0* ausgeschaltet und *.borderwidth = 1* eingeschaltet werden. Die Farbe und zum Teil auch die Breite des Rahmens können nicht beeinflusst werden.

Folgende Attribute sind in dieser Version noch nicht implementiert:

- *.colheadshadow*
- *.fieldshadow*
- *.rowheadshadow*
- *.xmargin*
- *.ymargin*

33.1.4.1.14 Toolbar

Die ***Toolbar*** kann unter Qt zwei unterschiedliche Ausprägungen haben, die über das Attribut *.style* eingestellt werden. Standardmäßig ist der Stil *toolbar* aktiv, der optisch den bekannten Toolbars entspricht.

Tabelle 7: *Attribut .style der Toolbar*

Wert	Beschreibung
.style = toolbar (Standardwert)	Herkömmliche Toolbar, entsprechend der bekannten IDM-Toolbar.
.style = notepage	Toolbars verwenden einen Dock-Bereich, der zwischen Toolbars und innerem Bereich liegt. „Tabbed Toolbars“ und „Nested Toolbars“ sind möglich (siehe Kapitel „Window“).

„Abbildung 14“ zeigt die Unterteilung eines Fensters mit verschiedenen Toolbars im eingedockten und ausgedockten Zustand. „Tb16“ und „Tb18“ sowie „Tb19“ und „Tb20“ sind „Tabbed Toolbars“, die sich ihren Platz teilen. „Tb17“ ist eine „Nested Toolbar“, die in die zweite Reihe des rechten Bereichs eingebettet ist.

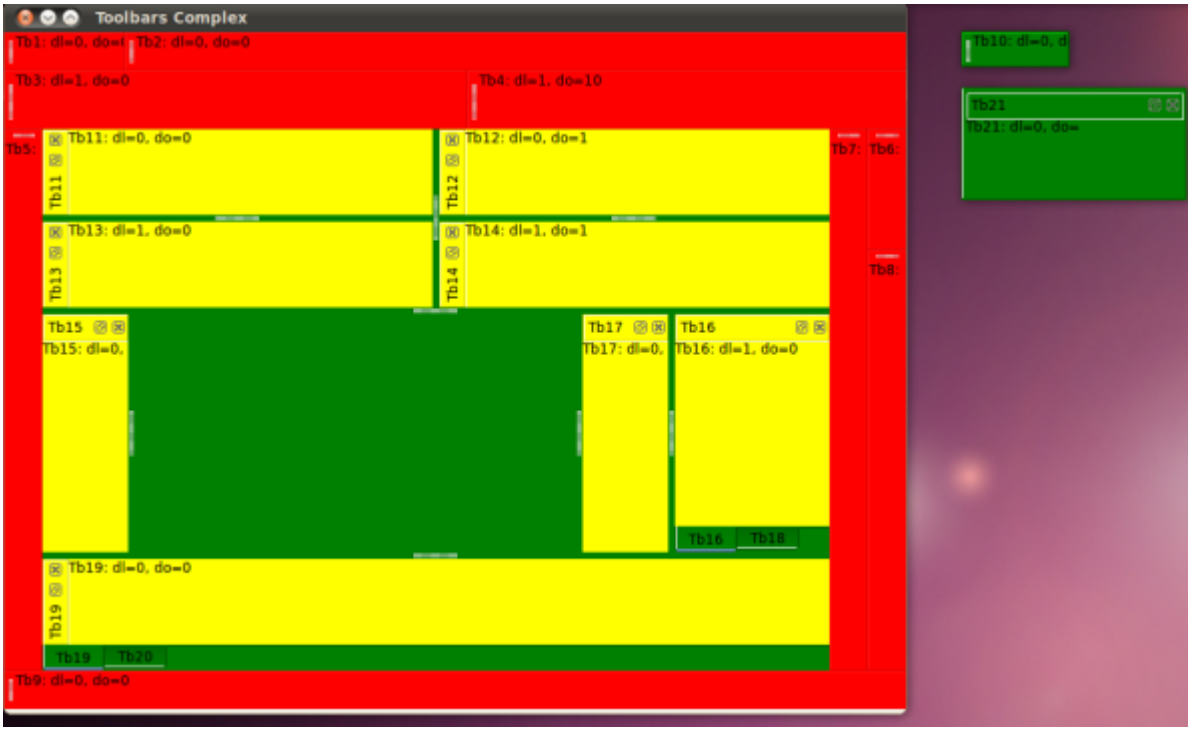


Abbildung 14: *Toolbar-Arten im IDM für Qt*

Der Stil *notepage* erlaubt ein Verschachteln mehrerer ***Toolbars*** in einem Dockbereich und ihre Anordnung in Form von Registerkarten (siehe Kapitel „Window“, „Tabelle 9“ für die entsprechenden Steuerungsoptionen). Die Toolbars haben dann eine Titelzeile und können über ihre Titelschaltflächen ausgedockt und geschlossen werden.

Toolbars unterschiedlichen Stils, die im selben Dockbereich definiert werden, können nicht gemischt werden und werden entsprechend ihrem Stil gruppiert angeordnet. Dabei werden Toolbars mit dem Stil *toolbar* immer am äußeren Fensterrand positioniert und Toolbars mit dem Stil *notepage* immer zwischen dem Clientbereich des Fensters und den Toolbars mit dem Stil *toolbar* (siehe „Abbildung 15“).

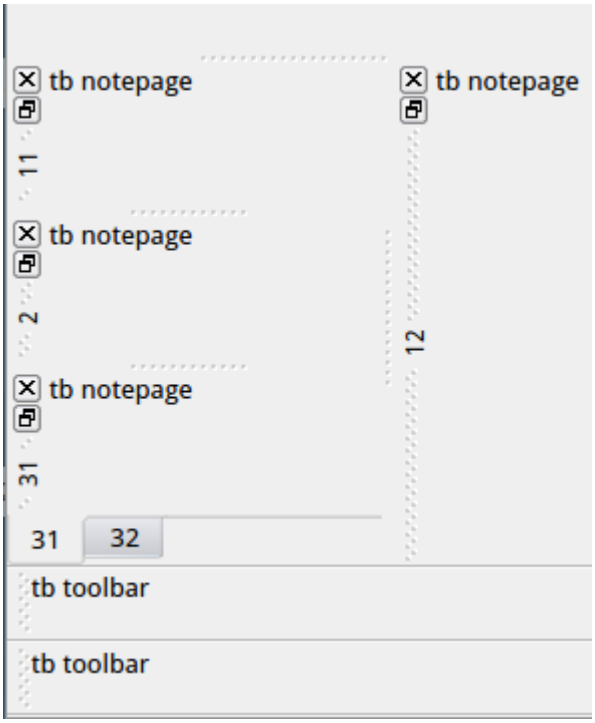


Abbildung 15: *Toolbars verschiedenen Stils im unteren Dockbereich (dock_down)*

Toolbars können nicht mit Abständen zwischen ihnen positioniert werden. Das Attribut *.autoalign* hat keine Funktion.

Toolbars mit dem Stil *toolbar* können nicht per Maus in der Größe verändert werden und erzeugen beim Verschieben keine *move*-Ereignisse.

Das Attribut *.sizeable* verhält sich bei *true* wie das Attribut *.autosize*, das heißt die gesamte verfügbare Breite oder Höhe wird ausgefüllt. Bei *false* wird die gesetzte Größe unter allen Umständen beibehalten.

Toolbars werden **nicht** automatisch in die nächste Zeile oder Spalte umgebrochen, wenn sie nicht alle in die vorgegebene Fensterbreite oder -höhe passen. Qt vergrößert das Fenster in diesem Fall entsprechend um allen Toolbars Platz zu bieten. Eine neue Zeile oder Spalte kann nur über das Attribut *.dock_line* erzwungen werden.

Bei ***Toolbars*** mit dem Stil *notepage* hängt die Positionierung von den ***Window***-Optionen *opt_nested_docks* und *opt_tabbed_docks* ab. Sind beide Optionen deaktiviert (Standard), dann wird *.dock_line* ignoriert und alle Toolbars werden nebeneinander (horizontal) oder übereinander (vertikal) angeordnet.

Ist die Option *opt_nested_docks* aktiv, dann wird *.dock_line* beachtet. Die Schachtelung erfolgt immer relativ zur vorherigen Toolbar desselben Dockbereichs. In „Abbildung 16“ haben die Toolbars „eins“, „zwei“ und „drei“ jeweils *.dock_line* = *1*, „vier“ hat *.dock_line* = *2* und nistet sich bei „drei“ ein.

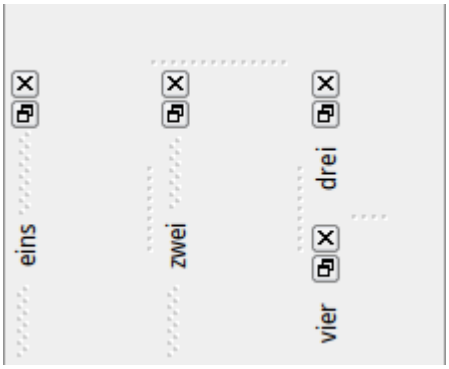


Abbildung 16: *Toolbars mit dem Stil notepage bei aktivem opt_nested_docks*

Ist die Option *opt_tabbed_docks* aktiv, dann werden alle Toolbars mit derselben *.dock_line* als hintereinander liegende Registerkarten mit Reitern dargestellt („Abbildung 17“).



Abbildung 17: *Toolbars mit dem Stil notepage bei aktivem opt_tabbed_docks*

Je nach UI-Stil wird bei *.style = notepage* unter Umständen keine Abgrenzung zum Clientbereich dargestellt.

Beim Wechsel zwischen horizontalem und vertikalem Eindocken werden Breite und Höhe nicht vertauscht, da je nach Art der Toolbar Ausrichtungswechsel bereits während des Verschiebens auftreten können.

Die Attribute *.height[]* und *.width[]* werden nur im Stil *notepage* unterstützt und angewendet.

33.1.4.1.14.1 Verhalten

Bei „Tabbed Toolbars“ wird beim Aktivieren eines Tabs bzw. dem damit verbundenen Sichtbarmachen ein *activate*-Ereignis verschickt (analog zu Notebook und Notepage).

Das Verschicken von Ereignissen wie *activate*, *deactive*, *close*, *move* und *resize* hängt stark vom verwendeten Qt-Toolbar-Objekt ab. Interne Zustände wie „Benutzer ist gerade noch am Verschieben“ können nicht berücksichtigt werden.

„Tabelle 8“ fasst Unterschiede und Einschränkungen der ***Toolbar*** des IDM FÜR QT zusammen.

Tabelle 8: *Unterschiede und Einschränkungen der Toolbar*

.style	Bedingungen	Einschränkungen und Unterschiede
alle		*.autoalign* wird nicht unterstützt.
alle	*.sizeraster = true* und *.docking <> dock_window*	Toolbar-Bereich wird nicht an Rastergröße angepasst.
alle	*.dock_offset > 0* und *.docking <> dock_window*	Toolbars lassen sich nicht mit Abstand (Leerraum) positionieren.
notepage		*.max_height*, *.max_width*, *.min_height* und *.min_width* haben keine Auswirkung.
toolbar	*.sizeable = true* und *.docking <> dock_window*	Keine Sizebar, Toolbar-Größe kann nur in einer Richtung (abhängig von der Dock-Position) beeinflusst werden. Die Größe in der anderen Richtung wird durch die maximale Breite bzw. Höhe bestimmt. Verhalten entspricht *.autosize = true*.
toolbar	*.sizeable = true* und *.docking = dock_window*	Toolbar-Größe kann nicht verändert werden.
toolbar	*.autosize = true* und *.docking <> dock_window*	Maximal mögliche Breite oder Höhe.
toolbar	*.docking <> dock_window* und gesetzte minimale bzw. maximale Breite oder Höhe	Minimal- und Maximal-Größen wirken nicht auf die Gesamtgröße der Toolbar, sondern schneiden Inhalte ab oder erzeugen leere Bereiche.
toolbar		Keine *activate*- und *deactivate*-Ereignisse.
toolbar	Verschieben mit Beibehaltung des Dock-Zustands bei *.docking = dock_window*	Keine *move*-Ereignisse.

33.1.4.1.15 Window

Das ***Window***-Objekt verfügt über vier weitere Optionen für die Unterstützung von Toolbars und „Tabbed Docks“ sowie um Größe und Position eines Fensters Qt-konform zu setzen.

Tabelle 9: *Optionen des Window-Objekts*

.options[…] Wert	Standardwert	Bedeutung
opt_window_size	***false***	*true*: Größenangaben (*.width*, *.height*, Minimal- und Maximalwerte) beziehen sich auf das gesamte Fenster, inklusive Menüleiste, Toolbars, Tabbed Widgets und Statusleiste, aber ohne Dekoration (Titelzeile, Ränder). ***false***: Größenangaben beziehen sich auf den Innenbereich („client area“) des Fensters.
opt_animated	***false***	*true*: Animiertes, interaktives Verschieben von ***Toolbars***. Dies führt allerdings zu deutlich mehr *resize*- und *move*-Ereignissen. ***false***: Keine Animationen beim Verschieben von ***Toolbars***.
opt_nested_docks	***false***	Die Option wirkt sich nur auf ***Toolbars*** mit dem Stil *notepage* aus. *true*: Nebeneinander liegende, mehrreihige Docks sind möglich. Dies führt allerdings zu einer nicht mehr so klaren Bedienung bei Interaktionen und Verschiebungen. ***false***: Nur einreihige Docks sind möglich.
opt_tabbed_docks	***false***	Die Option wirkt sich nur auf ***Toolbars*** mit dem Stil *notepage* aus. *true*: „Tabbed Docks“, die sich den Platz teilen, sind möglich. „Tabbed Docks“ lassen sich ähnlich wie Notebooks und Notepages bedienen (siehe Kapitel „Toolbar“). ***false***: „Tabbed Docks“ sind nicht möglich.

Wie unter MOTIF (und im Unterschied zu MICROSOFT WINDOWS) wirkt das Attribut *.sensitive* unter Qt nicht auf die Titelzeile des ***Fensters***. Auch wenn *.sensitive = false* ist (und sich der Fensterinhalt entsprechend nicht mehr bedienen lässt), können dennoch die Funktionen der Titelleiste (Schließen, Verschieben, Minimieren, Maximieren…) genutzt werden.

Das Attribut *.moveable* hat unter Qt keine Auswirkung. Fenster lassen sich immer verschieben.

Generell werden beim IDM FÜR QT deutlich mehr *move*- und *resize*-Ereignisse erzeugt als bei den anderen IDM-Versionen, da für jedes Pixel, um das vergrößert, verkleinert oder verschoben wird, ein entsprechendes Ereignis ausgelöst wird. Es existiert keine zuverlässige Möglichkeit, Zustand und Ende der Aktionen zu bestimmen.

Manche Fensterzustände lassen sich nur durch internes Neuanlegen („Clobbering“) ändern, wodurch allerdings *activate*- und *deactivate*-Ereignisse ausgelöst werden.

Es kann noch vorkommen, dass Größenangaben in bestimmten Konstellationen nicht korrekt umgesetzt werden (z. B. bei ihrer Änderung im minimierten Zustand). Momentan kann das Verhalten je nach Window Manager unterschiedlich sein (z. B. beim Minimieren).

33.1.4.2 Ressourcen

33.1.4.2.1 Mnemonics

Folgende Objekte unterstützen Mnemonics:

- ***Checkbox***
- ***Menubox***
- ***Menuitem***
- ***Pushbutton***
- ***Radiobutton***
- ***Statictext***

33.1.4.2.2 Color

33.1.4.2.2.1 Farbbezeichner und Null-Farbe

Als Farbbezeichner können alle SVG-Farbnamen verwendet werden (**http://www.w3.org/TR/SVG/types.html#ColorKeywords**), wobei Groß- und Kleinschreibung nicht beachtet wird. Bei der Verwendung dieser Farbnamen (zum Beispiel „green“) werden die Farben entsprechend der SVG-Farbnamen des W3C umgesetzt. Deshalb unterscheiden sich manche Farben in Ton und Intensität von den X11-Farben.

Außerdem stehen folgende, von den aktuellen Einstellungen abhängige Qt-Standardfarben zur Verfügung:

- BASE Hintergrund von Eingabefeldern und bestimmten Container-Objekten (Listen)
- BASETEXT Vordergrund von Eingabefeldern und bestimmten Container-Objekten (Listen)
- BUTTON Schaltflächenhintergrund
- BUTTONTEXT Schaltflächenvordergrund
- HIGHLIGHT Hintergrund für Selektion und aktiven Eintrag
- HIGHLIGHTTEXT Vordergrund für Selektion und aktiven Eintrag
- SHADOW Schattenfarbe, meist sehr dunkel bis schwarz
- WINDOW Fensterhintergrund
- WINDOWTEXT Fenstervordergrund

- BRIGHTTEXT Textfarbe bei schwachem Kontrast
- ALTERNATEBASE alternativ Hintergrundfarbe (meist bei Tabellen)
- TOOLTIPBASE Tooltiphintergrund
- TOOLTIPTEXT Tooltipvordergrund
- LINK Link-Vordergrund
- LINKVISITED besuchter Link-Vordergrund
- PLACEHOLDERTEXT Platzhalter Vordergrund für verschiedene Eingabe-Windgets, ab Qt 5.12
- ACTIVE_BASE
- ACTIVE_BASETEXT
- ACTIVE_BUTTON
- ACTIVE_BUTTONTEXT
- ACTIVE_HIGHLIGHT
- ACTIVE_HIGHLIGHTTEXT
- ACTIVE_SHADOW
- ACTIVE_WINDOW
- ACTIVE_WINDOWTEXT
- ACTIVE_BRIGHTTEXT
- ACTIVE_ALTERNATEBASE
- ACTIVE_TOOLTIPBASE
- ACTIVE_TOOLTIPTEXT
- ACTIVE_LINK
- ACTIVE_LINKVISITED
- ACTIVE_PLACEHOLDERTEXT ab Qt 5.12
- DISABLED_BASE
- DISABLED_BASETEXT
- DISABLED_BUTTON
- DISABLED_BUTTONTEXT
- DISABLED_HIGHLIGHT
- DISABLED_HIGHLIGHTTEXT
- DISABLED_SHADOW
- DISABLED_WINDOW
- DISABLED_WINDOWTEXT
- DISABLED_BRIGHTTEXT
- DISABLED_ALTERNATEBASE

- DISABLED_TOOLTIPBASE
- DISABLED_TOOLTIPTEXT
- DISABLED_LINK
- DISABLED_LINKVISITED
- DISABLED_PLACEHOLDERTEXT ab Qt 5.12
- INACTIVE_BASE
- INACTIVE_BASETEXT
- INACTIVE_BUTTON
- INACTIVE_BUTTONTEXT
- INACTIVE_HIGHLIGHT
- INACTIVE_HIGHLIGHTTEXT
- INACTIVE_SHADOW
- INACTIVE_WINDOW
- INACTIVE_WINDOWTEXT
- INACTIVE_BRIGHTTEXT
- INACTIVE_ALTERNATEBASE
- INACTIVE_TOOLTIPBASE
- INACTIVE_TOOLTIPTEXT
- INACTIVE_LINK
- INACTIVE_LINKVISITED
- INACTIVE_PLACEHOLDERTEXT ab Qt 5.12

Normal- und Active-Zustand sind dabei identisch.

Der Bezeichner *DEFAULT* dient als Null-Farbe. Setzt man diese Farbe, wird die jeweilige Farbe auf die entsprechende Standardfarbe (abhängig von den aktuellen Systemeinstellungen) zurückgesetzt.

33.1.4.2.2.2 Farbverläufe (Gradienten)

Color-Ressourcen unterstützen unter Qt Farbverläufe. Diese werden wie folgt definiert:

```
color {<Bezeichner>} gradient("<kind>[,<arg>]");
```

Der Parameter <*kind*> definiert die Verlaufsart. Es darf nur eine Verlaufsart angegeben werden.

Die weiteren Parameter können Folgendes enthalten:

- Ergänzungsparameter zur Verlaufsart
- Stopp-Punkt-Definitionen
- Farbdefinitionen

Diese **Verlaufsarten** stehen zur Verfügung:

***Tabelle 10:** Arten von Farbverläufen*

Verlauf	Definition	Erklärung
linear	`gradient("Linear, ...")` `gradient("LinearV, ...")`	vertikaler Verlauf
	`gradient("LinearH, ...")`	horizontaler Verlauf
radial	`gradient("Radial, ...")` `gradient("Radial, <R>, ...")`	radialer Verlauf mit Standardradius 50% radialer Verlauf mit **Ergänzungsparameter** Radius R% Radius wird in Prozent des verfügbaren Platzes angegeben
konisch	`gradient("Conical, ...")` `gradient("Conical, <S>, ...")`	konischer Verlauf mit Start bei 90° konischer Verlauf mit **Ergänzungsparameter** S°, der den Startwinkel angibt

Die **Farbdefinitionen** stehen immer nach der Verlaufsart und eventuellen Ergänzungsparametern. Es können Farbnamen, HTML-Notation sowie die für Farbressourcen bekannten Notationen *rgb(…)*, *hls(…)* und *grey(…)* verwendet werden.

Zusätzlich kann zu jeder Farbdefinition ein **Stopp-Punkt** angegeben werden, mit dem die Gewichtung der Farbe festgelegt wird. Ein Stopp-Punkt ist ein Prozentwert mit optionalem Prozentzeichen, der immer vor der jeweiligen Farbe steht und beeinflusst, wie viel Raum diese Farbe im Farbverlauf einnimmt. Ein Farbverlauf startet bei *0%* und endet bei *100%*, bezogen auf den Bereich, den er ausfüllt. Die Angabe *… , 20%, green, ...* bedeutet zum Beispiel, dass nach *20%* des zu füllenden Bereichs die Farbe Grün gesetzt wird. Wenn für den Bereich *0–20%* keine andere Farbe gesetzt ist, werden die ersten *20%* des Bereichs grün eingefärbt. Falls vor *20%* bereits eine andere Farbe gesetzt ist, dann wird ein Übergang zwischen dieser Farbe und Grün dargestellt.

Stopp-Punkte sollten immer in aufsteigender Reihenfolge gesetzt werden. Sind mehrere Farben mit demselben Stopp-Punkt definiert, gilt die Farbe, die in der Parameterliste am weitesten hinten steht. Die Definition *... , 20%, green, 40%, blue, 20%, red, ...* erzeugt beispielsweise einen Farbverlauf mit einem Rotton bei *20%*, der in einen Blauton übergeht, welcher ab *40%* satt dargestellt wird.

Beispiele

```
gradient("Linear, green, yellow, red");  benannte Farben, gleichmäßig
verteilt
gradient("Linear, #00FFFF, #00FF00, #FF0000");  HTML-Notation,
                                                Farben gleichmäßig verteilt
gradient("Linear,red, #00FF00, rgb(0,0,255)");  gemischte Notationen,
                                                Farben gleichmäßig verteilt
gradient("Linear, 20%, green, 60%, yellow, 80%, red");  benannte Farben mit
pro-
                                                        zentualen Stopp-
```

```
Punkten
gradient("Linear, 20, green, 60, yellow, 80 ,red");  das %-Zeichen bei den
                                                     Stopp-Punkten ist
optional
```

Hinweise

Qt lässt das Setzen von Farbverläufen an den meisten Stellen zu, wendet sie aber nicht unbedingt an. Ob ein Farbverlauf angezeigt wird, hängt sehr stark vom Objekt und UI-Stil ab. Dabei sind Flächen (z. B. Hintergründe) meist unproblematisch, bei filigranen Strukturen (z. B. Texten) wird der Verlauf oft durch eine einzelne Farbe ersetzt. Bei Gruppierungsobjekten werden Gradienten als Hintergrund in der Regel dargestellt.

33.1.4.2.3 Cursor

Es gibt folgende benannte Cursors, bei denen der entsprechende Qt CursorShape geladen wird:

- arrow
- blank
- busy
- closedhand
- cross
- dragcopy ab Qt 4.7
- dragmove ab Qt 4.7
- draglink ab Qt 4.7
- forbidden
- ibeam
- openhand
- pointinghand
- sizeall
- sizebdiag
- sizefdiag
- sizehor
- sizever
- splith
- splitv
- uparrow
- wait
- whatsthis

Die Darstellung der Cursors kann je nach UI-Stil unterschiedlich sein.

33.1.4.2.4 Font

Schriftarten können auf folgende Arten definiert werden:

X Logical Font Description (XLFD)

```
"-adobe-helvetica-medium-r-normal--12-*-*-*-p-*-iso8859-1"
"*adobe-helvetica-bold-r-normal-*-100-*-iso8859-1"
```

Hinweise

- QT hat Probleme bei der Verarbeitung von Wildcards in XLFDs.
- Wenn QT eine Schriftart nicht genau zuordnen kann, wird die am ehesten passende Fontfamily genommen. Es kommt zwangsläufig zur Substitution von Werten.
- Font-Aliase (Datei **fonts.alias** im Font-Verzeichnis von X11) werden von Qt nicht unterstützt. Deshalb kann zum Beispiel Font *fixed* nicht verwendet werden.

Achtung

- Seit QT5 wurden unterstützende Funktionalitäten zum Verarbeiten von XLFD Fonts einstellt.
- Die Verwendung von XLFD Fonts wird nicht empfohlen.

Qt-Stil mit kommaseparierter Parameterliste

```
Family, PointSize, PixelSize, QFont::StyleHint, QFont::Weight, QFont::Style,
Flag underline, Flag strikeout, Flag pixedPitch, Flag rawMode
```

```
"Helvetica, 12, -1, 5, 75, 1, 0, 0, 0, 0"
```

Es müssen nicht alle Parameter angegeben werden, `"Helvetica, 12"` reicht zum Beispiel.

Schriftart mit unabhängigen Darstellungsattributen (Größe und Stil)

```
"Helvetica", 12, bold
```

Die Stile *bold*, *italic* und *oblique* werden unterstützt.

Angaben im Qt-Stil können mit unabhängigen Darstellungsattributen kombiniert werden:

```
"Helvetica, 12", bold
```

33.1.4.3 Attribute

33.1.4.3.1 colorname[integer]

Attribut des ***Setup***-Objekts, das eine Liste mit den Namen der im WSI verfügbaren Farben enthält.

	Bezeichner		Datentyp
Regelsprache	**.colorname[integer]**		string
C	AT_colorname		DT_string
COBOL	AT-colorname		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	Setup		
Index-Datentyp	integer		
Zugriff	get	Standardwert	–
changed-Ereignis	nein	Vererbung	nein

Verfügbarkeit

Nur IDM FÜR QT, ab IDM-Version A.06.01.a.

33.1.4.3.2 cursorname[integer]

Attribut des ***Setup***-Objekts, das eine Liste mit den Namen der im WSI verfügbaren Cursor enthält.

	Bezeichner		Datentyp
Regelsprache	**.cursorname[integer]**		string
C	AT_cursorname		DT_string
COBOL	AT-cursorname		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	Setup		
Index-Datentyp	integer		
Zugriff	get	Standardwert	–
changed-Ereignis	nein	Vererbung	nein

Verfügbarkeit

Nur IDM FÜR QT, ab IDM-Version A.06.01.a.

33.1.4.3.3 fontname[integer]

Attribut des ***Setup***-Objekts, das eine Liste mit den Namen der im WSI verfügbaren Schriftarten enthält.

	Bezeichner		Datentyp
Regelsprache	**.fontname[integer]**		string
C	AT_fontname		DT_string
COBOL	AT-fontname		DT-string
Klassifizierung	Objektspezifisches Attribut		
Objekte	Setup		
Index-Datentyp	integer		
Zugriff	get	Standardwert	–
changed-Ereignis	nein	Vererbung	nein

Verfügbarkeit

Nur IDM FÜR QT, ab IDM-Version A.06.01.a.

33.1.4.3.4 picture und picture_hilite

Wenn bei ***Listbox*** und ***Poptext*** nur bei *.picture_hilite* ein ***Tile*** gesetzt ist, aber nicht bei *.picture*, dann zeigt Qt auch im unselektierten Zustand das in *.picture_hilite* definierte ***Tile*** an (oder eine leichte Abwandlung davon). Qt versucht für die Listeneinträge immer Bilder für alle Zustände bereitzustellen. Fehlt das Bild für einen Zustand, generiert Qt es aus den bereitgestellten Bildern.

33.1.4.3.5 statushelp

Beim Überfahren verschiedener Objekte mit der Maus wird der entsprechende Statustext angezeigt, wenn man von einem Elternobjekt auf ein Kindobjekt wechselt. Bewegt man allerdings die Maus danach wieder vom Kind auf das Elternobjekt, wird der Statustext des Elternobjekts nicht wieder angezeigt.

33.1.4.4 Schnittstellenfunktion DM_PictureHandler

Die Struktur **DM_PicInfo** unter Qt entspricht der unter Motif.

```
typedef struct {
  ...
  #  if defined(MOTIF) || defined(QT)
       DM_UInt2    type;        // Bildtyp
       DM_Pointer  image;       // Zeiger auf die Bilddaten
       DM_Pointer  trans_mask;  // Zeiger auf die Transparenzmaske
  #  endif
  ...
} DM_PicInfo;
```

Bildtyp (*type*) kann einer der folgenden Werte sein:

- *DM_GFX_QPIXMAP*
- *DM_GFX_QIMAGE*
- *DM_GFX_QICON*

Der Datentyp von *image* und *trans_mask* muss dem angegebenen Bildtyp entsprechen.

33.1.5 Bekannte Probleme und Einschränkungen dieser Version

33.1.5.1 Qt 4.6

Wir empfehlen – auch aus Gründen der Stabilität – die Qt-Version 4.8 zu verwenden:

- Unter Qt 4.6 kann es in bestimmten Konstellationen zu Zeichenproblemen kommen, sodass beispielsweise Objekte abgeschnitten wirken, Teile der Rahmen fehlen oder auch Bedienelement (Pfeile bei Poptext oder Spinbox) verzerrt dargestellt werden.
- Auch wurde beobachtet, dass Schriften nicht ganz „glatt“ sondern eher „pixelig“ wirkten.
- Es gibt Probleme bei der Tastaturnavigation bzw. der Ermittlung der gedrückten Tasten bei Remote-Konstellationen (WinAxe, VM...). Diese Probleme wurden erst unter Qt 4.8 gelöst.
- Das Attribut *.preeditsel* am ***Tablefield*** wird immer wie *presel_all* behandelt wenn ein einzeiliger Edittext angebunden ist. Der Grund ist, dass das verwendete ***QTableView***-Objekt ein **selectAll()** ausführt, bevor der Tastendruck weitergegeben wird.
- Das Scrollen innerhalb eines virtuellen Bereiches verlangsamt sich mit steigender Anzahl der Kindobjekte. Dieses Problem wurde erst unter Qt 4.8 gelöst.
- Es kann vorkommen, dass beim ***Poptext*** ein Listeneintrag zu viel angezeigt wird, wenn *.showitem* gesetzt ist.

33.1.5.2 Rahmen

Ist ein Objekt mit einem Rahmen im Stil *border_plain* (oder *border_compat* mit *.borderwidth* > *1*) horizontal negativ positioniert oder wird es innerhalb eines virtuellen Bereichs durch horizontales Scrollen

sichtbar gemacht, dann kann in der rechten unteren Ecke seines Rahmens eine diagonale Linie oder Aussparung (in der Hintergrundfarbe des Vaters) vorhanden sein.

33.1.5.3 Ereignisse

Es kann noch vorkommen, dass ein Objekt ohne zwischenzeitlichen Fokusverlust mehrfach Fokusereignisse erhält. Bestimmte Attributänderungen, die zum internen Neuanlegen von Objekten führen („Clobbering"), beeinflussen die Ereignisverarbeitung noch.

33.1.5.4 Drag-and-Drop

Die folgenden Erklärungen beziehen sich auf das Drag-and-Drop mit der Maus.

Die Ressourcen ***Source*** und ***Target*** sowie der Datentransfer zwischen Anwendungen oder Objekten wurden mit den von Qt dafür angebotenen Klassen und Ereignissen umgesetzt. Die Funktionsweise entspricht der Funktionsweise unter MICROSOFT WINDOWS, wobei nur die Typen *type_text* und *type_object* unterstützt werden. Die beim Drag-and-Drop angezeigten Cursors (während des Ziehens, Fallenlassen möglich bzw. unmöglich) werden von Qt gesteuert und sind abhängig vom UI-Stil.

Eventuelle Reaktionen mancher Objekte auf Drag-and-Drop-Aktionen (zum Beispiel das unerwünschte Scrollen von ***Listbox*** und ***Treeview*** beim Ziehen mit der Maus) bleiben zurzeit noch unberücksichtigt.

33.1.5.5 Startoption -IDMserver

Der IDM FÜR QT zeigt mit der Startoption folgende Informationen an:

- Laufzeitumgebung (runtime),
- Build-Version,
- UI-Stil (QStyle),
- Herstellername (vendor) des X11-Servers,
- Release.

33.2 Border-Attribute

33.2.1 Neuerungen

Objekt-Rahmen wurden bisher je nach WSI und seinen Möglichkeiten in unterschiedlicher Ausprägung dargestellt und beeinflussen auf unterschiedliche Weise Position und Ausdehnung eines Objekts.

Ab dem IDM Version 6 ist – auch im Sinne der Portierbarkeit von Dialogen – die Handhabung und Anwendung der Border-Attribute sowie deren Auswirkungen auf Erscheinungsbild und Geometrie eines Objekts vereinheitlicht worden.

Ab IDM Version 6 steht dem Anwender eine über alle WSIs weitestgehend einheitliche Verfügbarkeit, Umsetzung und Erweiterung der Border-Attribute sowie deren Auswirkungen auf Erscheinungsbild und Geometrie zur Verfügung.

Dadurch wird eine bessere Portierbarkeit von Dialogen auf andere WSIs gewährleistet, was bisher durch die zum Teil deutlichen Unterschiede in Geometrieumsetzung und Darstellung zwischen den einzelnen WSIs nicht der Fall war bzw. in aufwendigeren Portierungsarbeiten resultierte.

Durch das neue Attribut *.borderstyle* werden dem Anwender nun erweiterte Gestaltungsmöglichkeiten für Stil und Ausprägung von Rahmen an bestimmten Objekten geschaffen.

33.2.2 Rahmengestaltung

Das Attribut *.borderstyle* ist maßgeblich bestimmend für den Rahmenstil eines Objekts. Mögliche Stil-Optionen sind: „Rahmen aus“, „Toolkit-Rahmen“, „Linienförmiger Rahmen“, „Nach außen gewölbter 3D-Rahmen“ sowie „Nach innen gewölbter 3D-Rahmen“.

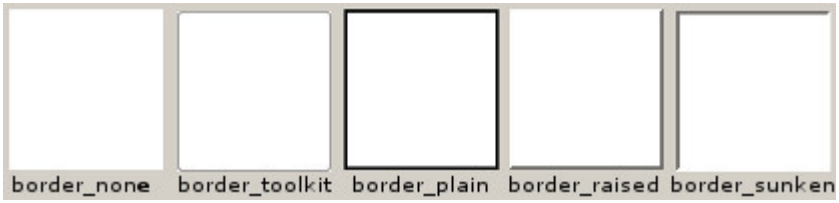


Abbildung 18: *Rahmenstile*

Folgende Objekte verfügen über das Attribut *.borderstyle*:

- ***Canvas***
- ***Edittext***
- ***Groupbox***
- ***Image***
- ***Layoutbox***
- ***Progressbar***
- ***Rectangle***
- ***Splitbox***
- ***Statusbar***
- ***Toolbar***
- ***Window***

Dabei ist der Wertebereich für die Objekte ***Edittext***, ***Image***, ***Progressbar***, ***Statusbar***, ***Toolbar*** und ***Window*** eingeschränkt auf die beiden Werte „Rahmen aus“ und „Toolkit-Rahmen“.

Das Attribut *.borderstyle* steht dabei in engem Zusammenhang mit den Attributen *.borderwidth* und *.bordercolor*. Im Stil „Linienförmiger Rahmen“ beeinflussen *.borderwidth* und *.bordercolor* Breite und

Farbe des Rahmens. Bei den beiden 3D-Rahmen kann über *.borderwidth* die Breite des Rahmens beeinflusst werden.

Dabei ist das Attribut *.borderstyle* immer vorrangig vor den Attributen *.borderwidth* und *.bordercolor*. Das heißt *.borderwidth* und *.bordercolor* werden nur beachtet wenn vom eingestellten *.borderstyle* erlaubt.

Beispiele

```
groupbox Gb
{
  .borderstyle border_plain;  // einfacher linienförmiger Rahmen
  .borderwidth 5;             // Rahmen ist 5 Pixel breit
  .bordercolor CoRed;         // Rahmen wird rot gezeichnet
}

groupbox Gb2
{
  .borderstyle border_none;  // Rahmen ausschalten
  .borderwidth 5;            // bleibt ohne Auswirkung, da ohne Rahmen
  .bordercolor CoRed;        // bleibt ohne Auswirkung, da ohne Rahmen
}

groupbox Gb3
{
  .borderstyle border_toolkit;  // Standard Toolkit-Rahmen des WSI
  .borderwidth 5;               // bleibt ohne Auswirkung, da WSI zeichnet
  .bordercolor CoRed;           // bleibt ohne Auswirkung, da WSI zeichnet
}

groupbox Gb4
{
  .borderstyle border_plain;  // einfacher linienförmiger Rahmen
  .borderwidth 0;             // wird zur Rahmen-Standardbreite
                              // des WSI umgesetzt
  .bordercolor CoRed;         // Rahmen wird rot gezeichnet
}
```

Objekte, die das Attribut *.borderstyle* nicht unterstützen, aus der Historie jedoch über die Attribute *.borderwidth* und *.bordercolor* verfügen, können diese Attribute weiterhin setzen und lesen, jedoch haben sie keinen Einfluss mehr auf die grafische Darstellung dieser Objekte.

33.2.2.1 Kompatibilität bzgl. Gestaltung

Bestehende Dialoge werden in einem **Kompatibilitätsmodus** gestartet. Das heißt, dass Objekte, die *.borderwidth* und/oder *.bordercolor* gesetzt haben und nun auch das neue Attribut *.borderstyle* unterstützen, entsprechend der Funktionsweise des IDM 5 dargestellt werden. Das heißt, bei betroffenen Objekten treten keine sichtbaren Änderungen auf.

Das Attribut *.borderstyle* ist in diesem Fall auf den Wert *border_compat* gesetzt. Der Anwender ist jedoch angehalten bei diesen Objekten den *.borderstyle* entsprechend auf eine gewünschte Variante anzupassen, da der Kompatibilitätsmodus *border_compat* zukünftig wegfallen kann. Der Kompatibilitätsmodus *border_compat* kann nicht aktiv durch den Anwender gesetzt werden.

Es gibt Objekte bei denen eine Rahmensetzung nun nicht mehr unterstützt wird und die Darstellung voll und ganz dem WSI überlassen wird. Dazu zählen ***Listbox***, ***Notebook***, ***Notepage***, ***Tablefield*** und ***Treeview***.

Bei diesen Objekten, wo *.borderwidth* und *.bordercolor* nun nicht mehr unterstützt werden, wird eine **Warnmeldung** über obsolete Attributsetzungen ausgegeben. Die Setzungen werden vom WSI nicht mehr beachtet und zeigen keine Auswirkungen. Der Anwender sollte die Stellen, an denen nun besagte Warnmeldungen kommen, bereinigen, da in zukünftigen Versionen an diesen Objekten diese Attribute wegfallen können.

33.2.2.2 Motif-Option .options[opt_xmborder_compat]

.borderwidth am ***Tablefield*** hat nun den Standardwert *0* statt früher *1*. Dies hat jedoch nur unter MOTIF Auswirkungen. Es gibt hier für MOTIF die Kompatibilitäts-Option *.options[opt_xmborder_compat]* um das ***Tablefield*** wieder in die frühere Darstellungsweise zu überführen. Eine Setzung von *.borderwidth = 1* führt auch zur früheren Darstellung, jedoch erscheint zusätzlich eine Warnmeldung da *.borderwidth* für das ***Tablefield*** zukünftig nicht mehr unterstützt wird.

33.2.2.3 WSI-Einschränkungen und -Besonderheiten

33.2.2.3.1 Motif

Der Toolkitrahmen für die ***Groupbox*** ist ein 3D-Rahmen.

Objekte mit **virtuellem Bereich** – dazu zählen die ***Groupbox*** mit *.vwidth* bzw. *.vheight* und die ***Layoutbox*** – können die Rahmenstile *border_raised* und *border_sunken* nicht darstellen, statt dessen wird ein Rahmen gezeichnet, der dem Stil *border_plain* entspricht und folglich auch die korrekte *.borderwidth* anwenden kann. Einzig eine ***Groupbox*** ohne virtuellen Bereich ist nicht eingeschränkt.

Da es unter MOTIF zu Zeichnungsproblemen kommen kann, wenn ein Objekt Position *0* oder *negativ* hat, wird in diesem Fall das Zeichnen des Rahmens unterlassen.

33.2.2.3.2 Windows

Die Objekte ***Canvas*** und ***Rectangle*** besaßen im IDM 5 einen 3-dimensionalen, farbigen Rahmen. Dieser Rahmen ist nur im Rahmenstil *border_compat* verfügbar. Mit den neuen Rahmenstilen gibt es entweder 3-dimensionale Rahmen in der Systemfarbe oder flache, farbige Rahmen.

Die Objekte ***Notepage*** und ***Tablefield*** besitzen kein *.borderstyle*-Attribut und somit auch keine Kompatibilität zum IDM 5. Das ***Tablefield***-Objekt besitzt nun eine Standardrahmenbreite und das Objekt ***Notepage*** besitzt nun keinen eigenen Rahmen mehr.

Die Höhe des ***Poptext***-Objekts wurde im IDM 5 um *2* Pixel zu groß berechnet, wenn „Visual Styles“ aktiv waren. Dies ist jetzt korrigiert, Ausnahme im Stil *listbox*, dort stimmte die Höhe.

Die Stile *border_raised* und *border_sunken* setzen eine Minimalbreite von *2* Pixeln voraus, um ggf. das Feedback darzustellen. Das heißt, wenn der Anwender hier eine *.borderwidth* von *0* oder *1* Pixel angegeben hat, wird der Rahmen trotzdem mit *2* Pixeln gezeichnet.

33.2.2.3.3 Qt

Bei folgenden Objekten bleibt das Attribut *.borderstyle* komplett ohne Auswirkungen:

- ***Progressbar***
- ***Statusbar***
- ***Toolbar***
- ***Window***

Das Objekt ***Spinbox*** zeichnet unter Qt immer einen Rahmen. Das Attribut *.borderstyle* bestimmt hierbei lediglich, wo dieser Rahmen gezeichnet wird. Bei *border_none* wird der Rahmen innerhalb des Objekts gezeichnet, bei *.borderstyle* = *border_toolkit* außerhalb.

Die Rahmen werden hier maßgeblich durch den eingestellten UI-Stil bestimmt und gezeichnet. Die Darstellung kann je nach UI-Stil variieren.

33.2.3 Geometrie

Gruppierungsobjekte zeichnen ihren Rahmen nun WSI-übergreifend immer außerhalb des Objekts.

Die Geometrieberechnung – egal ob normal oder mit Raster – erfolgt nach bisherigen Regeln und auf Basis der Referenzpunkte *.xleft* und *.ytop* sowie *.xright* und *.ybottom*.

Jedoch beschreiben diese Punkte sowie *.width* und *.height* nur die Objektposition und -größe exklusive des Rahmens. Sie definieren den **Clientbereich** des Gruppierungsobjekts. Dadurch wird den Kindobjekten der maximal verfügbare Platz geboten. Der Rahmen wird vom WSI **außen** um diese Dimension herum gezeichnet.

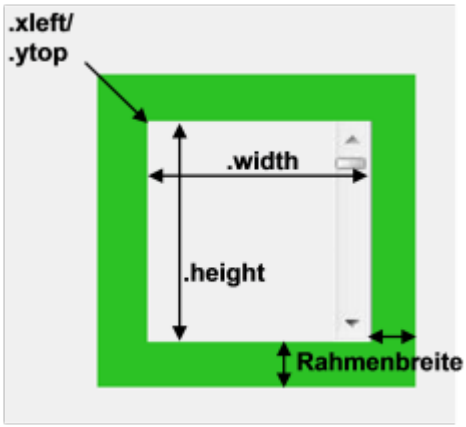


Abbildung 19: *Position und Größe bei Gruppierungsobjekten mit Rahmen (Beispiel border_plain)*

Als Ausnahme ist hier das ***Window***-Objekt zu nennen, an dessen Positionierung und Dimensionen sich nichts ändert.

Beispiel

```
Groupbox Gb {
  .width  100;  // Der Innenbereich der Groupbox hat eine Größe von
  .height 50;   // 100x50 Pixeln.
  .xleft  20;   // Der äußerste Punkt des Innenbereichs ist [20,15].
  .ytop   15;
  .borderstyle border_plain;
  .borderwidth 5;
}
// => Die eigentliche Größe des Objekts inklusive Rahmen beträgt 110x60
//    (Breite + Rahmen links + Rahmen rechts sowie Höhe + Rahmen oben +
//    Rahmen unten). Äußerster Punkt links oben ist [15,10].
```

Bei **Nicht-Gruppierungsobjekten** ergeben sich keine Änderungen bzgl. der Geometrie. Sie zeichnen ihren Rahmen weiterhin **innen**. Die Attribute *.xleft* und *.ytop* sowie *.xright* und *.ybottom* definieren somit die äußersten Punkte, *.width* und *.height* die vollständige Größe des Objekts.

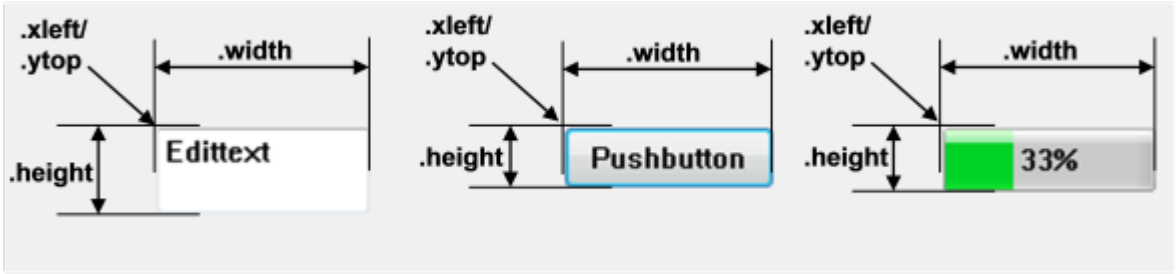


Abbildung 20: *Positionierung und Größe bei einfachen Objekten*

33.2.3.1 Kompatibilität bzgl. Positionierung

Unter MOTIF kann es bei bestehenden Dialogen zu einer leichten Verschiebung von Gruppierungsobjekten kommen, da die reale Größe von Gruppierungsobjekten mit Rahmen nun größer ist als vorher und die reale Position um eine Rahmenbreite nach links und oben verschoben ist. Das heißt aber auch, dass der Clientbereich solcher Gruppierungsobjekte nun größer ist und mehr Platz für seine Kinder bietet.

Weiterhin ergibt sich daraus, dass mehrere ***Groupboxen***, die über *.xauto = 0* und *.yauto = 0* ineinander geschachtelt wurden nun optimal den Platz ausnutzen und selbst, sowie deren Kinder nicht mehr beschnitten dargestellt werden.

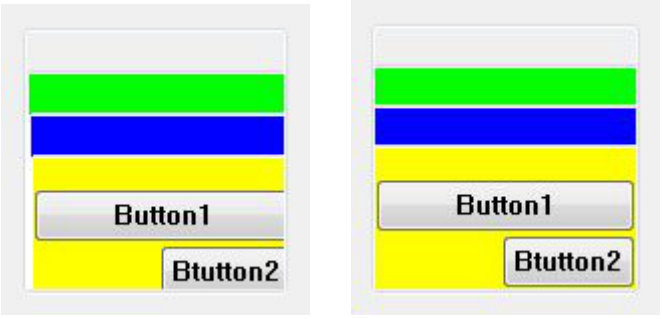


Abbildung 21: *Geschachtelte Groupboxen im IDM 5 (links) und im IDM 6 (rechts)*

33.2.3.2 Startoption -IDMborder5_compat

Unter MOTIF sind Objekte mit negativen Koordinaten sowie deren Kinder zum Teil nicht navigierbar und fokussierbar. Dies ist auch der Fall, wenn das Objekt nur zum Teil nicht im sichtbaren Bereich ist. Dies ist eine Einschränkung des WSI.

Dadurch, dass Rahmen bei Gruppierungsobjekten jetzt immer außen gezeichnet werden und es somit zu einer Positionsverschiebung kommt, kann es sein, dass bei bestehenden Dialogen Gruppierungobjekte und deren Kinder nun plötzlich nicht mehr navigierbar sind, da beispielsweise der Rahmen ins Negative gerückt wurde. Daher wird ein Geometrie-Kompatibilitätsmodus über eine Startoption `-IDMborder5_compat` eingeführt, welche die Geometrieberechnung nach IDM-5-Regeln vornimmt.

Unter MICROSOFT WINDOWS gibt es eine ganze Reihe von Kompatibilitätsoptionen, die nicht mehr ausgewertet werden. Wenn bestehende Dialoge nicht sofort angepasst werden können, kann die Auswertung der Kompatibilitätsoptionen über die Startoption `-IDMborder5_compat` wieder eingeschaltet werden. Folgende Optionen (Attribut *.options[]*) werden beeinflusst:

- *opt_wntsizebug_compat*
- *opt_w2kprefsize_compat*
- *opt_win31_sizing*

Außerdem wird die Höhe des ***Poptext***-Objekts wie im IDM 5 berechnet.

Der Modus „border5_compat“ ist auch mittels gesetzter Umgebungsvariable `IDM_BORDER5_COMPAT` aktivierbar.

33.2.4 Attribut .borderstyle

Dieses Attribut definiert den Stil, also die Darstellung und Ausprägung des Rahmens eines Objekts.

	Bezeichner		Datentyp
Regelsprache	**.borderstyle**		enum
C	AT_borderstyle		DT_enum
COBOL	AT-borderstyle		DT-enum
Klassifizierung	Layoutattribut		
Zugriff	get, set	Standardwert	*border_compat*
changed-Ereignis	ja	Vererbung	ja

Dabei ist folgender Wertebereich vorgegeben:

- *border_compat*
- *border_none*
- *border_plain*
- *border_raised*
- *border_sunken*
- *border_toolkit*

Wert	Bedeutung
border_compat	Kompatibilitätsmodus
border_none	es wird kein Rahmen (Begrenzung) um das Objekt gezeichnet
border_plain	der Rahmen wird als einfache, flache Linie um das Objekt gezeichnet, in Breite und Farbe beeinflussbar *
border_raised	der Rahmen wird als 3D-Rahmen um das Objekt gezeichnet, der Rahmen ist nach außen erhoben und in der Breite beeinflussbar *
border_sunken	der Rahmen wird als 3D-Rahmen um das Objekt gezeichnet, der Rahmen ist nach innen gedrückt und in der Breite beeinflussbar *
border_toolkit	es wird der Toolkitrahmen gezeichnet, wie vom WSI für das jeweilige Objekt vorgesehen

* sofern vom jeweiligen WSI unterstützt

Tabelle 11: *Unterstützung des Attributs .borderstyle*

Objekt	Unterstützung von .borderstyle
canvas	Attribut wird unterstützt.
groupbox	Attribut wird unterstützt.
layoutbox	Attribut wird unterstützt.
rectangle	Attribut wird unterstützt.
splitbox	Attribut wird unterstützt.
edittext	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
image	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
progressbar	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
spinbox	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
statusbar	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
toolbar	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
window	Attribut wird unterstützt, jedoch nur *border_none* und *border_toolkit* zulässig. *border_plain*, *border_raised* und *border_sunken* werden als *border_toolkit* umgesetzt.
checkbox	Attribut wird **nicht** unterstützt.
control	Attribut wird **nicht** unterstützt.
listbox	Attribut wird **nicht** unterstützt.
notebook	Attribut wird **nicht** unterstützt.

Objekt	Unterstützung von .borderstyle
notepage	Attribut wird **nicht** unterstützt.
poptext	Attribut wird **nicht** unterstützt.
pushbutton	Attribut wird **nicht** unterstützt.
radiobutton	Attribut wird **nicht** unterstützt.
scrollbar	Attribut wird **nicht** unterstützt.
statictext	Attribut wird **nicht** unterstützt.
tablefield	Attribut wird **nicht** unterstützt.
treeview	Attribut wird **nicht** unterstützt.

33.2.5 Ergänzungen im Editor

Über das Popup *„Properties“* kann im Bereich *„Object“* über den Reiter *„Layout“* der *„Borderstyle“* eines Objekts gewählt werden, sofern dieses Objekt das Attribut unterstützt.

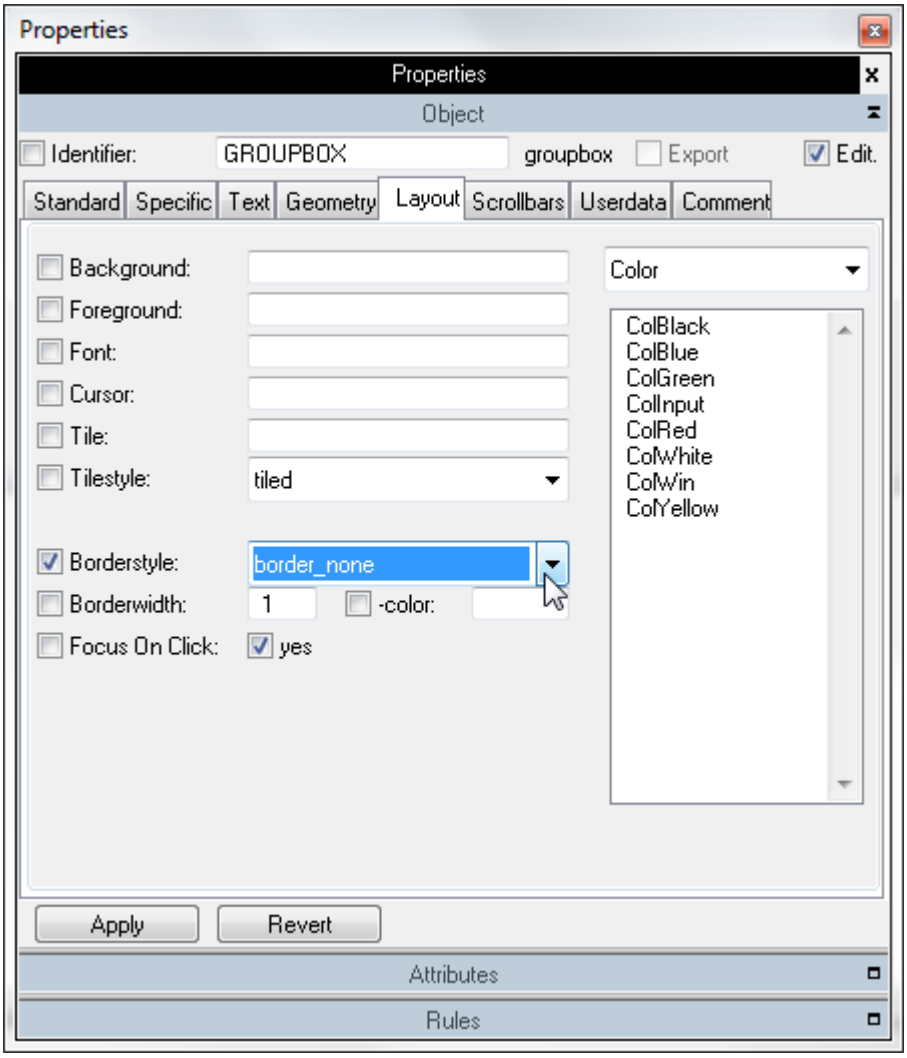


Abbildung 22: *Ändern von .borderstyle im IDM-Editor*

Da der Kompatibilitätsmodus nicht aktiv setzbar ist, wird der Anwender nach dem Umsetzen auf *border_compat* beim *„Apply“* eine Fehlermeldung erhalten.

Um den Standard *border_compat* wieder herzustellen empfiehlt es sich daher, das Häkchen vor *„Borderstyle“* zu entfernen. Somit wird wieder der Standard angewandt (es sei denn einer der Väter hat einen anderen Wert gesetzt, dann wird dieser geerbt).

33.3 Neue Objekte datetime und listview

33.3.1 datetime

Das ***datetime***-Objekt (Klassenname ***dmw_datetime***) stellt eine einfache und intuitive Eingabemöglichkeit für Datums- und Zeitwerte zur Verfügung. Es kann Datums- oder Zeitwerte oder beides in unterschiedlichen Formaten anzeigen. Neben der Eingabe per Tastatur kann der Anwender die Werte ändern, indem er zum Beispiel ein Datum aus einem Kalender auswählt (oberes ***datetime***-

Objekt in „Abbildung 23“) oder mit Spinbuttons einstellt (unteres ***datetime***-Objekt in „Abbildung 23“). Außerdem kann dem Anwender die Möglichkeit gegeben werden, den Wert mithilfe einer Checkbox auf „unbestimmt“ zu setzen (rechter Teil von „Abbildung 23“).

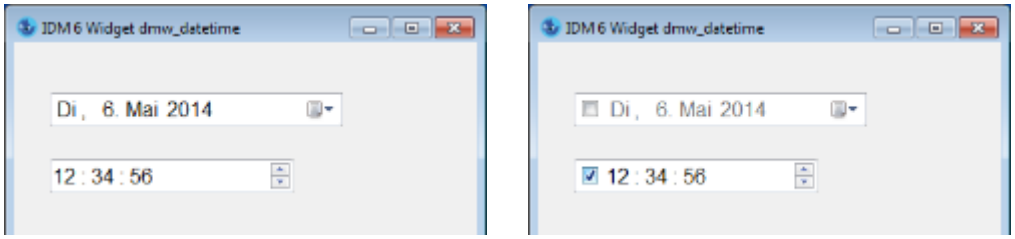


Abbildung 23: *datetime-Objekte*

„Abbildung 24“ zeigt ein ***datetime***-Objekt mit aufgeklapptem Kalender. Die Darstellung des Kalenders kann durch mehrere Attribute angepasst werden. Beispielsweise werden im rechten Teil von „Abbildung 24“ die Nummern der Kalenderwochen und das heutige Datum im Kalender angezeigt.

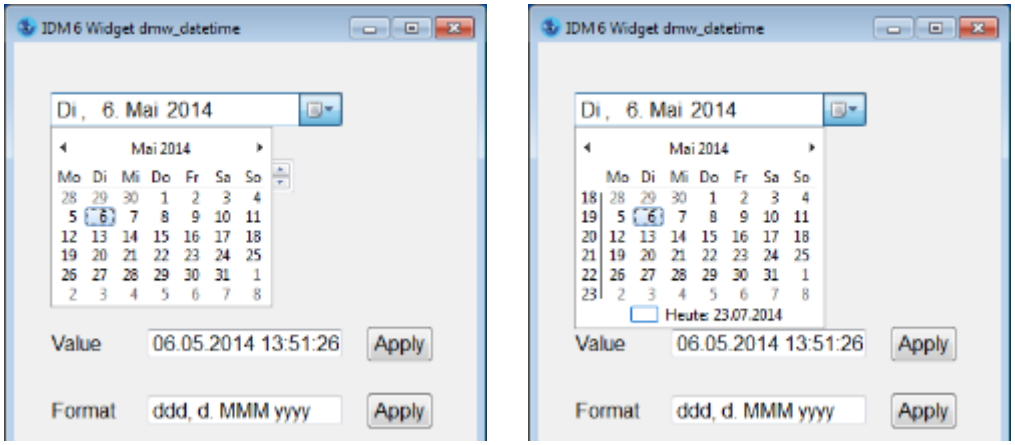


Abbildung 24: *datetime-Objekt mit aufgeklapptem Kalender*

Die Darstellung der Datums- und Zeitwerte kann über einen Formatstring festgelegt werden. Man kann vordefinierte Systemformate nutzen oder mit den entsprechenden Formatierungszeichen eigene Anzeigeformate definieren. In beiden Fällen werden die Sprach- und Regionseinstellungen des Systems berücksichtigt (zum Beispiel bei den Monatsnamen). „Abbildung 25“ zeigt ***datetime***-Objekte mit Formaten, die außer Teilen des Datums oder der Zeit noch zusätzlichen, nicht editierbaren, Text enthalten.

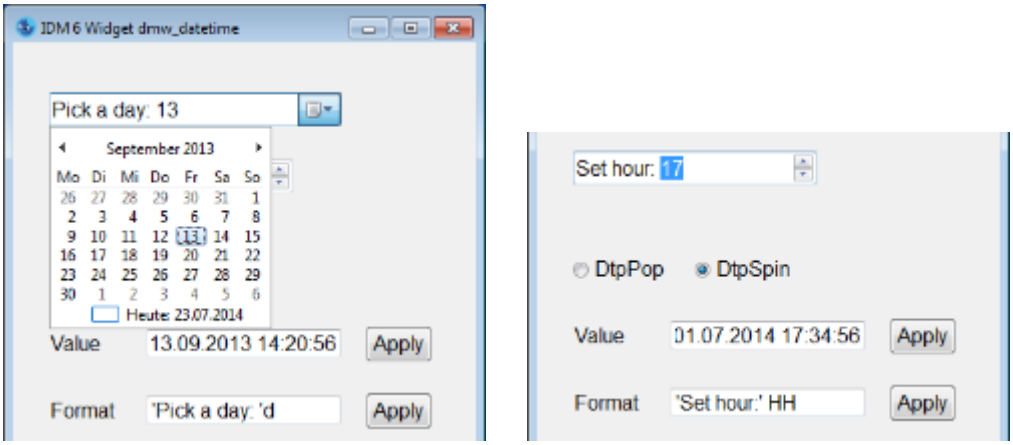


Abbildung 25: *datetime-Objekte mit unterschiedlichen Formaten*

Verfügbarkeit

- Nur unter MICROSOFT WINDOWS.
- Kann nur mit der USW-Option des ISA Dialog Managers verwendet werden.
- Die **idmwidgets.dll** muss sich im USW-Klassenpfad befinden (Standard: *<IDM-Installationsverzeichnis>\uswclasses*).

33.3.1.1 Definition

```
{ export | reexport } { model } dmw_datetime { <Bezeichner> }
{
  <Standardattribute>
  <Geometrieattribute>
  <Hierarchieattribute>
  <Layoutattribute>
  <Textattribute>
  <Objektspezifische Attribute>
}
```

Ereignisse	*changed*	*deselect*	*extevent*
	focus	*help*	*select*
Kinder	Keine		
Vater	***groupbox***	***layoutbox***	***notepage***
	splitbox	***toolbar***	***window***
Menü	Popup Menü		

33.3.1.2 Beschreibung der Ereignisse

33.3.1.2.1 deselect

Das *deselect*-Ereignis tritt auf, wenn das ***datetime*** den Fokus verloren hat. Dies ist ein Hinweis darauf, dass der Anwender die Eingabe oder Auswahl von Datum oder Zeit abgeschlossen hat. Dabei ist allerdings zu beachten, dass dieses Ereignis nicht nur durch direkte Anwenderaktionen ausgelöst wird, sondern auch als Nebeneffekt anderer Aktionen auftreten kann:

- Der Anwender setzt den Fokus mit der Maus oder Tastatur auf ein anderes Objekt.
- Der Anwender aktiviert eine andere Anwendung oder eine andere Anwendung drängt sich in den Vordergrund und zieht den Fokus auf sich.

Hinweis

Das Öffnen eines Menüs oder Popups setzt nicht unbedingt den Fokus um. Daher kann nicht vorhergesagt werden, ob bei diesen Aktionen ein *deselect*-Ereignis auftritt. Zum Beispiel kann es sein, dass ein *deselect*-Ereignis erst auftritt, während die *select*-Ereignisregel eines ***menuitems*** ausgeführt wird, wenn es überhaupt auftritt.

33.3.1.2.2 select

Das *select*-Ereignis tritt jedes Mal auf, wenn sich der Wert des Attributs *.value* ändert. Das Ereignis kann bei unterschiedlichen Aktionen des Anwenders ausgelöst werden:

- Der Anwender bearbeitet den Wert im Eingabefeld des ***datetime***.
- Der Anwender navigiert im Kalender, um zu einem bestimmten Datum zu gelangen.
- Der Anwender wählt mit Maus oder Tastatur einen Wert aus.
- Der Anwender scrollt mit der Spinbox des ***datetime***.

Das heißt, in der Regel treten mehrere *select*-Ereignisse auf, bevor der Anwender den endgültigen Wert eingegeben, eingestellt oder ausgewählt hat. Daher sollten in der Ereignisregel des *select*-Ereignisses keine umfangreichen Aktionen ausgeführt werden.

Mit dem Attribut *.real_modified* kann geprüft werden, ob sich der Wert des ***datetime*** durch die Anwenderaktionen tatsächlich verändert hat, und zwar im Vergleich zum Wert des ***datetime***, als es den Fokus erhalten hat.

Das *select*-Ereignis kann auch bei einer Verletzung von *.minvalue* oder *.maxvalue* auftreten. Dies hängt vom Systemobjekt ab, wobei auch hierbei eine Wertänderung von *.value* vorliegen muss.

33.3.1.3 Attributübersicht

Attribut	Kurzbeschreibung	Datentyp	Standard	Zugriff		C	V
				get	set		
.allowundefined	Zulassen unbestimmter Werte	*boolean*	*false*	x	x	x	x
.calendaralignment	Ausrichtung des Kalenders	*integer* (*-1*, *0*, *1*)	*1*	x	x	x	x
.format	Anzeigeformat	*string*	""	x	x	x	x
.maxvalue	Maximalwert	*string*	""	x	x	x	x
.minvalue	Minimalwert	*string*	""	x	x	x	x
.real_modified	zeigt tatsächliche Änderung von *.value* an	*boolean*		x	–	–	–
.shortdaynames	Anzeige kurzer Wochentagsnamen	*boolean*	*false*	x	x	x	x
.style	Werteingabe über aufklappbaren Kalender oder Spinbox	*class* (*poptext*, *spinbox*)	*poptext*	x	x	x	x
.today	Anzeige des aktuellen Datums	*boolean*	*true*	x	x	x	x
.todaymarker	Markierung des aktuellen Datums	*boolean*	*true*	x	x	x	x
.trailingdates	Anzeige von Tagen des vorherigen oder folgenden Monats	*boolean*	*true*	x	x	x	x
.value	Datums- und/oder Zeitwert	*string*	""	x	x	x	x
.weeknumbers	Anzeige der Kalenderwoche	*boolean*	*false*	x	x	x	x

C *changed*-Ereignis bei Änderung

V Attribut wird vererbt

33.3.1.4 Beschreibung der Attribute

33.3.1.4.1 .allowundefined

Dieses Attribut bestimmt, ob der Anwender den Wert des Objekts auf „unbestimmt“ setzen kann.

	Bezeichner		Datentyp
Regelsprache	**.allowundefined**		boolean
C	AT_allowundefined		DT_boolean
COBOL	AT-allowundefined		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*false*
changed-Ereignis	ja	Vererbung	ja

Wenn das Attribut den Wert *true* besitzt, wird im ***datetime*** eine Checkbox angezeigt. Ausschalten der Checkbox setzt den Wert auf „unbestimmt“. Das Attribut *.value* hat dann den Wert *""*.

Bei *.allowundefined* = *true* führen ungültige Werte von *.value* dazu, dass die Checkbox eingeschaltet wird.

Wertebereich

true	Der Wert von *.value* kann vom Anwender auf „unbestimmt“ gesetzt werden. Dafür wird im ***datetime*** eine Checkbox angezeigt.
false	Der Anwender kann *.value* **nicht** auf „unbestimmt“ (*""*) setzen.

Hinweis

Das Ändern *.allowundefined* im sichtbaren Zustand kann zu einem Zurücksetzen des Objekts führen.

33.3.1.4.2 .calendaralignment

Dieses Attribut definiert die Position des aufklappbaren Kalenders bezüglich des Objekts.

	Bezeichner	Datentyp
Regelsprache	**.calendaralignment**	integer
C	AT_calendaralignment	DT_integer

	Bezeichner		Datentyp
COBOL	AT-calendaralignment		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*1*
changed-Ereignis	ja	Vererbung	ja

Wertebereich

1 Der Kalender ist **links** ausgerichtet (Standard).

0 Der Kalender ist **zentriert**.
Dieser Wert wird nicht unterstützt; es wird der Standardwert verwendet.

-1 Der Kalender ist **rechts** ausgerichtet.

33.3.1.4.3 .format

Dieses Attribut definiert das Anzeigeformat des ***datetime***-Objekts.

	Bezeichner		Datentyp
Regelsprache	**.format**		string, object (text)
C	AT_format		DT_string, DT_text
COBOL	AT-format		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*""*
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Wertebereich

"" kurzes Datumsformat (wie *"widget:d"*)

widget:d	Systemformat: kurzes Datumsformat
widget:dd	Systemformat: kurzes Datumsformat mit 4-stelliger Jahreszahl
widget:ddd	Systemformat: langes Datumsformat
widget:t	Systemformat: Zeitformat
<format_string>	Formatstring zur Definition des Anzeigeformats

Systemformate

Die **Systemformate** sind abhängig von den Sprach- und Regionseinstellungen des Betriebssystems. Sie bestimmen das Anzeigeformat jeweils vollständig und sind weder miteinander noch mit Formatstrings kombinierbar. Es kann zum Beispiel keine Beschriftung zu einem Systemformat hinzugefügt werden.

Formatstrings

Formatstrings bestehen aus Formatierungszeichen, die definieren, welche Teile eines Datums angezeigt werden und wie diese Teile angezeigt werden.

Tabelle 12: *Formatierungszeichen des datetime-Objekts*

Formatierungszeichen	Anzeige
d	Tag mit so vielen Stellen wie notwendig
dd	Tag mit 2 Stellen, gegebenenfalls mit führender *„0“*
ddd	Name des Wochentags (sprachabhängig), abgekürzt auf 3 Zeichen
dddd	vollständiger Name des Wochentags (sprachabhängig)
h	Stunde im 12-Stundenformat mit so vielen Stellen wie notwendig
hh	Stunde im 12-Stundenformat mit 2 Stellen, gegebenenfalls mit führender *„0“*
H	Stunde im 24-Stundenformat mit so vielen Stellen wie notwendig
HH	Stunde im 24-Stundenformat mit 2 Stellen, gegebenenfalls mit führender *„0“*
m	Minute mit so vielen Stellen wie notwendig
mm	Minute mit 2 Stellen, gegebenenfalls mit führender *„0“*
M	Monat mit so vielen Stellen wie notwendig

Formatierungszeichen	Anzeige
MM	Monat mit 2 Stellen, gegebenenfalls mit führender *„0“*
MMM	Monatsname (sprachabhängig), abgekürzt auf 3 Zeichen
MMMM	vollständiger Monatsname (sprachabhängig)
s	Sekunde mit so vielen Stellen wie notwendig
ss	Sekunde mit 2 Stellen, gegebenenfalls mit führender *„0“*
t	Vormittags- bzw. Nachmittagskennzeichen (sprachabhängig, im Englischen *„AM“* und *„PM“*) mit einem Zeichen; Verwendung wird nicht empfohlen
tt	Vormittags- bzw. Nachmittagskennzeichen (sprachabhängig, im Englischen *„AM“* und *„PM“*) mit 2 Zeichen
yy	Jahr mit 2 Stellen
yyyy	Jahr mit 4 Stellen

Andere Zeichen als die genannten Formatierungszeichen werden in die Anzeige übernommen. Zeichen, die angezeigt werden sollen, können optional in einfache Anführungszeichen (') eingeschlossen werden. Sie müssen in einfache Anführungszeichen eingeschlossen werden, wenn sie einem Formatierungszeichen entsprechen. Um ein einfaches Anführungszeichen anzuzeigen, muss es im Formatstring verdoppelt werden ('').

Beispiel

Der Formatstring *'Heute ist 'dddd', der 'dd'. 'MMM yyyy* ergibt folgende Anzeige: *„Heute ist Freitag, der 04. Jul 2014“*.

Hinweise

- Bei einem fehlerhaften Formatstring wird das systemabhängige Standardformat verwendet (entspricht *.format = ""*).
- Die für *ddd*, *dddd*, *MMM*, *MMMM*, *t* und *tt* angezeigten Namen und Kennzeichen werden durch die Sprach- und Regionseinstellungen des Systems bestimmt.
- Bei langen Anzeigetexten kann es sein, dass auf der Schaltfläche zum Öffnen des Kalenders kein Kalendersymbol mehr dargestellt wird.

33.3.1.4.4 .maxvalue

Mit diesem Attribut wird der Maximalwert festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.maxvalue**		string, object (text)
C	AT_maxvalue		DT_string, DT_text
COBOL	AT-maxvalue		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Wertebereich

""	Kein Maximalwert gesetzt.
<*value_string*>	Setzt den enthaltenen Datums- oder Zeitwert als Maximalwert. Syntax und Auswertung des Strings sind im Kapitel „.value“ beschrieben.

Das Attribut definiert den größten Wert, der vom Anwender eingegeben oder ausgewählt werden kann. Ein vorhandener Wert von *.value* wird in der Regel nicht korrigiert um *.maxvalue* einzuhalten. Es ist auch nicht definiert, ob beim Sichtbarmachen des ***datetime*** der angezeigte Wert korrigiert wird.

33.3.1.4.5 .minvalue

Mit diesem Attribut wird der Minimalwert festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.minvalue**		string, object (text)
C	AT_minvalue		DT_string, DT_text
COBOL	AT-minvalue		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Wertebereich

""	Kein Minimalwert gesetzt.
<*value_string*>	Setzt den enthaltenen Datums- oder Zeitwert als Minimalwert. Syntax und Auswertung des Strings sind im Kapitel „.value“ beschrieben.

Das Attribut definiert den kleinsten Wert, der vom Anwender eingegeben oder ausgewählt werden kann. Ein vorhandener Wert von *.value* wird in der Regel nicht korrigiert um *.minvalue* einzuhalten. Es ist auch nicht definiert, ob beim Sichtbarmachen des ***datetime*** der angezeigte Wert korrigiert wird.

33.3.1.4.6 .real_modfied

Gibt an, ob das Attribut *.value* seit dem Erhalt des Fokus geändert wurde.

	Bezeichner		Datentyp
Regelsprache	**.real_modified**		boolean
C	AT_real_modified		DT_boolean
COBOL	AT-real-modified		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get	Standardwert	–
changed-Ereignis	nein	Vererbung	nein

Ein Setzen des Attributs *.value* setzt *.real_modified* zurück auf *false*.

Bei einem unsichtbaren ***datetime***-Objekt ist *.real_modified* immer *false*.

Wertebereich

false	Der Wert von *.value* ist unverändert.
true	Der Wert von *.value* hat sich geändert.

33.3.1.4.7 .shortdaynames

Dieses Attribut bestimmt, ob im aufklappbaren Kalender kurze Wochentagsnamen angezeigt werden.

	Bezeichner		Datentyp
Regelsprache	**.shortdaynames**		boolean
C	AT_shortdaynames		DT_boolean
COBOL	AT-shortdaynames		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*false*
changed-Ereignis	ja	Vererbung	ja

Das Attribut zeigt erst ab WINDOWS VISTA Wirkung.

Wertebereich

false	Im Kalender werden normale Wochentagsnamen angezeigt.
true	Im Kalender werden kurze Wochentagsnamen angezeigt.

Hinweis

Im Deutschen besteht kein Unterschied zwischen kurzen und normalen Wochentagsnamen.

33.3.1.4.8 .style

Das Attribut bestimmt das Aussehen und die Art der Bedienung des ***datetime***-Objekts.

	Bezeichner		Datentyp
Regelsprache	**.style**		class
C	AT_style		DT_class
COBOL	AT-style		DT-class
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*poptext*
changed-Ereignis	ja	Vererbung	ja

Wertebereich

Wert	Beschreibung
poptext	Zum Auswählen eines Datums kann ein Kalender geöffnet werden.
spinbox	Werte können über eine Spinbox eingestellt werden.

Hinweis

Das Ändern des Attributs im sichtbaren Zustand führt zu einem Zurücksetzen des Objekts und sollte vermieden werden.

33.3.1.4.9 .today

Das Attribut bestimmt, ob am unteren Rand des aufklappbaren Kalenders das heutige Datum angezeigt wird.

	Bezeichner		Datentyp
Regelsprache	**.today**		boolean
C	AT_today		DT_boolean
COBOL	AT-today		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*true*
changed-Ereignis	ja	Vererbung	ja

Das Attribut zeigt erst ab WINDOWS VISTA Wirkung.

Wertebereich

false	Im Kalender wird das heutige Datum nicht angezeigt.
true	Im Kalender wird am unteren Rand das heutige Datum angezeigt.

33.3.1.4.10 .todaymarker

Dieses Attribut legt fest, ob das heutige Datum im aufklappbaren Kalender markiert wird.

	Bezeichner	Datentyp
Regelsprache	**.todaymarker**	boolean

	Bezeichner		Datentyp
C	AT_todaymarker		DT_boolean
COBOL	AT-todaymarker		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*true*
changed-Ereignis	ja	Vererbung	ja

Das Attribut zeigt erst ab WINDOWS VISTA Wirkung.

Wertebereich

false	Im Kalender wird das heutige Datum nicht markiert.
true	Im Kalender wird das heutige Datum markiert.

33.3.1.4.11 .trailingdates

Dieses Attribut bestimmt, ob im aufklappbaren Kalender die an den aktuellen Monat angrenzenden Tage angezeigt werden.

	Bezeichner		Datentyp
Regelsprache	**.trailingdates**		boolean
C	AT_trailingdates		DT_boolean
COBOL	AT-trailingdates		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*true*
changed-Ereignis	ja	Vererbung	ja

Das Attribut zeigt erst ab WINDOWS VISTA Wirkung.

Wertebereich

false	Im Kalender werden nur Tage des eingestellten Monats angezeigt.
true	Im Kalender werden auch Tage des vorherigen und folgenden Monats angezeigt, sodass Wochen immer vollständig angezeigt werden.

33.3.1.4.12 .value

Dieses Attribut legt den Wert des ***datetime***-Objekts fest.

	Bezeichner		Datentyp
Regelsprache	**.value**		string, object (text)
C	AT_value		DT_string, DT_text
COBOL	AT-value		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Wertebereich

""
: Das aktuelle Datum bzw. die aktuelle Uhrzeit werden angezeigt.

<value_string>
: Setzt den enthaltenen Datums- oder Zeitwert.

Syntax und Auswertung des Wertstrings

Der Wert wird entsprechend der „Definition der Zeitabstände“ für das timer-Objekt ausgewertet, wobei folgende absolute Zeitdefinition erlaubt ist:

```
<Value>     ::= {<Datevalue>}{ <TimeValue>}
<DateValue> ::= {<day>}.{<month>}.{<year>}
<TimeValue> ::= {<hour>}:{<minute>}{:{<second>}}
```

Abweichend von der Definition der Zeitabstände des ***timers*** ist es also erlaubt, nur ein Datum anzugeben.

Das ***datetime*** verwendet den gregorianischen Kalender. Der Wertebereich geht von *01.01.1601 00:00:00* bis *31.12.9999 23:59:59*. Zweistellige Jahreszahlen werden auf den Zeitraum *1970* bis *2069* abgebildet.

Wenn der Anwender im ***datetime*** einen Wert auswählt, einstellt oder eingibt, dann enthält *.value* diesen Wert im vollständigen Format *dd.MM.yyyy HH:mm:ss* (siehe Attribut *.format*). Bei einer Änderung tritt ein *select*-Ereignis auf.

Nicht angegebene Teile werden in der Anzeige durch die entsprechenden Werte des aktuellen Systemdatums bzw. der aktuellen Systemzeit ergänzt. Zum Beispiel wird für ein fehlendes Jahr das aktuelle Jahr eingesetzt.

Bei einem syntaktisch falschen Wertstring wird das aktuelle Datum bzw. die aktuelle Uhrzeit angezeigt (wie bei *""*).

Beispiele

Das aktuelle Datum sei „25.06.2014“ und die aktuelle Uhrzeit „11:53:07“.

Wertstring	Angezeigter Wert
12.1.	*„12.01.2014 11:53:07“* Jahr und Zeit werden aus den aktuellen Werten ergänzt.
12.1	*„25.06.2014 11:53:07“* Wertstring ist syntaktisch falsch. Aktuelles Datum und aktuelle Zeit werden angezeigt.

Ist der Wertstring zwar syntaktisch korrekt, stellt aber kein gültiges Datum oder keine gültige Zeit dar (zum Beispiel „29.2.2013“ oder „33.33.33“), dann ist undefiniert, was im ***datetime*** anzeigt wird. Wenn das System einen Fehler liefert, wird das aktuelle Datum bzw. die aktuelle Uhrzeit angezeigt. Ansonsten wird angezeigt, was das System aus dem Wertstring macht.

Wenn *.value* = *""* oder fehlerhaft ist und das Attribut *.allowundefined* den Wert *true* hat, dann wird der Wert als unbestimmt (ausgeschaltete Checkbox) angezeigt.

Die Ergänzungen und Korrekturen, die vom Objekt für die Anzeige des Werts durchgeführt werden, ändern den Attributwert nicht. Das heißt, *.value* behält den gesetzten Wert so lange, bis der Anwender im ***datetime*** einen Wert auswählt, einstellt oder eingibt.

Das aktuelle Datum und die aktuelle Uhrzeit werden beim Sichtbarmachen berechnet. Außerdem werden sie berechnet, wenn *.value* im sichtbaren Zustand auf einen neuen Wert gesetzt wird den das Objekt für die Anzeige ergänzt oder korrigiert (zum Beispiel *""*). Auch dabei gilt, dass der Attributwert in der Regel nicht an den angezeigten Wert angepasst wird. Ein wiederholtes Setzen desselben Werts (auch von *""*) stellt deshalb keine Wertänderung dar, sodass in diesem Fall auch keine Aktualisierung des angezeigten Werts erfolgt.

33.3.1.4.13 .weeknumbers

Dieses Attribut bestimmt, ob im aufklappbaren Kalender die Nummern der Kalenderwochen angezeigt werden.

	Bezeichner	Datentyp
Regelsprache	**.weeknumbers**	boolean

	Bezeichner		Datentyp
C	AT_weeknumbers		DT_boolean
COBOL	AT-weeknumbers		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_datetime***		
Zugriff	get, set	Standardwert	*false*
changed-Ereignis	ja	Vererbung	ja

Das Attribut zeigt erst ab WINDOWS VISTA Wirkung.

Wertebereich

false	Im Kalender werden keine Kalenderwochen angezeigt
true	Im Kalender werden Kalenderwochen angezeigt.

Hinweis

Die erste Kalenderwoche eines Jahres ist die erste Woche, bei der mindestens 4 Tage im jeweiligen Jahr liegen.

33.3.2 listview

Das ***listview*** (Klassenname ***dmw_listview***) ist ein erweitertes Listenobjekt, das verschiedene Darstellungsarten unterstützt. Das Objekt ist prinzipiell durch den „Windows Explorer“ bekannt, wo es zur Anzeige der Verzeichnisse des Dateisystems verwendet wird.

Die folgenden Abbildungen zeigen die unterschiedlichen Darstellungsarten, des ***listview***-Objekts:

1. Die Symbolansicht stellt große Symbole zusammen mit einer Beschriftung dar. Als Beschriftung dient die erste Inhaltsspalte.



Abbildung 26: *Symbolansicht mit großen Symbolen des listviews*

2. Die kleine Symbolansicht ist der Symbolansicht sehr ähnlich, einziger Unterschied ist, dass kleine Symbole verwendet werden.



Abbildung 27: *Symbolansicht mit kleinen Symbolen des listviews*

3. Die Listenansicht verwendet auch die kleinen Symbole und die Beschriftung, jedoch sind die Einträge von oben nach unten in einer Liste angeordnet. Bei Bedarf werden weitere Spalten angefügt.



Abbildung 28: *Listenansicht des listviews*

4. In der Detailansicht werden alle verfügbaren Informationen in einer Tabelle angezeigt. Jetzt ähnelt das ***listview***-Objekt einer Tabelle.

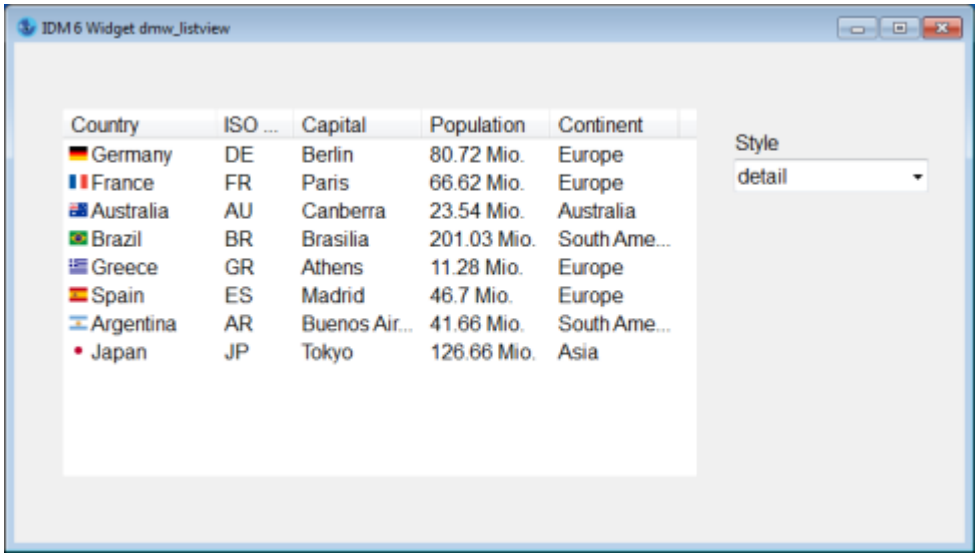


Abbildung 29: *Detailansicht des listviews*

5. Die Kachelansicht entspricht der Listenansicht, zeigt aber große Symbole an. Daneben erscheint die Beschriftung.

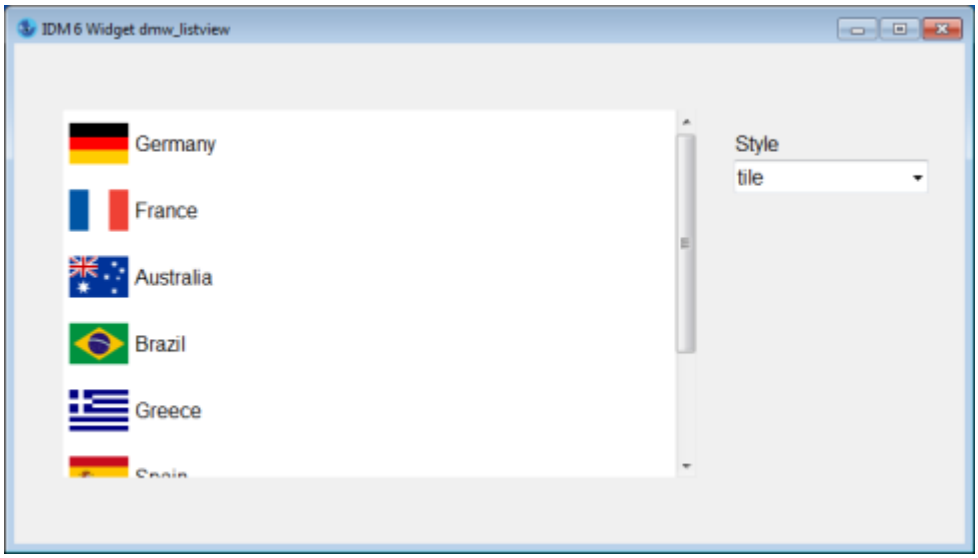


Abbildung 30: *Kachelansicht des listviews*

Verfügbarkeit

» Nur unter MICROSOFT WINDOWS.

» Kann nur mit der USW-Option des ISA Dialog Managers verwendet werden.

» Die **idmwidgets.dll** muss sich im USW-Klassenpfad befinden (Standard: *<IDM-Installationsverzeichnis>\uswclasses*).

33.3.2.1 Definition

```
{ export | reexport } { model } dmw_listview { <Bezeichner> }
{
  <Standardattribute>
  <Geometrieattribute>
  <Hierarchieattribute>
  <Layoutattribute>
  <Textattribute>
  <Objektspezifische Attribute>
}
```

Ereignisse	*activate*	*changed*	*dbselect*
	deactivate	*extevent*	*focus*
	help	*resize*	*select*
Kinder	Keine		

Vater	***groupbox***	***layoutbox***	***notepage***
	splitbox	***toolbar***	***window***
Menü	Popup Menü		

33.3.2.2 Beschreibung der Ereignisse

33.3.2.2.1 activate

Das *activate*-Ereignis tritt auf, wenn ein Listeneintrag markiert wird (*.selected[l] := true*).

Das Attribut *.index* von ***thisevent*** enthält den Index des Eintrags, der markiert wurde. Der Datentyp von *thisevent.index* ist *index* und damit konsistent zum *select*-Ereignis. Der Spaltenwert von *thisevent.index* ist immer *1*, da nur diese Spalte einen Markierungsstatus besitzt.

33.3.2.2.2 dbselect

Das *dbselect*-Ereignis tritt auf, wenn auf dem ***listview*** ein Doppelklick ausgeführt wurde.

Das Attribut *.index* von ***thisevent*** enthält den Index des Eintrags, auf den geklickt wurde. Der Datentyp von *thisevent.index* ist *index* und damit konsistent zum *select*-Ereignis. Wurde kein Eintrag getroffen, dann ist *thisevent.index* nicht gesetzt.

Die Kopfzeile der Detailansicht (Spaltenüberschriften) besitzt kein Doppelklick-Ereignis.

33.3.2.2.3 deactivate

Das *deactivate*-Ereignis tritt auf, wenn ein Listeneintrag seine Markierung verliert (*.selected[l] := false*).

Das Attribut *.index* von ***thisevent*** enthält den Index des Eintrags, der seine Markierung verloren hat. Der Datentyp von *thisevent.index* ist *index* und damit konsistent zum *select*-Ereignis. Der Spaltenwert von *thisevent.index* ist immer *1*, da nur diese Spalte einen Markierungsstatus besitzt.

33.3.2.2.4 resize

Das *resize*-Ereignis tritt auf, wenn in der Detailansicht die Spaltenbreite geändert wurde.

Das Ereignis tritt **nicht** auf für Spalten, deren Breite auf *0* gesetzt wurde und vom ***listview*** berechnet wird.

33.3.2.2.5 select

Das *select*-Ereignis tritt auf, wenn auf dem ***listview*** ein Klick ausgeführt wurde.

Das Attribut *.index* von ***thisevent*** enthält den Index des Eintrags, auf den geklickt wurde. Der Datentyp von *thisevent.index* ist *index* und damit konsistent zum *activate*-Ereignis. Der Spaltenwert kann nur in der Detailansicht ungleich *1* sein.

Wurde kein Eintrag getroffen, dann ist *thisevent.index* nicht gesetzt. Bei einem Klick auf die Kopfzeile der Detailansicht (Spaltenüberschriften) ist der Zeilenwert von *thisevent.index 0*.

33.3.2.3 Attributübersicht

Attribut	Kurzbeschreibung	Datentyp	Standard	Zugriff		C	V
				get	set		
.colcount	Anzahl der Spalten	*integer*	*1*	x	x	x	x
.coltitle[integer]	Spaltenüberschriften	*string*	*""*	x	x	x	x
.colwidth[integer]	Spaltenbreiten in der Detailansicht	*integer*	*0*	x	x	x	x
.content[index]	Listeneinträge, Beschriftung in Spalte 1	*string*	*""*	x	x	x	x
.mincolwidth[integer]	minimale Spaltenbreite in der Detailansicht	*integer*	*0*	x	x	x	x
.picheight	Höhe der großen Symbole	*integer*	*0*	x	x	x	x
.picture[integer]	große Symbole	*object* (*tile*)	*""*	x	x	x	x
.picwidth	Breite der großen Symbole	*integer*	*0*	x	x	x	x
.rowcount	Anzahl der Zeilen	*integer*	*0*	x	x	x	x
.selected[integer]	Markierung der Listeneinträge	*boolean*	*false*	x	x	x	x
.smallpicheight	Höhe der kleinen Symbole	*integer*	*0*	x	x	x	x
.smallpicture[integer]	kleine Symbole	*object* (*tile*)	*""*	x	x	x	x
.smallpicwidth	Breite der kleinen Symbole	*integer*	*0*	x	x	x	x

Attribut	Kurzbeschreibung	Datentyp	Standard	Zugriff		C	V
				get	set		
.style	Darstellungsart	*integer* (*0…4*), *string*	*0*	x	x	x	x

C *changed*-Ereignis bei Änderung

V Attribut wird vererbt

33.3.2.4 Beschreibung der Attribute

33.3.2.4.1 .colcount

Das Attribut definiert die Anzahl der Spalten in der Detailansicht.

	Bezeichner		Datentyp
Regelsprache	**.colcount**		integer
C	AT_colcount		DT_integer
COBOL	AT-colcount		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*1*
changed-Ereignis	ja	Vererbung	ja

Wertebereich

0 Leert das ***listview***-Objekt.

> 0 Anzahl der Spalten in der Detailansicht.

Der Wert von *.colcount* kann implizit erhöht werden, indem *.content* um jeweils eine neue Spalte erweitert wird.

Spalte 1 wird in allen Ansichten angezeigt.

Bei ungültigen Werten wird der Standardwert *1* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.2 .coltitle[integer]

Mit diesem Attribut werden die Spaltenüberschriften in der Detailansicht festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.coltitle[integer]**		string, object (text)
C	AT_coltitle		DT_string, DT_text
COBOL	AT-coltitle		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Index-Datentyp	integer		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Der Wertebereich des Index ist *0 … .colcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .colcount* dient.

Wenn für eine Spalte keine Überschrift definiert wird und auch kein Standardwert *.coltitle[0]* definiert ist, dann wird diese Spalte ohne Spaltenüberschrift angezeigt.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.3 .colwidth[integer]

Dieses Attribut definiert die Breite der Spalten in der Detailansicht.

	Bezeichner	Datentyp
Regelsprache	**.colwidth[integer]**	integer
C	AT_colwidth	DT_integer
COBOL	AT-colwidth	DT-integer
Klassifizierung	Objektspezifisches Attribut	
Objekte	***dmw_listview***	
Index-Datentyp	integer	

	Bezeichner		Datentyp
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich des Index ist *0 … .colcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .colcount* dient.

Wertebereich

0 Die Spaltenbreite wird vom ***listview*** berechnet.
Der gesetzte Wert *0* bleibt erhalten, bis der Anwender die Spaltenbreite interaktiv ändert.

> 0 Breite der jeweiligen Spalte in der Detailansicht.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.4 .content[index]

Das Attribut definiert die Listeneinträge des ***listviews***.

	Bezeichner		Datentyp
Regelsprache	**.content[index]**		string, object (text)
C	AT_content		DT_string, DT_text
COBOL	AT-content		DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Index-Datentyp	index		
Zugriff	get, set	Standardwert	""
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *string* und ***text***-Ressourcen angegeben werden. ***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *string*.

Der Wertebereich des Index ist *[0,1] … [.rowcount,.colcount]*, wobei die Werte mit Index *[0,C]* als Standardwerte für nicht gesetzte Werte im Bereich *[1,C] … [.rowcount,C]* dienen.

Spalte 1 ist die Beschriftung des Listeneintrags, die in allen Ansichten angezeigt wird. Die anderen Spalten werden nur in der Detailansicht angezeigt.

Wenn für *.content[R,C]* kein Inhalt definiert wird und auch kein Standardwert *.content[0,C]* definiert ist, dann wird zwar nichts angezeigt, dennoch ist der Eintrag vorhanden.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.5 .mincolwidth[integer]

Das Attribut legt die minimale Breite der Spalten in der Detailansicht des ***listview***-Objekts fest. Anwender können die Spalte interaktiv nicht schmaler machen, als in diesem Attribut definiert.

	Bezeichner		Datentyp
Regelsprache	**.mincolwidth[integer]**		integer
C	AT_mincolwidth		DT_integer
COBOL	AT-mincolwidth		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Index-Datentyp	integer		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich des Index ist *0 … .colcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .colcount* dient.

Wertebereich

0 Keine minimale Spaltenbreite.
Der Anwender kann die Spalte ausblenden, indem er den rechten Rand des Spaltenkopfs mit gedrückter Maustaste bis zum linken Spaltenrand zieht. Die Spalte kann wieder eingeblendet werden, indem die Maustaste im Spaltenkopf der nachfolgenden Spalte leicht rechts vom linken Spaltenrand gedrückt und die Maus mit gedrückter Taste nach rechts gezogen wird. Man achte auf die in diesem Fall unterschiedliche Form des Mauszeigers direkt über dem linken Spaltenrand und leicht rechts davon.

> 0 Minimale vom Anwender einstellbare Breite der Spalte in der Detailansicht.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.6 .picheight

Mit diesem Attribut wird die Höhe des großen Symbols festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.picheight**		integer
C	AT_picheight		DT_integer
COBOL	AT-picheight		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Wertebereich

0 Die Breite *.picwidth* wird verwendet. Ist diese ebenfalls *0*, dann wird die Größe des Symbols vom System bestimmt.

> 0 Höhe des großen Symbols.

Das große Symbol wird in der Symbolansicht (*.style = "icon"*, *.style = "picture"*) und in der Kachelansicht (*.style = "tile"*) angezeigt.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.7 .picture[integer]

Das Attribut definiert das große Symbol für jeden Listeneintrag.

	Bezeichner	Datentyp
Regelsprache	**.picture[integer]**	object (tile)
C	AT_picture	DT_tile
COBOL	AT-picture	DT-tile
Klassifizierung	Objektspezifisches Attribut	
Objekte	***dmw_listview***	

	Bezeichner		Datentyp
Index-Datentyp	integer		
Zugriff	get, set	Standardwert	*null*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich des Index ist *0 … .rowcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .rowcount* dient.

Wertebereich

null Das angezeigte Symbol ergibt sich wie folgt:

- Wenn *.picture[0]* <> *null* ist, wird das dort definierte Symbol verwendet.
- Wenn *.picture[0]* = *null* ist, dann wird das kleine Symbol vergrößert dargestellt (je nach Vorhandensein *.smallpicture[I]* oder *.smallpicture[0]*).
- Wenn weder ein großes noch ein kleines Symbol vorhanden sind, dann wird der Eintrag ohne Symbol dargestellt.

tile Ressource, die das große Symbol enthält.

Das große Symbol wird in der Symbolansicht (*.style = "icon"*, *.style = "picture"*) und in der Kachelansicht (*.style = "tile"*) angezeigt.

Großes und kleines Symbol gehören zusammen und sollten dieselbe Information darstellen. Beide Symbole können nur gemeinsam referenziert werden. Deshalb sind unterschiedliche Kombinationen zu vermeiden, da jede Kombination Speicherplatz verbraucht.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.8 .picwidth

Mit diesem Attribut wird die Breite des großen Symbols festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.picwidth**		integer
C	AT_picwidth		DT_integer
COBOL	AT-picwidth		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*

	Bezeichner		Datentyp
changed-Ereignis	ja	Vererbung	ja

Wertebereich

0	Die Breite des Symbols wird vom System bestimmt.
> 0	Breite des großen Symbols.

Das große Symbol wird in der Symbolansicht (*.style = "icon"*, *.style = "picture"*) und in der Kachelansicht (*.style = "tile"*) angezeigt.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Wenn *.picheight = 0* ist, bestimmt *.picwidth* auch die **Höhe** des großen Symbols.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.9 .rowcount

Dieses Attribut definiert die Anzahl der Listeneinträge.

	Bezeichner		Datentyp
Regelsprache	**.rowcount**		integer
C	AT_rowcount		DT_integer
COBOL	AT-rowcount		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich von *.rowcount* ist *0 … n* (*n > 0*).

Der Wert von *.rowcount* kann implizit erhöht werden, indem *.content* um jeweils eine neue Zeile erweitert wird.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.10 .selected[integer]

Dieses Attribut legt für jeden Listeneintrag des ***listview***-Objekts fest, ob er markiert ist.

	Bezeichner		Datentyp
Regelsprache	**.selected[integer]**		boolean
C	AT_selected		DT_boolean
COBOL	AT-selected		DT-boolean
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Index-Datentyp	integer		
Zugriff	get, set	Standardwert	*false*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich des Index ist *0 … .rowcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .rowcount* dient.

Wertebereich

false	Der Listeneintrag ist **nicht** markiert.
true	Der Listeneintrag ist markiert.

Das Attribut wird durch Interaktion des Anwenders (*activate*- und *deactivate*-Ereignisse) geändert.

Hinweis

Die Darstellung markierter Einträge hängt vom System ab. Dies gilt insbesondere für die Sichtbarkeit der Markierung, wenn das ***listview*** nicht den Fokus besitzt.

33.3.2.4.11 .smallpicheight

Mit diesem Attribut wird die Höhe eines kleinen Symbols festgelegt.

	Bezeichner	Datentyp
Regelsprache	**.smallpicheight**	integer
C	AT_smallpicheight	DT_integer
COBOL	AT-smallpicheight	DT-integer

	Bezeichner		Datentyp
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

0
: Die Breite *.smallpicwidth* wird verwendet. Ist diese ebenfalls *0*, dann wird die Größe des Symbols vom System bestimmt.

> 0
: Höhe des kleinen Symbols.

Das kleine Symbol wird in der kleinen Symbolansicht (*.style = "smallicon"*, *.style = "smallpicture"*), in der Listenansicht (*.style = "list"*) und in der Detailansicht (*.style = "detail"*, *.style = "report"*) angezeigt.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.12 .smallpicture[integer]

Das Attribut definiert das kleine Symbol für jeden Listeneintrag.

	Bezeichner		Datentyp
Regelsprache	**.smallpicture[integer]**		object (tile)
C	AT_smallpicture		DT_tile
COBOL	AT-smallpicture		DT-tile
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Index-Datentyp	integer		
Zugriff	get, set	Standardwert	*null*
changed-Ereignis	ja	Vererbung	ja

Der Wertebereich des Index ist *0 … .rowcount*, wobei der Wert mit Index *0* als Standardwert für nicht gesetzte Werte im Bereich *1 … .rowcount* dient.

Wertebereich

null Das angezeigte Symbol ergibt sich wie folgt:

- Wenn *.smallpicture[0]* <> *null* ist, wird das dort definierte Symbol verwendet.
- Wenn *.smallpicture[0]* = *null* ist, dann wird das große Symbol verkleinert dargestellt (je nach Vorhandensein *.picture[I]* oder *.picture[0]*).
- Wenn weder ein kleines noch ein großes Symbol vorhanden sind, dann wird der Eintrag ohne Symbol dargestellt.

tile Ressource, die das kleine Symbol enthält.

Das kleine Symbol wird in der kleinen Symbolansicht (*.style = "smallicon"*, *.style = "smallpicture"*), in der Listenansicht (*.style = "list"*) und in der Detailansicht (*.style = "detail"*, *.style = "report"*) angezeigt.

Kleines und großes Symbol gehören zusammen und sollten dieselbe Information darstellen. Beide Symbole können nur gemeinsam referenziert werden. Deshalb sind unterschiedliche Kombinationen zu vermeiden, da jede Kombination Speicherplatz verbraucht.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.13 .smallpicwidth

Mit diesem Attribut wird die Breite eines kleinen Symbols festgelegt.

	Bezeichner		Datentyp
Regelsprache	**.smallpicwidth**		integer
C	AT_smallpicwidth		DT_integer
COBOL	AT-smallpicwidth		DT-integer
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Wertebereich

0 Die Breite des Symbols wird vom System bestimmt.

> 0 Breite des kleinen Symbols.

Das kleine Symbol wird in der kleinen Symbolansicht (*.style = "smallicon"*, *.style = "smallpicture"*), in der Listenansicht (*.style = "list"*) und in der Detailansicht (*.style = "detail"*, *.style = "report"*) angezeigt.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

Wenn *.smallpicheight = 0* ist, bestimmt *.smallpicwidth* auch die **Höhe** des kleinen Symbols.

Das Ändern des Attributs im sichtbaren Zustand kann zu einem Flackern des Objekts führen.

33.3.2.4.14 .style

Dieses Attribut bestimmt die Darstellungsart des ***listviews***.

	Bezeichner		Datentyp
Regelsprache	**.style**		integer, string, object (text)
C	AT_style		DT_integer, DT_string, DT_text
COBOL	AT-style		DT-integer, DT-string, DT-text
Klassifizierung	Objektspezifisches Attribut		
Objekte	***dmw_listview***		
Zugriff	get, set	Standardwert	*0*
changed-Ereignis	ja	Vererbung	ja

Beim Schreibzugriff („set“) können Werte vom Datentyp *integer*, *string* und ***text***-Ressourcen angegeben werden.***text***-Ressourcen werden automatisch zu *string* gewandelt.

Lesezugriff („get“) liefert immer einen Wert vom Datentyp *integer*.

Wertebereich

Wert		Beschreibung
integer	string	
0	*"icon"* *"picture"*	Symbolansicht mit großen Symbolen. Anordnung der Listeneinträge zuerst von links nach rechts, dann von oben nach unten. Das Symbol wird über der Beschriftung des Eintrags angezeigt.
1	*"smallicon"* *"smallpicture"*	Symbolansicht mit kleinen Symbolen. Anordnung der Listeneinträge zuerst von links nach rechts, dann von oben nach unten. Das Symbol wird links von der Beschriftung des Eintrags angezeigt.

Wert		Beschreibung
integer	string	
2	*"list"*	Listenansicht mit kleinen Symbolen. Anordnung der Listeneinträge zuerst von oben nach unten, dann von links nach rechts. Das Symbol wird links von der Beschriftung des Eintrags angezeigt.
3	*"detail"* *"report"*	Liste mit kleinen Symbolen, bei der die Listeneinträge untereinander angeordnet werden. Links neben der Beschriftung wird ein kleines Symbol angezeigt. Zu jedem Listeneintrag werden Detailinformationen in mehreren Spalten angezeigt. Die Darstellung ähnelt einer Tabelle.
4	*"tile"*	Darstellung der Listeneinträge als Kacheln mit großem Symbol links sowie Beschriftung rechts. Anordnung der Listeneinträge untereinander.

Bei ungültigen Werten wird der Standardwert *0* verwendet. Der Attributwert wird allerdings nicht geändert.

33.4 Erweiterung der USW-Schnittstelle

33.4.1 UC_InqName

Der Datentyp *UC_InqName* wurde erweitert:

```
typedef enum {
  UC_I_undef = 0,
  ...
  UC_I_notify,
  UC_I_replace  // ab IDM-Version A.06.01.a
} UC_InqName;
```

33.4.2 UC_API

Die Struktur **UC_API** wurde erweitert:

```
typedef struct {
  DM_UInt1 version;
  ...
```

```
  UC_ControlFunc          control;          // ab IDM-Version A.05.02.g
  UC_GetToolkitDataExFunc getToolkitDataEx  // ab IDM-Version A.06.01.a
} UC_API;
```

33.4.3 UC_InqArg

Die Struktur **UC_InqArg** wurde erweitert:

```
typedef struct {
  DM_InqName name;
  union {
    ...
    UC_Notify  notify;
    UC_Replace replace;  // ab IDM-Version A.06.01.a
  } arg;
} UC_InqArg;
```

33.4.4 UC_Replace

Benachrichtigung, dass die Definition einer verwendeten IDM-Ressource geändert wurde. Es bleibt dem WSI überlassen, wann und für welche Ressourcen ein Aufruf von **UC_Inquire** mit *UC_I_Replace* erfolgt.

```
typedef struct {
  DM_ID dmid;
} UC_Replace;
```

Das Element *dmid* bezeichnet die Ressource welche geändert wurde.

33.4.5 getToolkitDataEx

Funktion zur Abfrage der Toolkit-Daten (z.B. von Ressourcen).

```
typedef FPTR (DML_default DM_EXPORT *UC_GetToolkitDataExFunc)
  __((DM_ID        dmid,
      DM_Attribute attr,
      FPTR         data,
      DM_Options   options));
```

Parameter

DM_ID dmid

Identifikator des Objekts.

DM_Attribute attr

IDM-Attribut, das erfragt werden soll.

FPTR data

Zeiger auf zusätzliche Daten, wenn nicht verwendet muss hier *NULL* stehen.

DM_Options options

IDM-Option, derzeit noch keine verfügbar.

Rückgabewert

Je nach Art des abgefragten Wertes liefert diese Funktion die entsprechenden auf *FPTR* konvertierten Werte (siehe C-Schnittstellenfunktion **DM_GetToolkitDataEx**).

33.4.6 UC_Inquire

Anfrage	Input	Output
`name == UC_I_Replace` Die USW-Klasse kann die Benachrichtigung, dass eine Ressource geändert wurde, bearbeiten.	`arg.replace.dmid` Die IDM-ID der Ressource, die sich geändert hat.	keiner

33.5 Datentypen für Sammlungen

33.5.1 Erweiterung der Datentypen

Folgende neue Sammlungsdatentypen sind nun im IDM erlaubt. Dabei ist zu beachten, dass für die Adressierung wie auch die adressierten Einzelwerte die Beschränkung gilt, dass nur skalare Typen erlaubt sind. Ein Aufbau von mehrdimensionalen Listen ist also nicht erlaubt.

***Tabelle 13:** Datentypen für Sammlungen (collections)*

Datentyp	Bedeutung
list	Dieser Datentyp definiert eine Liste mit beliebigen Werten die im Adressierungsbereich von 0 … 2^{31} liegen. Dabei wird über *0* der Standardwert adressiert, der an alle Listenwerte vererbt wird. Dieser Datentyp ist für die Manipulation von Listen in der Regelsprache vorgesehen und optimiert.
vector	Dieser Datentyp definiert eine Liste, die Werte des gleichen Typs beinhaltet. Einen Standardwert gibt es nicht. Der Adressierungsbereich ist somit von 1 … 2^{31}. Dieser Datentyp ist speziell für die Kommunikation mit Objekt-Attributen vorgesehen und **nicht** für eine schnelle Manipulation optimiert.

Datentyp	Bedeutung
refvec	Dieser Datentyp definiert eine Liste, die nur Werte des Typs *object* beinhaltet. Der Adressierungsbereich geht von 1 … 2^{31} und bietet keinen Standardwert. Jede object-ID wird nur ein einziges Mal aufgenommen, ein *null* wird nicht in die Liste aufgenommen.
matrix	Dieser Datentyp definiert ein zweidimensionales Feld dessen Werte mit dem *index*-Datentyp adressiert werden können. Ist die angegebene Zeile (first-Wert des Index) oder die Spalte (second-Wert des Index) *0*, so handelt es sich um einen Standardwert der in der Adressierungsreihenfolge [0,0] → [Zeile,0] → [0,Spalte] → [Zeile,Spalte] weitervererbt wird
hash	Dieser Datentyp definiert ein assoziatives Feld mit beliebigem skalaren Adressierungsbereich und beliebigen Werten und einem optionalen Standardwert der zurückgegeben wird für einen Zugriff bei dem der Schlüssel nicht enthalten ist.

Analog zu vordefinierten Vektor- und Matrixattributen wie *.content[]*, *.userdata[]* usw. ist eine Erweiterung von Sammlungen direkt hinter ihrem letzten Element möglich und erlaubt.

Allerdings ist die Verwendung dieser Sammlungsdatentypen für benutzerdefinierte Attribute nicht möglich. Benutzerdefinierte und vordefinierte Attribute besitzen keine eindeutige Zuordnung (Attribute existieren an einem Objekt in einer skalaren **und** feldhaften Variante) und erlauben einen unindizierten Zugriff auf den Standardwert.

33.5.2 Erweiterung der Ausdrücke

Liste und Hashes

Sammlungen mit den Datentypen *hash*, *list*, *matrix*, *refvec* oder *vector* können über die folgende Syntax innerhalb des Regelcodes definiert werden. Eine Definition im statischen Teil eines Dialogs oder Moduls ist nicht als Expression möglich, lediglich konstante Werte sind hier erlaubt. Ohne explizite Angabe eines Listen-Datentyps entsteht eine Liste vom Datentyp *list*. Bei Verwendung des Verweisoperators => entsteht automatisch ein *hash* (assoziatives Feld), wobei der Ausdruck vor dem => den Schlüssel bzw. Index definiert, der zweite den Wert.

Achtung

Aus Kompatibilitätsgründen genießt die Index-Expression Vorrang. Will man also eine zweielementige Liste mit zwei Integer-Werten aufbauen, so sollte man den Listendatentyp voranstellen.

Syntax

```
<Liste> = [ <Datentyp> ] '[' [ <Expression> { ',' <Expression> } ] ']'
<Hash-Liste> = [ <Datentyp> ] '[' [ <Expression> '=>' <Expression>
    { ',' <Expression> '=>' <Expression> } ] ']'
```

Beispiele

```
[1, .xleft, 17+4, Wi.Pb]
["ZDF" => 2, itoa(7) => 7]
matrix[ [1,1]=>"Vorname", [1,2]=>"Nachname", [1,3]=>"Telefonnr." ]
```

33.5.3 Listengröße und Standardwerte

Grundsätzlich hat jeder Sammlungsdatentyp eine aktuelle Größe, die sich entsprechend den gesetzten Werten ergibt. Wertelücken in diesem aktuell gültigen Indizierungsbereich gibt es nicht, lediglich ungesetzte Werte (siehe hierzu auch die Beschreibungen zu **itemcount()**, **countof()**).

Um die Konsistenz zu den bisherigen statischen lokalen Variablen herzustellen erlauben die Datentypen *hash*, *list* und *matrix* einen Standardwert.

Dieser wird angezeigt an ungesetzten Werte-Positionen innerhalb des aktuell gültigen Indizierungsbereiches bis zur aktuellen Größe. Bei *hash*-Datentypen wird bei gesetztem Standardwert dieser für alle möglichen Indexierungen zurückgeliefert.

Die Reihenfolge für den Zugriff auf die Standardwerte bei ungesetztem Wert an der Position *<Row>,<Col>* in einer Matrix *M* ist *M[<Row>,<Col>]* → *M[0,<Col>]* → *M[<Row>,0]* → *M[0,0]*.

Die Zugriffsreihenfolge bis zum Standardwert bei ungesetztem Wert an *<Index>* in einer Liste *L* ist *L[<Index>]* → *L[0]*.

33.5.4 Zugriff und Zuweisungen von Sammlungen

Der Zugriff auf die indizierten Werte (Elemente) einer Sammlung erfolgt, wie bisher von lokalen Variablen oder vordefinierten und benutzerdefinierten Attribute gewohnt, über die [ ]-Operation.

Ohne die [ ]-Operation wird der gesamte Wert geholt oder gesetzt.

Achtung

Es ist zu beachten, dass ein Zugriff auf ein benutzerdefiniertes oder vordefiniertes Attribut ohne [ ]-Index den Standardwert bzw. das skalare Attribut holt oder setzt. Möchte man alle Werte eines Attributs holen oder setzen, so sollten die Funktionen **getvector()** oder **setvector()** verwendet werden.

Beispiele

```
variable list Primes := [2,3,5,7,11,13];
variable vector[integer] Numbers;
variable hash Months := [1=>"Jan", 2=>"Feb", 3=>"Mar"];

print Primes[1];
Months[4] := "Apr";
Numbers := Primes;
Primes[0] := -1;
Primes := [17, 19, 23];
```

Das Ausgeben von Sammlungen über **print** oder **sprintf()** ist ebenso möglich. Dabei werden aber keine Standardwerte mit ausgegeben. Bei den Datentypen *hash* und *matrix* erfolgt die Ausgabe inklusive Index.

Besonderheiten *refvec*

Weiterhin ist zu beachten das der Datentyp *refvec* eine besondere Behandlung genießt, da er für die Eindeutigkeit der Werte sorgt und keine *null*-Werte speichert.

Bei einer indizierten Zuweisung einer schon vorhandenen object-ID wird die object-ID an der Position ersetzt und danach für die Eindeutigkeit gesorgt, was damit auch eine Verschiebung der schon vorhandenen object-ID an eine andere Position und eine Verkleinerung der *refvec*-Liste zur Folge haben kann.

Bei einer indizierten Zuweisung einer *null* wird die object-ID aus der *refvec*-Liste gelöscht und die nachfolgenden object-IDs rücken vor, was wiederum eine Verkleinerung der *refvec*-Liste zur Folge hat.

33.5.5 Automatische Konvertierung

In Zuweisungen oder Parameteraufrufen findet bei Sammlungen eine automatische Konvertierung in den verlangten Listentyp statt. Dabei werden alle Werte ohne Index in ihrer „natürlichen“ Reihenfolge kopiert, wenn möglich inklusive der Standardwerte, z.B. *[0]* bei einer *list*-Variablen wird weitergegeben, wenn der Zieltyp ebenso einen Standardwert besitzt.

Ausnahme

Bei der Zuweisung von einem *hash* mit *index*-Datentyp an eine *matrix* wird der Index übernommen. Bei der umgekehrten Zuweisung von einer *matrix* an einen *hash* wird ebenso der Index mit übernommen. Will man dies verhindern, so sollte die eingebaute **values()**-Funktion verwendet werden.

33.5.6 Performanz

Grundsätzlich sollte bedacht werden, dass die Verwendung von Sammlungen (Gesamtwerte) innerhalb von Ausdrücken initial ein Kopieren der Werte erforderlich macht, dann aber bei lesenden Operationen lediglich Referenzen dieser Liste, also ohne weiteres Kopieren, Anwendung finden. So wird die nötige Performanz bei Umgang mit Sammlungen gewährleistet. Die Weitergabe an Applikationsfunktionen oder die Manipulation durch eingebaute Funktionen machen aber eine erneute Kopieraktion, unter Umständen mit einer Codepage-Konvertierung von String-Werten, erforderlich.

Insofern sollte bedacht werden, dass bei komplexen Ausdrücken, welche Sammlungen bearbeiten, durchaus temporär große Datenmengen auflaufen können. Die Kodierung und Aufteilung von Ausdrücken sollte mit Berücksichtigung der maximalen Listengrößen erfolgen.

33.5.7 Lokale und globale Variablen

Die neuen Sammlungsdatentypen sind bei lokalen und bei globalen Variablen nun verfügbar. Eine Initialisierung einer globalen Sammlung ist ebenso mit konstanten Werten möglich.

Bisher konnten lokale, statische Variablen auch ein Feld oder assoziatives Feld sein, welches nur sehr beschränkt initialisiert werden konnte. Sammlungsdatentypen sind nun auch genauso für lokale und auch lokale, statische Variablen erlaubt. Der Initialisierungsausdruck ist dabei nicht mehr eingeschränkt.

Achtung Verhaltensänderung

Die bisherige Schreibweise für Felder und assoziative Felder für statische lokale Variablen ist wie bisher auch erlaubt und wird auf die Datentypen *list* oder *hash* abgebildet. Die statische Initialisierung über `.<Bezeichner>[<Index>] := <Wert>;` nach dem Variablenteil ist aber **nicht mehr möglich** um die Konsistenz zu anderen lokalen Variablen zu gewährleisten! Der Initialwert setzt den gesamten Variablenwert, und nicht wie früher nur den Standardwert!

Beispiel

```
dialog D

variable hash Prices := [ "iMac" => 1300, "Samsung Tab" => 300 ];
variable vector[string] Weekdays := ["Mon","Tue","Wed"];

on dialog start
{
  variable string Stations[integer] := [1=>"ARD", 2=>"ZDF"];
  variable hash Stations2 := [3=>"SWR", 4=>"RTL"];
  static variable list AllStations := join(Stations,Stations2);
  /* Nicht mehr erlaubt:
   * variable string Stations[integer] := "UNBEKANNT";
   * .Stations[1] := "ARD";
   * oder
   * static variable integer AssArray[string] := -1;
   * Stattdessen möglich:
   * static variable integer AssArray[string] := [nothing=>-1];
   */
  exit();
}
```

33.5.8 Vergleichsoperatoren

Für Sammlungen sind nur die Vergleichsoperatoren = und <> erlaubt.

Gleichheit herrscht bei folgenden Bedingungen:

1. Der Datentyp muss gleich sein.
2. Alle enthaltenen Werte bzw. Index/Wert-Paare müssen gleich sein, inklusive der Standardwerte.

3. Die Anzahl der enthaltenen Werte und ihre Positionen müssen gleich sein.

Beim Vergleich zwischen einem String und einer Text-ID erfolgt der Vergleich auf String-Basis.

Bei der Verwendung von Vergleichsoperatoren ist insbesondere auf den Datentyp zu achten.

Beispiel

```
dialog D
window Wi {
  listbox Lb {
    .content[1] "Sat";
    .content[2] "Sun";
  }
}
on dialog start {
  variable list WeekendDays := ["Sat", "Sun"];
  variable hash DayNumber := ["Sat"=>6,"Sun"=>7];
  variable list Days;

  print WeekendDays = getvector(Lb, .content);
  print vector["Sat","Sun"] = getvector(Lb, .content);
  print values(WeekendDays) = values(getvector(Lb, .content));
  print WeekendDays = keys(DayNumber);
  Days := WeekendDays;
  WeekendDays[0] := "??";
  print Days=WeekendDays;
  exit();
}
```

Ausgabe

```
false => Datentyp verschieden (list<>vector)
true
true
true
false => WeekendDays hat einen Standardwert, Days aber nicht
```

33.5.9 Uninitialisierte Variablen

Wie bisher auch verhält sich der IDM beim Zugriff auf uninitialisierte Werte unterschiedlich. Wird auf eine uninitialisierte lokale oder statische Variable zugegriffen, erhält man *nothing*. Beim Zugriff auf eine uninitialisierte globale Variable (entspricht einem Variablen-Objekt) oder ein benutzerdefiniertes Attribut wird der Zugriff mit einem Fehler *„cannot get value“* verweigert.

33.5.10 Neue und geänderte eingebaute Funktionen

33.5.10.1 itemcount()

Diese Funktion liefert die Anzahl von indizierten Werten in einer Sammlung zurück. Ausgenommen von der Anzahl sind der/die Standardwert(e). Zusammen mit **indexat()** und **valueat()** lassen sich somit einfach Schleifen durch beliebige Sammlungen realisieren.

Typischerweise liefern **countof()** und **itemcount()** für Werte des Datentyps *refvec*, *vector* und *list* den gleichen Wert zurück. **itemcount()** bietet aber die generischere Verwendung, wohingegen **countof()** sich für die strukturierte, typangepasste Verwendung besser eignet.

Definition

```
integer itemcount
(
  anyvalue Value input
)
```

Parameter

anyvalue *Value* input
In diesem Parameter wird der Wert angegeben für den die Anzahl ermittelt werden soll.

Rückgabewert

0	Der übergebene Wert ist skalar oder eine leere Sammlung.
1 … 2^{31}	Anzahl der Werte in der Sammlung ohne Standardwerte.

Beispiel

```
dialog D

on dialog start
{
  variable matrix Matrix := [
    [0,0] => "-?-",
    [1,1] => "germany",
    [1,2] => "berlin",
    [2,1] => "france"
    /* [2,2] => inherited from default [0,0] */ ];
  variable integer I;
  variable anyvalue Idx;

  /* print the Matrix values [1,1] [1,2] ... [2,2] */
  for I:=1 to itemcount(Matrix) do
    Idx := indexat(Matrix, I);
```

```
    print sprintf("%s : %s", Idx, Matrix[Idx]);
  endfor
  exit();
}
```

Ausgabe

```
"[1,1] : germany"
"[1,2] : berlin"
"[2,1] : france"
"[2,2] : -?-"
```

Siehe auch

indexat(), **valueat()**, **countof()**, *.itemcount[<A>]*, **:index()**, **DM_ValueCount()**

33.5.10.2 countof()

Diese Funktion liefert die Größe einer Sammlung zurück. Dies ist typischerweise der höchste Indexwert.

Typischerweise liefern **countof()** und **itemcount()** für Werte des Datentyps *refvec*, *vector* und *list* den gleichen Wert zurück. **itemcount()** bietet aber die generischere Verwendung, wohingegen **countof()** sich für die strukturierte, typangepasste Verwendung besser eignet.

```
anyvalue countof
(
  anyvalue Value input
)
```

Parameter

anyvalue *Value* input
: In diesem Parameter wird der Wert angegeben für den der Indizierungstyp, bzw. der höchste Indexwert, ermittelt werden soll.

Rückgabewert

nothing	Der übergebene Wert ist skalar.
1 … 2^{31}	Höchster Index der übergebenen Liste (Datentypen *list*, *vector*, *refvec*).
[0 … 65535,0 … 65535]	Höchster Index der übergebenen Matrix.
anyvalue	Datentyp des Index des übergebenen assoziativen Felds (Datentyp *hash*).

Beispiel

```
dialog D

on dialog start
{
  variable matrix Matrix := [
    [0,0] => "-?-",
    [1,1] => "germany",
    [1,2] => "berlin",
    [2,1] => "france"
    /* [2,2] => inherited from default [0,0] */ ];
  variable integer Row, Col;
  variable anyvalue Count, Idx;

  /* print the Matrix values [0,0] [0,1] ... [2,2] */
  Count := countof(Matrix);
  for Row:=0 to first(Count) do
    for Col:=0 to second(Count) do
      Idx := [Row,Col];
      print sprintf("%s : %s", Idx, Matrix[Idx]);
    endfor
  endfor
  exit();
}
```

Siehe auch

valueat(), **itemcount()**, *.count[<A>]*, **:index()**, **DM_ValueCount()**

33.5.10.3 indexat()

Diese Funktion liefert den Index einer Sammlung an einer bestimmten Position. Die erlaubten Positionen gehen von *1 … itemcount()* und ermöglichen somit einen Schleifendurchgang durch alle indizierten Werte.

Für Werte des Typs *list*, *refvec* und *vector* ist die Position mit dem eigentlichen Index identisch. Für einen *hash*-Wert gibt es keine definierte Reihenfolge der gelieferten Indizes. Für *matrix*-Werte erfolgt die Abbildung der aufsteigenden Positionen auf eine Ordnung, bei der alle (Spalten-)Werte einer Zeile aufsteigend hintereinander folgen.

Die Funktion gibt einen Fehler (`fail`) zurück falls die Position außerhalb des erlaubten Bereichs liegt oder es sich beim *Value*-Parameter nicht um eine Sammlung handelt.

Definition

```
anyvalue indexat
(
```

```
  anyvalue Value input,
  integer  Pos   input
)
```

Parameter

anyvalue *Value* input
In diesem Parameter wird die Werteliste angegeben, von welcher der Index erfragt werden soll.

integer *Pos* input
Position, für welche der Indexwert ermittelt werden soll.

Rückgabewert

Indexwert an der als Parameter übergebenen Position.

Beispiel

```
dialog D

on dialog start
{
  variable matrix Matrix := [
    [0,0] => "-?-",
    [1,1] => "germany",
    [1,2] => "berlin",
    [2,1] => "france" ];
  variable integer Pos;

  /* print the Matrix elements [1,1] [1,2] ... [2,2] */
  for Pos:=1 to itemcount(Matrix) do
    print sprintf("%s : %s", indexat(Matrix, Pos), valueat(Matrix, Pos));
  endfor
  exit();
}
```

Siehe auch

itemcount(), **valueat()**, **keys()**, **:index()**, **DM_ValueIndex()**

33.5.10.4 valueat()

Diese Funktion liefert den indizierten Wert einer Sammlung an der angegebenen Position zurück. Die erlaubten Positionen gehen von *1 … itemcount()* und ermöglichen somit einen Schleifendurchgang durch alle indizierten Werte ohne den tatsächlichen Index zu kennen.

Die Funktion gibt einen Fehler (`fail`) zurück falls die Position außerhalb des erlaubten Bereichs liegt oder es sich beim *Value*-Parameter nicht um eine Sammlung handelt.

Definition

```
anyvalue valueat
(
  anyvalue Value input,
  integer  Pos   input
)
```

Parameter

anyvalue *Value* input

In diesem Parameter wird die Werteliste angegeben, von welcher der indizierte Wert ermittelt werden soll.

integer *Pos* input

Position, für welche der indizierte Wert ermittelt werden soll.

Rückgabewert

Wert, der sich in der Sammlung an der als Parameter übergebenen Position befindet.

Beispiel

```
dialog D

on dialog start
{
  variable matrix Matrix := [
    [0,0] => "-?-",
    [1,1] => "germany",
    [1,2] => "berlin",
    [2,1] => "france" ];
  variable integer Pos;

  /* print the Matrix elements [1,1] [1,2] ... [2,2] */
  for Pos:=1 to itemcount(Matrix) do
    print sprintf("%s : %s", indexat(Matrix, Pos), valueat(Matrix, Pos));
  endfor
  exit();
}
```

Siehe auch

indexat(), **values()**, **:index()**, **DM_ValueGet()**, **DM_ValueIndex()**

33.5.10.5 keys()

Diese Funktion liefert eine Liste aller Indexwerte einer Sammlung zurück. Die Indizes von Standardwerten fließen dabei nicht in die Liste ein. Wird als Parameter ein Skalar angegeben, so erhält

man eine leere Liste.

Definition

```
list keys
(
  anyvalue Value input
)
```

Parameter

anyvalue *Value* input
In diesem Parameter wird der Wert angegeben auf den die Funktion angewendet wird.

Rückgabewert

Liste mit allen Indizes einer Sammlung.

Beispiel

```
dialog D

on dialog start
{
  variable matrix Matrix := [
    [0,0] => "-?-",
    [1,1] => "germany",
    [1,2] => "berlin",
    [2,1] => "france" ];
  variable anyvalue Idx;

  /* print the Matrix elements [1,1] [1,2] ... [2,2] */
  foreach Idx in keys(Matrix) do
    print sprintf("%s : %s", Idx, Matrix[Idx]);
  endfor
  exit();
}
```

Siehe auch

indexat(), **itemcount()**, **countof()**, **values()**, `foreach`

33.5.10.6 values()

Diese Funktion liefert eine Liste aller Werte einer Sammlung. Standardwerte fließen dabei nicht ein. Auf einen skalaren Wert angewandt wird dieser als Liste zurückgeliefert.

Definition

```
anyvalue values
(
  anyvalue Value input
)
```

Parameter

anyvalue *Value* input
In diesem Parameter wird der Wert angegeben auf den die Funktion angewendet wird.

Rückgabewert

Liste mit allen Werten einer Sammlung ohne die Standardwerte.

Bei einem Skalar als Parameter wird eine Liste zurückgegeben, die nur diesen Parameter enthält.

Beispiel

```
dialog D

on dialog start
{
  variable hash DomainHash := [
    ".de" => "germany",
    ".us" => "usa",
    ".fr" => "france",
    ".uk" => "united kingdom" ];
  variable string Country;

  /* print all country names from the DomainHash */
  foreach Country in values(DomainHash) do
    print Country;
  endfor
  exit();
}
```

Ausgabe

```
"germany"
"france"
"united kingdom"
"usa"
```

Siehe auch

valueat(), **:index()**, **DM_ValueCount()**

33.5.10.7 join()

Diese Funktion bildet aus den angegebenen Parametern eine neue Werteliste. Der Typ der Werteliste ergibt sich dabei aus dem ersten Aufrufparameter. Ist dieser eine Sammlung oder ein Sammlungsdatentyp (*list*, *vector*, *refvec*, *hash*, *matrix*), so hat die Resultatswerteliste den gleichen Typ. Ein Datentyp als erster Parameterwert fließt grundsätzlich nicht in die Resultatsliste ein. Handelt es sich bei den angegebenen Parametern um Sammlungen, so werden hiervon nur die Werte (ohne Standardwerte) herangezogen. Skalare Werte werden zur Resultatswerteliste hinzugenommen.

Die Reihenfolge der Werte in der Resultatswerteliste entspricht der Reihenfolge der Parameter und bei Sammlungen deren „normaler“ Reihenfolge (siehe hierzu **indexat()**).

Die Indexierung erfolgt normalerweise über den erhöhten letzten Indexwert oder die Werteanzahl (siehe **countof()** und **itemcount()**). Das heißt, die Werte in der Resultatsliste erhalten normalerweise die Indexierung über *1, 2, 3, …* oder *[1,1], [1,2], [1,3], ….*

Handelt es sich aber beim Typ der Resultatsliste wie auch beim Parameterwert um einen *hash*, so wird der Index der Hash-Werte übernommen. Hinzugefügte skalare Parameter erhalten als Index den erhöhten **itemcount()**-Wert der Resultatsliste.

Zu beachten ist bei der Verwendung von *refvec* als Resultatslistentyp, dass es zu einer Rückgabe eines Fehlers (`fail`) kommt, falls ein Wert nicht vom Datentyp *object* ist.

Definition

```
anyvalue join
(
      anyvalue Value input
  { , anyvalue Par2  input }
  ...
  { , anyvalue Par16 input }
)
```

Parameter

anyvalue *Value* input
: In diesem Parameter wird der Wert angegeben auf den die Funktion angewendet wird. Dieser Parameter wird auch genutzt um den Typ für die Resultatsliste zu bestimmen.

*anyvalue **Par2** input*

…

*anyvalue **Par16** input*
: Zusätzliche optionale Parameter auf welche die Funktion angewendet wird.

Rückgabewert

Sammlung, die alle Werte aus den übergebenen Parametern enthält. Wenn der erste Parameter ein Sammlungsdatentyp oder eine Sammlung ist, hat die zurückgegebene Sammlung diesen Datentyp. Ein Datentyp als erster Parameter wird nicht in die zurückgegebene Sammlung übernommen.

Änderung ab IDM-Version A.06.01.b bei der Übergabe des Datentyps *string* als ersten Parameter

Wird der Funktion **join()** als erstes Argument der Datentyp *string* übergeben, dann werden die nachfolgenden Argumente in den Datentyp *string* konvertiert und anschließend verkettet. Sind die nachfolgenden Argumente Sammlungen, werden zunächst deren Werte in der natürlichen Reihenfolge des Index zu einem String verkettet.

Falls das erste Argument ein Datentyp, aber weder *string* noch ein Sammlungsdatentyp ist, gibt **join()** einen Fehler zurück.

Beispiel

```
dialog D

on dialog start
{
  variable hash DomainHash := [
    ".de" =>" germany",
    ".us" => "usa",
    ".fr" => "france",
    ".uk" => "united kingdom" ];
  variable list Countries;
  variable hash Domains;

  /* join the values to a list */
  Countries := join(list, DomainHash, "norway", "spain");
  /* join the values and keys into a hash */
  Domains := join(DomainHash, [".us" => "united states of america"]);

  print Countries;
  print Domains;

  exit();
}
```

Ausgabe

```
["norway","spain","germany","usa","france","united kingdom"]
[".de"=>"germany",".us"=>"united states of america",".fr"=>"france",
".uk"=>"united kingdom"]
```

Siehe auch

keys(), **DM_ValueChange()**

33.5.10.8 find()

Diese Funktion sucht einen angegebenen Wert in einer Werteliste und gibt den ersten gefundenen Index zurück an der dieser Suchwert zu finden ist.

Die Suche schließt dabei niemals das Standardelement (also z.B. Index *[0]* oder *[0,0]*) ein.

Durch Angabe eines Start- und Endindex kann die Suche beschränkt werden. Die gültigen Werte sind:

- Kein Wert angegeben.
- *integer*-Wert im Bereich von *1 … Feldgröße* (muss aber > *0* sein) für Sammlungen mit den Datentypen *vector*, *refvec*, *list* oder *hash*.
- *index*-Wert im Bereich von *[1,1]* bis *[rowcount,colcount]* wobei auch hier *rowcount* und *colcount* > *0* sein müssen.
- Gültiger Index in einem *hash*.

Bei der Suche nach Strings kann auch „case-insensitive“ oder nach dem ersten Vorkommen als Präfix gesucht werden.

Da die Indizes von Hashes keiner Ordnung unterworfen sind, sollte als Index nur ein *integer*-Wert im Bereich *1 … itemcount()* angegeben werden, der die Position des Start- bzw. End-Elements angibt.

Bei zweidimensionalen Feldern (analog zur **:find()**-Methode für *.content[]* am ***tablefield***) bewirkt die Angabe eines Indexbereichs *[Sy,Sx]* bis *[Ey,Ex]* die Einschränkung des Suchbereichs auf den Bereich *[min(Sy,Ey),min(Sx,Ex)] … [max(Sy,Ey),max(Sx,Ex)]*. Somit ist die Einschränkung auf Zeilen und Spalten problemlos möglich.

Definition

```
anyvalue find
(
      anyvalue ListValue  input,
      anyvalue Value      input
  { , anyvalue StartIndex input
  { , anyvalue EndIndex   input } }
  { , boolean  CaseSensitive := false       input }
  { , enum     MatchType     := match_exact input }
)
```

Parameter

anyvalue *ListValue* input

In diesem Parameter wird ein Listenwert erwartet in dem gesucht wird.

anyvalue *Value* input

Nach dem in diesem Parameter angegebenen Wert wird gesucht. Grundsätzlich werden nur Werte verglichen, deren Typen „gleich“ bzw. überführbar sind (ein ***text*** wird hierzu in einen *string* überführt).

anyvalue ***StartIndex*** *input*

In diesem optionalen Parameter wird der Startindex angegeben, ab dem in der Sammlung nach einem Wert gesucht werden soll. Wird hier kein Wert angegeben, wird *1* bei eindimensionalen bzw. *[1,1]* bei zweidimensionalen Sammlungen angenommen.

anyvalue ***EndIndex*** *input*

In diesem optionalen Parameter wird der Endindex angegeben, bis zu dem in der Sammlung nach einem Wert gesucht werden soll. Die Angabe ist nur zulässig, wenn auch ein Startindex angegeben wurde.

boolean ***CaseSensitive*** *:= false input*

Dieser optionale Parameter findet nur bei *string*-Werten Beachtung und definiert, ob die Suche unter Beachtung der Groß-/Kleinschreibung erfolgen soll oder nicht. Der Standardwert ist *false*.

enum ***MatchType*** *:= match_exact input*

Dieser optionale Parameter findet nur bei *string*-Werten Beachtung und definiert wie nach einem String gesucht werden soll. Die zulässigen Werte sind:

MatchType	Bedeutung
match_begin	Findet den ersten String in der Sammlung, der mit dem Suchstring beginnt.
match_first	Findet den String in der Sammlung, dessen Anfang die größte Übereinstimmung mit dem Suchstring aufweist. Enthält die Sammlung einen exakt mit dem Suchstring übereinstimmenden String, wird dessen Index zurückgegeben.
match_exact (Standardwert)	Findet den ersten String in der Sammlung, der mit dem gesuchten String exakt übereinstimmt.
match_substr	Findet den ersten String in der Sammlung, der den gesuchten String enthält.

Rückgabewert

0	Element wurde in der übergebenen Liste (Datentypen *list*, *vector*, *refvec*) nicht gefunden.
[0,0]	Element wurde in der übergebenen Matrix nicht gefunden.
nothing	Element wurde in dem übergebenen assoziativen Feld (Datentyp *hash*) nicht gefunden.
sonst	Index des Elements in der übergebenen Sammlung. Der zurückgegebene Wert hat den Index-Datentyp der übergebenen Sammlung.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl, wenn der *ListValue*-Parameter keine Sammlung ist, die Indexbereiche außerhalb des gültigen Bereichs liegen oder ein anderer ungültiger Parameterwert vorliegt.

Anmerkung

Wenn beim Durchsuchen von Hashes die Parameter *StartIndex* und *EndIndex* einen anderen Datentyp als *integer* haben, wird die Position der Start- und End-Elemente aus den angegebenen Indizes berechnet. Hierin unterscheidet sich das Durchsuchen von Hashes mit der Funktion **find()** vom Durchsuchen eines assoziativen Arrays mit der Methode **:find()**.

Beispiel

```
dialog D

on dialog start
{
  variable hash Hash := [
    "030"  => "berlin",
    "0711" => "stuttgart",
    "089"  => "munich" ];
  variable matrix Matrix := [
    [1,1] => "area code", [1,2] => "call number",
    [2,1] => 089,         [2,2] => 713400,
    [3,1] => 0711,        [3,2] => 805439 ];
  variable anyvalue Idx;

  Idx := find(Hash, "stuttgart");
  if Idx <> nothing then
    print "Found: stuttgart => " + Idx;
    Idx := find(Matrix, atoi(Idx), [2,1], [3,1]);
    if Idx <> [0,0] then
      print "Call: " + Matrix[first(Idx),2];
    endif
  endif
  exit();
}
```

Siehe auch

sort(), **:find()**

33.5.10.9 sort()

Diese Funktion liefert eine sortierte Kopie des angegebenen Listenwertes zurück. Es erfolgt nur eine Sortierung der Werte, nicht der Indizes. Insofern ist auch eine Wertesortierung bei assoziativen Feldern (*hash*-Datentyp) möglich.

Die Sortierung erfolgt gruppiert in den Datentypen der Werteelemente, aufsteigend nach dem Ordnungswert des Datentyps und des Wertes. Bei der Sortierung von Texten bzw. Strings erfolgt immer ein Vergleich im für eine Sortierung besser geeigneten Datentyp *string*.

Die Sortierung kann wie folgt gesteuert werden:

» Mit dem *Reverse*-Parameter kann die Sortierreihenfolge umgedreht werden.

» Über den *SortType*-Parameter kann die Sortierordnung von Strings und Texten beeinflusst werden

Definition

```
anyvalue sort
(
      anyvalue ListValue input
  { , boolean  Reverse  := false       input }
  { , enum     SortType := sort_binary input }
)
```

Parameter

anyvalue *ListValue* input

In diesem Parameter wird der Listenwert angegeben, der sortiert werden soll.

*boolean **Reverse** := false input*

Steht dieser optionale Parameter auf *true*, erfolgt die Sortierung in absteigender Reihenfolge. Der Standardwert ist *false*, was einer aufsteigenden Sortierung entspricht.

*enum **SortType** := sort_binary input*

Dieser optionale Parameter findet nur Anwendung bei Elementwerten, die vom Datentyp *string* sind bzw. zur Sortierung temporär in einen String gewandelt werden. Die zulässigen Werte sind:

SortType	Bedeutung
sort_binary (Standardwert)	Sortierung entsprechend dem Unicode-Zeichencode.
sort_linguistic	Sortierung entsprechend der Sprach- und Regionseinstellungen (locale) des Systems. Abhängig von den Systemeinstellungen und der eingestellten Codepage werden Groß- und Kleinschreibung sowie Umlaute berücksichtigt. Für Unix bzw. Linux empfehlen wir die Verwendung des UTF8-Locales. Da bei dieser Sortierung Codepage-Konvertierungen notwendig sind, ist sie langsamer als *sort_binary*. **Siehe auch** Dokumentation von locale, LC_CTYPE, LC_COLLATE, strcoll ihres Betriebssystems.

Rückgabewert

Sortierte Sammlung mit demselben Datentyp wie die übergebene Sammlung.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl, wenn der *ListValue*-Parameter keine Sammlung ist oder ein anderer Parameter einen unerlaubten Wert aufweist.

Beispiel

```
// UTF8
dialog D

on dialog start
{
  variable list List := [ "Äcker", "Bande", "Bäcker", "binden",
                          "Bund", "Aachen", "an" ];
  variable string S;

  foreach S in sort(List, sort_linguistic) do
    print S;
  endfor
  exit();
}

/* liefert: Aachen Äcker an Bäcker Bande binden Bund */
```

Siehe auch

find(), **keys()**, **values()**

33.5.10.10 min()

Diese Funktion liefert den kleinsten Integer-Wert, der in allen seinen Aufrufparametern vorkommt.

Die Aufrufparameter können dabei skalare Integer-Werte oder Sammlungen (*vector*, *list*, *matrix*, *hash*) sein. Für letztere werden die indizierten Elemente (ohne die Standardwerte) für die Bestimmung des Minimalwerts herangezogen.

Definition

```
integer min
(
      anyvalue Par1  input
  { , anyvalue Par2  input }
  ...
  { , anyvalue Par16 input }
)
```

Parameter

anyvalue *Par1* input

*anyvalue **Par2** input*

…

*anyvalue **Par16** input*

In diesen Parametern werden die Werte angegeben, auf welche die Funktion angewendet wird.

Rückgabewert

Der kleinste vorkommende Integer-Wert wird geliefert.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl, wenn in den Argumenten kein skalarer Integer-Wert vorkommt oder Argumente bzw. deren Elemente einen Wert beinhalten, der kein Integer-Wert ist.

Beispiel

```
dialog D

on dialog start
{
  variable list List := [17, 3, 23, 5];

  print min(List, 24, 4);  /* print 3 */
  exit();
}
```

Siehe auch

max()

33.5.10.11 max()

Diese Funktion liefert den größten Integer-Wert, der in allen seinen Aufrufparametern vorkommt.

Die Aufrufparameter können dabei skalare Integer-Werte oder Sammlungen (*vector*, *list*, *matrix*, *hash*) sein. Für letztere werden die indizierten Elemente (ohne die Standardwerte) für die Bestimmung des Maximalwerts herangezogen.

Definition

```
integer max
(
      anyvalue Par1  input
  { , anyvalue Par2  input }
  ...
```

```
  { , anyvalue Par16 input }
)
```

Parameter

anyvalue *Par1* input

*anyvalue **Par2** input*

…

*anyvalue **Par16** input*

In diesen Parametern werden die Werte angegeben, auf welche die Funktion angewendet wird.

Rückgabewert

Der größte vorkommende Integer-Wert wird geliefert.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl, wenn in den Argumenten kein skalarer Integer-Wert vorkommt oder Argumente bzw. deren Elemente einen Wert beinhalten, der kein Integer-Wert ist.

Beispiel

```
dialog D

on dialog start
{
  variable list List := [17, 3, 23, 5];

  print max(List, 24, 4);  /* print 24 */
  exit();
}
```

Siehe auch

min()

33.5.10.12 getvector()

Diese Funktion kann angewendet werden um von Vektor- oder Matrixattributen (benutzerdefinierten wie auch vordefinierten) den Gesamtwert, das heißt eine Auflistung der indizierten Werte, oder einen Ausschnitt daraus, in einem *vector* zu erhalten.

Definition

```
vector getvector
(
      object    Object     input,
```

```
      attribute Attribute  input
  { , anyvalue  FirstIndex input
  { , anyvalue  LastIndex  input } }
)
```

Parameter

object *Object* input
Dieser Parameter definiert das Objekt dessen Attributwerte erfragt werden soll.

attribute *Attribute* input
Dieser Parameter definiert das Attribut welches erfragt werden soll.

anyvalue ***FirstIndex*** *input*
Dieser optionale Parameter definiert den Startindex ab dem die Elementwerte erfragt werden. Für eindimensionale Feldattribute sollte hier ein *integer*-Wert angegeben werden, für zweidimensionale Felder ein *index*-Wert. Der Standardwert für eindimensionale Feldattribute ist dabei der *integer*-Wert *1*, für zweidimensionale Felder der *index*-Wert *[1,1]*.

anyvalue ***LastIndex*** *input*
In diesem optionalen Parameter wird der Endindex angegeben bis zu dem einschließlich die Werte aus dem Attribut in die Resultatswerteliste übernommen werden. Auch hier wird für eindimensionale Feldattribute ein *integer*-Wert erwartet, für zweidimensionale Felder ein *index*-Wert. Wird kein *LastIndex*-Parameter angegeben, so werden alle Werte bis zum Ende des Feldes aufgenommen. Der *LastIndex*-Wert sollte hinter dem *FirstIndex*-Wert liegen.

Rückgabewert

Die vom Objektattribut ermittelten indizierten Werte werden in Form einer Werteliste vom Typ *vector* zurückgegeben.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl bei einem ungültigen Objekt bzw. Attribut oder bei einem ungültigen Indexbereich.

Beispiel

```
dialog D
{
  string Extra[integer];
  .Extra[1] := "Anna";
  .Extra[2] := "Wolfgang";
}

window Wi
{
  .title "Names";
```

```
  listbox Lb
  {
     .xauto 0;  .yauto 0;
     .content[1] "..NAMES..";
     .content[2] "Ines";
     .content[3] "Dieter";
     .content[4] "Hugo";
  }

  on close { exit(); }
}

on dialog start {
  variable vector Names;

  Names := join(getvector(Lb, .content, 2), getvector(D,.Extra));
  setvector(Lb, .content, sort(Names), 2);
}
```

Siehe auch

setvector(), **DM_GetVectorValue()**, **DM_SetVectorValue()**

33.5.10.13 setvector()

Diese Funktion kann zur Modifikation von vektoriellen Attributen (vordefiniert oder benutzerdefiniert) verwendet werden. Dabei kann das Attribut als Ganzes oder auch nur ein zusammenhängender Teilbereich mit neuen Werten belegt werden.

Definition

```
boolean setvector
(
      object    Object     input,
      attribute Attribute  input,
      vector    NewValue   input
  { , anyvalue  FirstIndex input
  { , anyvalue  LastIndex  input } }
  { , boolean   SendEvent := true input }
)
```

Parameter

object *Object* input

Dieser Parameter definiert das Objekt dessen Attributwerte geändert werden sollen.

attribute *Attribute* input

Dieser Parameter definiert das Attribut, das gesetzt werden soll.

vector *Value* input

In diesem Parameter wird die Werteliste angegeben, die dem vektoriellen Attribut gesetzt wird.

*anyvalue **FirstIndex** input*

Dieser optionale Parameter definiert den Startindex ab dem die Attributwerte modifiziert werden. Für eindimensionale Feldattribute sollte hier ein *integer*-Wert angegeben werden, für zweidimensionale Felder ein *index*-Wert. Der Standardwert für eindimensionale Feldattribute ist dabei der *integer*-Wert *1*, für zweidimensionale Felder der *index*-Wert *[1,1]*.

*anyvalue **LastIndex** input*

In diesem optionalen Parameter wird der Endindex angegeben bis zu dem einschließlich die Werte im Attribut durch die Werte aus der Werteliste modifiziert werden sollen. Auch hier wird für eindimensionale Feldattribute ein *integer*-Wert erwartet, für zweidimensionale Felder ein *index*-Wert. Der *LastIndex*-Wert sollte hinter dem *FirstIndex*-Wert liegen.
Wird kein *LastIndex*-Parameter angegeben (Standardwert *nothing*), so definiert die Größe des *Value*-Vektors die Anzahl bis zu der das vektorielle Attribute modifiziert wird. Die dahinterliegenden Elementwerte des Attributs werden dann abgeschnitten.

*boolean **SendEvent** := true input*

Über diesen optionalen Parameter wird gesteuert, ob das Setzen des Attributs ein *attribute-changed*-Ereignis auslösen soll. Der Standardwert ist *true*, was eine Verschickung des *changed*-Ereignisses für das Attribut bewirkt. Bei einem Parameterwert von *false* wird kein Ereignis verschickt.

Rückgabewert

true	Attribut wurde geändert.
false	Änderung des Attributs konnte nicht ausgeführt werden.

Fehlerverhalten

Der Aufruf der Funktion schlägt fehl bei einem ungültigen Objekt bzw. Attribut, bei einem ungültigen Indexbereich oder wenn das Setzen z.B. wegen einer Diskrepanz zwischen Attributtyp und Wertetyp nicht möglich ist.

Beispiel

```
dialog D

window Wi
{
  .title "Cities";
  .width 300;

  tablefield TfCities
  {
```

```
    .xauto 0;  .yauto 0;
    .rowheight[0] 25;  .colwidth[0] 80;
    .rowcount 1;  .colcount 2;
    .rowheader 1;
    .direction 2;
  }

  on close { exit(); }
}

on dialog start
{
  variable hash Capitals := [
    "germany" => "berlin",
    "france"  => "paris",
    "england" => "london" ];

  // fill the title row
  setvector(TfCities, .content, ["Country", "City"]);
  // fill the country and city column
  setvector(TfCities, .field, keys(Capitals), [1,1],
            [itemcount(Capitals),1]);
  setvector(TfCities, .field, values(Capitals), [1,2]);
}
```

Siehe auch

DM_SetVectorValue()

33.5.11 Schleifenkonstrukt „foreach“

Dieses Konstrukt dient zur Formulierung von Schleifen, die über alle Elemente einer Sammlung gehen.

Syntax

```
foreach <Counter> in <Expression> do
  <Statements>
endfor
```

Die Schleife bricht mit einem Fehler ab, wenn der Ausdruck (*<Expression>*) keine Sammlung ist oder der Schleifenwert nicht dem *<Counter>* zugewiesen werden kann. Die *<Expression>* wird nur einmal Initial ausgewertet. Die Laufvariable *<Counter>* darf innerhalb der *<Statements>* verändert werden. Eine Änderung der Laufvariablen hat keinen Einfluss auf die Durchlaufzahl oder den Wert der Laufvariablen im nächsten Durchlauf.

Beispiel

```
dialog D
window Wi
{
  listbox Lb
  {
    .xauto 0;  .yauto 0;
  }
  on close { exit(); }
}

on dialog start
{
  variable list StationList:= ["ARD", "ZDF", "SWR3", ""];
  variable string Station;

  foreach Station in StationList do
    Lb.content[Lb.itemcount+1] := Station;
  endfor
}
```

33.5.12 C-Schnittstellenerweiterung

Sammlungen können nun auch als Argumente oder Rückgabewerte von C-Funktionen (gilt nicht für COBOL) oder für Datenmodell-Callbacks genutzt werden.

Neu ist ebenso, dass als Rückgabetyp für C-Funktionen auch der Datentyp *index* (*DT_index*) erlaubt ist.

33.5.12.1 Ergänzung der DM_Value-Struktur um Sammlungsdatentypen

Name	Belegtes Element	Bedeutung
DT_list	*pointer*	Werteliste
DT_hash	*pointer*	Assoziatives Feld
DT_vector	*pointer*	Feld mit gleichem Wertetyp
DT_matrix	*pointer*	Zweidimensionales Feld
DT_refvec	*pointer*	Referenzliste mit DM_ID-Werten, bei der dieselbe ID nur einmal vorkommen kann

Grundsätzlich dürfen **DM_Value**-Strukturen, die Sammlungen beinhalten können bzw. sollen, **nur** über spezielle DM-Funktionen manipuliert und erfragt werden, da der IDM hierfür die Speicherallokierung und Verwaltung von Strings durchführt.

Siehe auch

DM_ValueInit(), **DM_ValueChange()**, **DM_ValueGet()**, **DM_ValueCount()**, **DM_ValueIndex()** und **DM_ValueReturn()**

Um den Umgang mit der **DM_Value**-Struktur sowie mit Strings (*DM_String*) in C-Funktionen zu erleichtern wurde das folgende Konzept eingeführt.

Die C-Schnittstelle des IDM kann Wertereferenzen (*DM_Value** und *DM_String**) nach ihrer Verwendung unterscheiden bzw. für eine spezielle Verwendung initialisieren:

1. Wertereferenz ist ein Argument des Funktionsaufrufs und noch ungemanagt.
2. Wertereferenz ist ein statischer bzw. globaler Wert und wird durch den IDM gemanagt.
3. Wertereferenz ist ein lokaler Wert der vom IDM gemanagt wird. Die Rückgabe an den IDM ist erlaubt. Nach Beendigung des Funktionsaufrufs wird er freigeben.
4. Wertereferenz verweist auf einen vom IDM gemanagten Wert der ungültig ist, zum Beispiel weil er fehlerhaft durch den Anwender manipuliert wurde.
5. Wertereferenz verweist auf ungemanagten Wert (quasi wie bisher).

Dieses Konzept wird von den Funktionen der Art **DM_Value*()** und **DM_*Return()** unterstützt. Es ermöglicht die Manipulation von Werten (z.B. Änderung eines Strings, Hinzufügen von Werten zu einer Liste bzw. einem Hash), den Zugriff auf Elemente bzw. Indizes von Wertelisten sowie die Rückgabe von Wertelisten oder Strings ohne dass man sich explizit um die Verwaltung und Allokierung kümmern muss.

Regeln für die Verwendung

Für Werte und **DM_Value**-Strukturen der Art 1–3 muss Folgendes berücksichtigt werden:

- Eine direkte Manipulation der Werte in der Struktur muss unterbleiben. Lediglich der lesende Zugriff auf skalare Werte ist erlaubt.
- Das direkte Kopieren dieser Struktur in einen anderen Speicherbereich oder an eine andere Adresse sowie die Verwendung einer solchen Kopie sind verboten! Hierfür kann keine korrekte Verwaltung und Zugriff gewährleistet werden!
- Erlaubt ist lediglich das Kopieren eines gemanagten Wertes in einen noch ungemanagten Rückgabeparameter.
- Eine einmal gesetzte lokale oder statische Verwendung kann nicht mehr geändert werden.
- Die Rückgabe eines lokalen Wertes, Strings oder Index sollte über die Funktionen **DM_ValueReturn()**, **DM_StringReturn()** und **DM_IndexReturn()** erfolgen, sodass vom IDM gewährleistet werden kann, dass auch von einer lokalen Variablen der Wert korrekt zurückgegeben wird. Bei statischen Werten und Strings ist dies nicht nötig.

Da die vom IDM gemanagten Werte auch Strings beinhalten können, ist zu beachten, dass diese Strings in der Applikations-Codepage gehalten und zurückgeliefert werden. Eine Änderung dieser Codepage durch einen **DM_Control()**-Aufruf führt nicht zu einer Codepage-Änderung der gemanagten Strings. Die Applikations-Codepage sollte also so früh wie möglich korrekt gesetzt werden.

Die Nutzung der nachfolgenden Funktionen ist nur für lokale und verteilte (DDM) C-Funktionen möglich. Eine Nutzung in Formatfunktionen ist nicht erlaubt.

Eine fehlerhafte Manipulation eines vom IDM gemanagten Wertes oder einer Zustandsänderung durch den Anwender wird in der C-Schnittstelle, wenn erkannt, durch die Fehlercodes *„DME_InvalidContext“* oder *„DME_BadContext“* gemeldet. Typischerweise sollten auch Ein- und Ausgabeparameter nicht auf die gleiche Wertereferenz zeigen, was wiederum mit einem Fehler `DME_InvalidArg` angezeigt wird.

Lokal gemanagte Wertereferenzen werden nach dem zugehörigen Funktionsaufruf aufgeräumt und falls nötig Unterstrukturen und Strings freigegeben.

Für besonders zeitkritische Funktionen sollte unter Umständen von der Nutzung gemanagter Werte und Strings Abstand genommen bzw. das Zeitverhalten geprüft werden, da deren Nutzung auf jeden Fall einen gesteigerten Verwaltungsaufwand im IDM bedeutet.

33.5.13 Neue und geänderte C-Schnittstellenfunktionen

33.5.13.1 DM_ValueInit

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

Die Wertereferenz wird mit dem entsprechenden Typ initialisiert. Erlaubt sind auch die Sammlungsdatentypen *DT_list*, *DT_vector*, *DT_hash*, *DT_matrix* und *DT_refvec*.

Wird die Wertereferenz über die Option *DMF_StaticValue* als statisch bzw. global initialisiert, so ist ein Zugriff auch außerhalb des Funktionsaufrufs möglich. Eine Freigabe von Wertelisten und Strings findet beim Funktionsende nicht statt. Nicht erlaubt ist die Initialisierung von Argumenten zu einer statischen bzw. globalen gemanagten Wertereferenz.

Initialisiert werden String-Werte mit dem *NULL*-Pointer. Sammlungen werden ohne Elementwerte angelegt. Auch alle anderen Wertetypen werden zu einem *0*-Wert initialisiert.

Das Hinzufügen oder Ändern von Werten oder Teilwerten bzw. Elementen kann über die Funktion **DM_ValueChange()** geschehen. Eine Reinitialisierung über **DM_ValueInit()** ist ebenfalls möglich.

```
DM_Boolean DML_default DM_EXPORT DM_ValueInit
(
  DM_Value   *value,
  DM_Type     type,
  DM_Value   *count,
  DM_Options  options
```

```
)
```

Parameter

-> DM_Value *value

Dieses ist die Wertereferenz, welche initialisiert werden soll.

-> DM_Type type

Dieser Parameter bezeichnet den geforderten initialen Typ.

-> DM_Value *count

In diesem Parameter kann die initiale Größe von Sammlungen wie *list* oder *matrix* angegeben werden oder der zugehörige Wertetyp bei *vector*-Werten.

-> DM_Options options

Option	Bedeutung
0	Wertereferenz wird als lokaler Wert initialisiert.
DMF_StaticValue	Wertereferenz wird als globaler, statischer Wert initialisiert.

Rückgabewert

DM_TRUE Funktion konnte erfolgreich ausgeführt werden, die Wertereferenz ist damit initialisiert.

DM_FALSE Wertereferenz konnte nicht initialisiert werden.

Beispiel

Dialogdatei

```
dialog Dialog

function void     AppendToList(anyvalue List input output, anyvalue Value);
function anyvalue FindMinMax(anyvalue IntegerList);
function string   ListToString(anyvalue List);
function void     SetElemSep(string Sep);

on dialog start
{
   variable vector[string] WeekDays:=["Mo","Tu","Wed","Thu","Fri","Sat"];
   variable list DaysPerWeek := [31,30,28,27];

   SetElemSep(" , ");
   AppendToList(WeekDays,"Sun");
   print ListToString(WeekDays);
```

```
   print FindMinMax(DaysPerWeek);
   exit();
}
```

C-Teil

```
...

static DM_String elemSep = NULL;

void DML_default DM_ENTRY SetElemSep(DM_String sep)
{
  DM_StringInit(&elemSep, DMF_StaticValue);
  DM_StringChange(&elemSep, sep, 0);
}

DM_Value* DML_default DM_ENTRY FindMinMax(DM_Value *IntegerList)
{
  DM_Value subval, minMaxList;  /* managed local values */
  DM_UInt  count;
  DM_UInt  minValueCount=0, maxValueCount=0;
  DM_Value index, mindex, data;  /* unmanaged values */

  /* initialize local managed values */
  DM_ValueInit(&subval, DT_void, NULL, 0);
  DM_ValueInit(&minMaxList, DT_list, NULL, 0);

  /* determine the itemcount of the list */
  count = DM_ValueCount(IntegerList, NULL, 0);
  index.type = DT_integer;
  index.value.integer = 1;

  while(count>0)
  {
    /* loop through the index 1,2,3,... */
    if (DM_ValueGet(IntegerList, &index, &subval, 0)
        && subval.type == DT_integer)
    {
      if (minValueCount==0 || subval.value.integer<minValue)
      {
        minValueCount++;
        /* store maximum value at [1] in minMaxList */
        mindex.type = DT_integer;
        mindex.value.integer = 1;
        DM_ValueChange(&minMaxList, &mindex, &subval, 0);
      }
      if (maxValueCount==0 || subval.value.integer>maxValue)
      {
```

```
        maxValueCount++;
        /* store maximum value at [2] in minMaxList */
        mindex.type = DT_integer;
        mindex.value.integer = 2;
        DM_ValueChange(&minMaxList, &mindex, &subval, 0);
      }
    }
    count--;
    index.value.integer++;
  }
  /* return the minMaxList (without compiler warnings) */
  return DM_ValueReturn(&minMaxList, 0);
}

...

void DML_default DM_ENTRY AppendToList(DM_Value *List, DM_Value *Value)
{
  DM_Value newList;

  /* demonstrate the returning of a list-value by two ways * /
  if (List->type == DT_list || List->type == DT_vector)
  {
    /* 1) returning a manipulated argument (auto-management) */
    DM_ValueChange(List, NULL, Value, DMF_AppendValue);
  }
  else
  {
    /* 2) creation of a managed local list-value */
    DM_ValueInit(&newList, DT_list, NULL, 0);
    DM_ValueChange(&newList, NULL, List, DMF_AppendValue);
    DM_ValueChange(&newList, NULL, Value, DMF_AppendValue);
    *List = newList;
  }
}

DM_String DML_default DM_ENTRY ListToString(DM_Value *List)
{
  DM_Value  index;
  DM_Value  value;
  DM_UInt   count;
  DM_String retString;

  /* initialize the managed values */
  DM_ValueInit(&index, DT_void, NULL, 0);
  DM_ValueInit(&value, DT_void, NULL, 0);
  DM_StringInit(&retString, 0);
```

```
  count = DM_ValueCount(List, NULL, 0);
  while(count>0)
  {
    if (DM_ValueIndex(List, count--, &index, 0)
        && DM_ValueGet(List, &index, &value, 0)
        && value.type == DT_string)
    {
      DM_StringChange(&retString, value.value.string, DMF_AppendValue);
      if (count>0)
        DM_StringChange(&retString, elemSep, DMF_AppendValue);
    }
  }
  return DM_StringReturn(retString, 0);
}
```

33.5.13.2 DM_ValueChange

Mit dieser Funktion kann eine von IDM gemanagte Wertereferenz manipuliert werden. Entweder kann der Gesamtwert ersetzt werden oder ein einzelner Elementwert in einer Sammlung.

Es findet eine automatische Umwandlung von einer ungemanagten Wertereferenz in eine gemanagte Wertereferenz statt. Ist in diesem Fall die Option *DMF_StaticValue* gesetzt, so wird eine statische bzw. globale gemanagte Wertereferenz gesetzt. Für eine bessere Kontrolle empfiehlt sich auch die explizite Nutzung von **DM_ValueInit()**.

Handelt es sich beim *value*-Parameter um eine Sammlung, z.B. vom Typ *DT_vector*, *DT_list*, *DT_hash*, *DT_matrix* oder *DT_refvec*, so kann durch Angabe des *index*-Parameters ein Elementwert ausgetauscht werden. Analog zu den vordefinierten Attributen ist eine Erweiterung einer Sammlung möglich indem schrittweise der um *+1* erhöhte Index verwendet wird. Für assoziative Felder ist einfach die Nutzung eines noch nicht verwendeten Indexschlüssels möglich.

Wird eine Sammlung im *data*-Parameter dem Ziel komplett (also mit *NULL* als *index*-Parameter) zugewiesen, so wird der gesamte Wert mit allen Werteelementen kopiert. Die Umwandlung eines Argumentes in einen lokal gemanagten Wert erfordert ebenso ein komplettes Kopieren um die weitere Manipulation zu ermöglichen.

```
DM_Boolean DML_default DM_EXPORT DM_ValueChange
(
  DM_Value   *value,
  DM_Value   *index,
  DM_Value   *data,
  DM_Options  options
)
```

Parameter

-> DM_Value * value

Dieser Parameter verweist auf die Wertereferenz, welche verändert werden soll. Falls sie noch nicht vom IDM gemanagt wird, wird sie in eine gemanagte Wertereferenz überführt.

-> DM_Value * index

Dieser Parameter kann für die Änderung von Elementwerten in Sammlungen verwendet werden und dient zur Angabe des Index. Ansonsten sollte er mit *NULL* belegt werden. Dieser Parameter muss nicht eine gemanagte Wertereferenz sein.

-> DM_Value * data

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln. Ist dieser Wert *NULL*, so wird der Wert bzw. das Element auf *DT_undefined* gesetzt.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *value*-Parameters in eine gemanagte Wertereferenz wird diese als statische bzw. globale Wertereferenz behandelt.
DMF_AppendValue	Bei Sammlungen wird der Datenwert aus *data* hinten angehängt. Der Index muss hierfür *NULL* sein.
DMF_SortBinary	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung des neu gesetzten Wertes. Kombinierbar mit *DMF_SortReverse*.
DMF_SortLinguistic	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung, welche für Strings nach linguistischen Regeln (siehe hierzu die Funktion **sort()**) erfolgt. Kombinierbar mit *DMF_SortReverse*.
DMF_SortReverse	Bei Sammlungen erfolgt für den neu gesetzten Wert abschließend eine Sortierung in umgekehrter Reihenfolge.

Rückgabewert

DM_TRUE	Funktion erfolgreich ausgeführt, die Wertesetzung war erfolgreich oder der Wert war schon gesetzt.

DM_FALSE	Wertesetzung konnte nicht durchgeführt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue FindData(hash DataHash, string Pattern,
                           anyvalue FirstIndex output);

on dialog start
{
  variable hash Stations := ["1"=>"ARD","2"=>"ZDF","9"=>"SWR3"];
  variable anyvalue Idx;

  print "Found(D)=" + FindData(Stations,"D",Idx);
  print " at " + Idx;
  exit();
}
```

C-Teil

```
...

static DM_Value InvalidIndex;
static DM_Value InvalidValue;

DM_Value * DML_default DM_ENTRY FindData(DM_Value *DataHash, DM_String
Pattern,
                                         DM_Value *FirstIndex)
{
  DM_Value index;
  DM_Value value;

  /* initialize the managed values */
  DM_ValueInit(&index, DT_void, NULL, 0);
  DM_ValueInit(&value, DT_void, NULL, 0);
  DM_StringInit(&retString, 0);

  count = DM_ValueCount(DataHash, NULL, 0);
  while(count>0)
  {
    /* loop through the hash */
    if (DM_ValueIndex(DataHash, count--, &index, 0)
        && DM_ValueGet(DataHash, &index, &value, DMF_GetLocalString)
```

```
        && value.type == DT_string)
    {
      if (strstr(value.value.string, Pattern))
      {
        /* return the first found index & value */
        DM_ValueChange(FirstIndex, NULL, &index, 0);
        return DM_ValueReturn(&value, 0);
      }
    }
  }

  /* return the invalid index & value */
  DM_ValueChange(FirstIndex, NULL, &InvalidIndex, 0);
  return &InvalidValue;
}

...

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

  ...

  data.type = DT_string;
  data.value.string = "NO-VALUE";
  DM_ValueChange(&InvalidValue, NULL, &data, DMF_StaticValue);
  data.type = DT_string;
  data.value.string = "INVALID-INDEX";
  DM_ValueChange(&InvalidIndex, NULL, &data, DMF_StaticValue);

  ...

  DM_StartDialog(...)
```

33.5.13.3 DM_ValueGet

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

Ist der definierte Index *NULL*, so wird der Gesamtwert zurückgegeben, was normalerweise einem Kopieren des Wertes entspricht.

Handelt es sich beim *retvalue*-Parameter um einen ungemanagten Wert, so bleibt dieser auch ungemanagt. Bei der Rückgabe von Strings werden diese nur in einem temporären Puffer gemerkt, der beim nächsten Aufruf einer DM-Funktion ohne *DMF_DontFreeLastStrings*-Option wieder gelöscht bzw. überschrieben werden kann.

```
DM_Boolean DML_default DM_EXPORT DM_ValueGet
(
  DM_Value   *value,
  DM_Value   *index,
  DM_Value   *retvalue,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Dieser Parameter verweist auf die Wertereferenz, von welcher der Elementwert geholt wird. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_Value * index

Dieser Parameter definiert den Index von dem der Elementwert geholt wird. Dieser Parameter muss kein gemanagter Wert sein. Falls der Parameter auf *NULL* gesetzt ist, wird der *value*-Parameter in den Rückgabeparameter *retvalue* kopiert.

-> DM_Value * retvalue

Dieser Parameter definiert den Wert, der gesetzt werden soll. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln. Ist dieser Wert *NULL*, so wird der Wert bzw. das Element auf *DT_undefined* gesetzt.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *value*-Parameters in eine gemanagte Wertereferenz wird diese als statische bzw. globale Wertereferenz behandelt.
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.

Option	Bedeutung
DMF_ DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

DM_ TRUE	Das Holen des Wertes war erfolgreich.
DM_ FALSE	Der Wert konnte nicht geholt werden. Das kann auf fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Indizierung hinweisen.

Beispiel

Siehe **DM_ValueChange()**, **DM_ValueCount()**, **DM_ValueIndex()**

33.5.13.4 DM_ValueCount

Liefert die Anzahl der Werte in einer Sammlung (ohne die Standardwerte) zurück. Wenn gewünscht kann auch der Indextyp bzw. der höchste Indexwert zurückgeliefert werden.

Der zurückgelieferte Wert gibt die Anzahl der Werte (ohne die Standardwerte) an. Zusammen mit der Funktion **DM_ValueIndex()** lassen sich so einfach Schleifen über alle indizierten Werte bzw. Elemente realisieren.

Folgende Rückgaben sind je nach *value*-Parameter zu erwarten:

***value*-Typ**	***retvalue*-Rückgabetyp**	**Bemerkung**
DT_refvec, *DT_list*, *DT_vector*	*DT_integer*	Höchster Indexwert
DT_matrix	*DT_index*	Höchster Indexwert
DT_hash	*DT_datatype*	Beliebiger Indextyp (*anyvalue*)
ansonsten	*DT_void*	Unindizierter WertN

```
DM_UInt DML_default DM_EXPORT DM_ValueCount
(
  DM_Value   *value,
  DM_Value   *retvalue,
  DM_Options  options
```

```
)
```

Parameter

-> DM_Value* value

Diese Parameter verweist auf die Wertereferenz, von welcher die Werteanzahl geholt wird. Es sollte eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_Value * retvalue

Wenn dieser Parameter nicht *NULL* ist, wird hierin der Count-Wert zurückgeliefert. Es kann sich hierbei um eine gemanagte Wertereferenz handeln, muss aber nicht.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

0 … INT_MAX	Anzahl von Werten (ohne die Standardwerte bei Index *[0]*, *[0,*]* oder *[*,0]*).
retvalue	Höchster Indexwert, kann entweder *void* (skalarer Wert), ein *integer*-Wert (eindimensionales Feld), ein *index*-Wert (zweidimensionales Feld) oder ein Datentyp (assoziatives Feld) sein.

Beispiel

Dialogdatei

```
dialog YourDialog
function integer CountIntegers(anyvalue List);

on dialog start
{
  variable matrix M := [
    [0,0]=>-1,[1,1]=>"PLZ",[1,2]=>"City",[2,1]=>53123,[2,2]=>"Bonn" ];
  print "#Integers in Hash: " + CountInteger(M);
  exit();
}
```

C-Teil

```
...

static DM_Value InvalidIndex;
static DM_Value InvalidValue;

DM_Integer DML_default DM_ENTRY CountInteger(DM_Value *List)
{
  DM_Value   index;
  DM_Value   count;
  DM_Value   value;
  DM_Integer icount = 0;

  /* initialize the managed values */
  if (DM_ValueCount(List, &count, 0)>0 && List->type == DT_matrix
      && count->type == DT_index)
  {
    index.type = DT_index;
    index.value.index.first = 0;
    index.value.index.second = 1;

    /* loop through [0,1],[1,1],[2,1],... */
    while(index.value.index.first<=count.value.index.second)
    {
      if (DM_ValueGet(List, &index, &value, 0)
          && value.type == DT_integer)
        icount++;
      index.value.index.first++;
    }
  }
  return icount;
}
```

Siehe auch

itemcount(), **countof()**, **DM_ValueGet()**, **DM_ValueIndex()**, **DM_ValueChange()**

33.5.13.5 DM_ValueIndex

Mit dieser Funktion kann der zu einer Position zugehörige Index einer Sammlung ermittelt werden. Dies ist vor allem wichtig für Werte vom Typ *DT_hash* und *DT_matrix* um so auf einfachste Weise auf alle zugehörigen Indizes zugreifen zu können. Die Funktion erlaubt nur den Zugriff auf Indizes die nicht zu Standardwerten gehören.

Der zurückgelieferte Index kann in einem gemanagten *retvalue*-Parameterwert abgelegt werden oder in einem ungemanagten. Bei letzterem wird wenn nötig eine Allokierung des String-Wertes im temporären Puffer ausgeführt.

```
DM_UInt DML_default DM_EXPORT DM_ValueIndex
(
  DM_Value   *value,
  DM_UInt     indexpos,
  DM_Value   *retvalue,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Diese Parameter verweist auf die Wertereferenz, von welcher der Index an der Position geholt wird. Es muss eine gemanagte Wertereferenz oder Funktionsargument sein.

-> DM_UInt indexpos

Dieser Parameter definiert die Position des Index und sollte zwischen *0 < indexpos <= DM_ValueCount()* liegen.

-> DM_Value * retvalue

Wenn dieser Parameter nicht *NULL* ist, wird hier der entsprechende Index abgelegt. Es kann sich dabei um eine gemanagte oder auch um eine ungemanagte Wertereferenz handeln.

If this parameter is not *NULL*, the retrieved index is stored here. This may be a managed or an unmanaged value reference.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_GetLocalString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der aktuell eingestellten Sprache zurückgegeben werden sollen.
DMF_GetMasterString	Diese Option bedeutet, dass bei textuellen Werten (IDs vom Typ *DT_text*) diese als String in der Entwicklungssprache zurückgegeben werden sollen, unabhängig davon, mit welcher Sprache der Benutzer gerade arbeitet.
DMF_DontFreeLastStrings	Normalerweise werden Strings in einem temporären Puffer an die Anwendung übergeben, der bis zum nächsten Aufruf an den IDM erhalten bleibt. Sollen Strings länger in der Anwendung gültig sein, muss diese Option *DMF_DontFreeLastStrings* gesetzt werden. Der Speicher wird erst dann wieder freigegeben, wenn ein Aufruf an eine IDM-Funktion ohne diese Option erfolgt und dabei ein String vom IDM an die Anwendung zurückgegeben wird.

Rückgabewert

DM_TRUE	Der Wert hat einen Index an dieser Position, der geholte Index steht danach in **retvalue* falls *retvalue != NULL* ist.
DM_FALSE	Der Index konnte nicht ermittelt werden. Das kann auf einen fehlerhaften Aufruf, eine nicht gemanagte oder ungültige Wertereferenz oder eine fehlerhafte Position hinweisen.

Beispiel

Siehe **DM_ValueChange()**

Siehe auch

itemcount(), **countof()**, **:index()**, **DM_ValueGet()**, **DM_ValueIndex()**, **DM_ValueChange()**, **DM_ValueCount()**

33.5.13.6 DM_ValueReturn

Diese Funktion dient zur sicheren Funktionsrückgabe von lokalen **DM_Value**-Werten an den Aufrufer. Bei Verwendung von lokalen Variablen und Strukturen in einer C-Funktion sind diese nach der Rückgabe ungültig. Mit dieser Funktion kann problemlos und sicher eine lokale **DM_Value**-Variable zurückgegeben werden.

Zu diesem Zweck wird, wenn nötig, eine temporäre Kopie des zurückzugebenden Wertes angelegt (z.B. der darin liegende String kopiert). Ein Kopieren findet für gemanagte Wertereferenzen nicht statt. Insofern sollte der zurückgegebene **DM_Value**-Pointer dann mit `return` an den Aufrufer weitergegeben werden.

```
DM_Value* DML_default DM_EXPORT DM_ValueReturn
(
  DM_Value   *value,
  DM_Options  options
)
```

Parameter

-> DM_Value* value

Dieser Parameter verweist auf die Wertereferenz welche zurückgegeben werden soll. Es kann sich dabei um eine gemanagte Wertereferenz handeln, muss aber nicht.

-> DM_Options options

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

Rückgabewert

Zurückgegeben wird ein Zeiger auf eine gültige **DM_Value**-Struktur oder *NULL* im Fehlerfall. Ein Fehler kann z.B. vorliegen, wenn die Funktion im falschen „runstate“ aufgerufen wird, der gemanagte String ungültig ist oder das Kopieren nicht ausgeführt werden konnte.

Zu beachten

Für die Rückgabe von *output*-Parametern können die Funktionen **DM_ValueInit()** und **DM_ValueChange()** verwendet werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue StringOf(integer I);

on dialog start
{
  print StringOf(123);
  print StringOf(-42);
  exit();
}
```

C-Teil

```
...

DM_Value * DML_default DM_ENTRY StringOf(DM_Integer I)
{
  char     buf[10];
  DM_Value data;
```

```
  DM_Value negative;

  if (I>=0)
  {
    /* return an unmanaged value */
    sprintf(buf, "%d", (int)I);
    data.type = DT_string;
    data.value.string = buf;
    /* wrong: return &data => data is a local structure! */
    return DM_ValueReturn(&data, 0);
  }
  else
  {
    /* return a managed value */
    DM_ValueInit(&negative, DT_string, NULL, 0);  /* can be omitted */
    data.type = DT_string;
    data.value.string = "!!negative!!";
    DM_ValueChange(&negative, NULL, &data, 0);
    /* return &negative; => possible but generates compiler warning! */
    return DM_ValueReturn(&negative);
  }
}
```

Siehe auch

itemcount(), **countof()**, **:index()**, **DM_ValueGet()**, **DM_ValueIndex()**, **DM_ValueChange()**, **DM_ValueCount()**

33.5.13.7 DM_StringInit

Mit dieser Funktion kann ein String in einen vom IDM gemanagten lokalen oder globalen bzw. statischen String umgewandelt werden. Dadurch sind weitere Manipulationen des Strings durch die **DM_StringChange()**-Funktion möglich, wie auch die erleichterte Rückgabe als Parameter oder Rückgabewert. Der IDM übernimmt dann die Verwaltung des Speichers für den String.

Für gemanagte Strings darf nur ein lesender Zugriff auf die String-Zeichen oder den String-Pointer erfolgen. Eine Freigabe über **DM_Free()** darf nicht erfolgen!

Der String wird durch diese Funktion mit *NULL* initialisiert.

Durch Angabe der Option *DMF_StaticValue* wird der String als statisch angesehen und nicht, wie sonst für lokale Strings üblich, nach der Rückkehr der Funktion freigegeben.

Grundsätzlich dürfen nur String-Argumente oder lokale bzw. globale Strings gemanagt werden.

```
DM_Boolean DML_default DM_EXPORT DM_StringInit
(
  DM_String * pstring,
  DM_Options  options
```

```
)
```

Parameter

-> DM_String * pstring

Dieser Parameter verweist auf den String-Pointer der zu initialisieren ist und gemanagt werden soll.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Der *pstring*-Parameter wird als statische bzw. globale String-Referenz initialisiert.

Rückgabewert

DM_TRUE	String wurde erfolgreich initialisiert und ist nun gemanagt.
DM_FALSE	Der String konnte nicht gemanagt oder initialisiert werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function string FormatString(integer Op, string String);

on dialog start
{
  print FormatString (-1, "**");          // set decoration
  print FormatString (0, "hello world");  // print with decoration
  exit();
}
```

C-Teil

```
...

DM_String DML_default DM_ENTRY FormatString (DM_Integer Op, DM_String String)
{
  static DM_String decoration = NULL;
  DM_String newstring;

  if (!decoration)
  {
```

```
    /* static initialization */
    DM_StringInit(&decoration, DMF_StaticValue);
    DM_StringChange(&decoration, "-", 0);
    /* above can be done simpler via:
     * DM_StringChange(&decoraton, "-", DMF_StaticValue);
     */
  }

  switch(Op)
  {
  case -1: /* change decoration */
    DM_StringChange(&decoration, String);
    return &decoration;
  case 1:  /* decor only at the beginning */
    DM_StringChange(&newString, decoration, 0);
    DM_StringChange(&newString, String, DMF_AppendValue);
    break;
  case 2:  /* decor only at the end */
    DM_StringChange(&newString, String, 0);
    DM_StringChange(&newString, decoration, DMF_AppendValue);
    break;
  default:
    DM_StringChange(&newString, decoration, 0);
    DM_StringChange(&newString, String, DMF_AppendValue);
    DM_StringChange(&newString, decoration, DMF_AppendValue);
    break;
  }
  return newstring;
  /* also possible: return DM_StringReturn(&newstring, 0); */
}
```

Siehe auch

DM_ValueInit(), **DM_StringChange()**, **DM_StringReturn()**

33.5.13.8 DM_StringChange

Diese Funktion kann zur Änderung bzw. Manipulation eines gemanagten Strings verwendet werden.

Neben der reinen Ersetzung des Strings ist über die *DMF_AppendValue*-Option das Konkatenieren von Strings möglich.

Handelt es sich beim *pstring*-Parameter um einen noch ungemanagten String, so wird dieser kopiert und von da an vom IDM gemanagt. Wurde dabei noch die Option *DMF_StaticValue* gesetzt, wird dieser als statischer bzw. globaler String behandelt und nicht wie sonst bei der Rückkehr der Funktion wieder freigegeben.

Grundsätzlich dürfen nur String-Argumente oder lokale bzw. globale Strings gemanagt werden.

```
DM_Boolean DML_default DM_EXPORT DM_StringChange
(
  DM_String * pstring,
  DM_String   newstring,
  DM_Options  options
)
```

Parameter

<-> DM_String * pstring

Dieser Parameter verweist auf die String-Referenz die manipuliert werden soll. Es kann sich dabei um einen schon gemanagten String handeln, muss es aber nicht.

-> DM_String newstring

Dieser Parameter verweist auf den String, welcher der String-Referenz (*pstring*-Parameter) zugewiesen oder an sie angehängt werden soll.

-> DM_Options options

Hier sind folgende Optionen möglich:

Option	Bedeutung
DMF_StaticValue	Bei der automatischen Umwandlung des *pstring*-Parameters in eine gemanagte String-Referenz wird diese als statische bzw. globale String-Referenz behandelt.
DMF_AppendValue	Der String im *newstring*-Parameter wird an den String im *pstring*-Parameter angehängt.

Rückgabewert

DM_TRUE	String wurde erfolgreich manipuliert.
DM_FALSE	Ein Fehler ist aufgetreten. Dabei kann es sich um einen falschen „runstate“ bei Aufruf handeln, oder aber eine ungültigen String-Referenz.

Beispiel

Siehe **DM_StringReturn()**

Siehe auch

DM_StringInit(), **DM_StringReturn()**, **DM_ValueChange()**

33.5.13.9 DM_StringReturn

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

Zu diesem Zweck wird, wenn nötig, eine temporäre Kopie angelegt (z.B. der darin liegende String kopiert). Ein Kopieren findet für gemanagte Wertereferenzen nicht statt. Der zurückgegebene *DM_String* sollte dann mit `return` an den Aufrufer weitergegeben werden.

```
DM_String DML_default DM_EXPORT DM_StringReturn
(
  DM_String * pstring,
  DM_Options  options
)
```

Parameter

-> DM_String * pstring

Dieser Parameter verweist auf den String welcher zurückgegeben wird. Es kann sich dabei um einen gemanagten String handeln, muss aber nicht.

-> DM_Options options

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

Rückgabewert

Zurückgegeben wird ein für die Funktionsrückgabe gültiger String oder *NULL* im Fehlerfall. Ein Fehler kann z.B. vorliegen wenn die Funktion im falschen „runstate“ aufgerufen wird, der gemanagte String ungültig ist oder das Kopieren nicht ausgeführt werden konnte.

Zu beachten

Für die Rückgabe von *output*-String-Parametern können die Funktionen **DM_StringInit()** und **DM_StringChange()** verwendet werden.

Beispiel

Dialogdatei

```
dialog YourDialog
function anyvalue StringOf(integer I);

on dialog start
{
  print StringOf(123);
  print StringOf(-42);
  exit();
```

```
}
```

C-Teil

```
...

DM_String DML_default DM_ENTRY StringOf(DM_Integer I)
{
  char      buf[10];
  DM_String data;
  DM_String negative;

  if (I>=0)
  {
    /* return an unmanaged local string */
    sprintf(buf, "%d", (int)I);
    data = buf;
    /* wrong: return data; => buf is a local char array! */
    return DM_StringReturn(&data, 0);
  }
  else
  {
    /* return a managed string */
    DM_StringInit(&negative, 0);  /* can be omitted */
    DM_StringChange(&negative, "!!negative", &data, 0);
    return DM_StringReturn(&negative);
    /* return &negative; => also possible for managed strings! */
  }
}
```

Siehe auch

DM_ValueReturn(), **DM_IndexReturn()**, **DM_StringInit()**, **DM_StringChange()**

33.5.13.10 DM_IndexReturn

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

Zu diesem Zweck wird eine temporäre Kopie angelegt.

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

```
)
```

Parameter

-> DM_Index * pindex

Dieser Parameter verweist auf den Index welcher zurückgegeben wird.

-> DM_Options options

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

Rückgabewert

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

Beispiel

Dialogdatei

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

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

C-Teil

```
...

DM_Index* DML_default DM_ENTRY StringOf(DM_Index* Idx)
{
  DM_Index newIdx;
  newIdx.first = Idx->second;
  newIdx.second = Idx->first;

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

Siehe auch

DM_ValueReturn(), **DM_StringReturn()**

33.6 Datenmodell

33.6.1 Motivation

Die Motivation für die Einführung des neuen „Datenmodells“ im IDM ist die damit erreichbare bessere Entkopplung zwischen Oberflächenobjekten und der Applikationsschicht, welche die Daten für die Darstellung liefert.

Die Hauptmerkmale für dessen Design sind:

- Trennung der Benutzerschnittstelle und der Applikation auf Basis des Entwurfsmusters Model-View-Presenter (MVP).
- Der Dialogentwickler definiert lediglich die Kopplung zwischen Oberflächenobjekten (View) zur Datenhaltung in der Applikationsschicht (Model).
- Möglichst kein bzw. wenig Regelcode durch den steuerbaren Einsatz einer automatischen Synchronisierung sowie durch automatische Typkonvertierungen.
- Beibehaltung des schon bekannten Datenschemas und Zugriffskonzeptes für indizierte Attribute sodass Daten in skalarer oder vektorieller Form verwendbar sind.
- Nutzung der bestehenden IDM-Möglichkeiten für die Datenhaltung bzw. für die Implementierung der Model-Schicht: Objekte des IDM (***Record***-Objekt) wie auch Applikationsfunktionen die ganz transparent auch auf der DDM-Serverseite liegen können.

33.6.2 Einführung

Der IDM erlaubt mit dem neu eingeführten Datenmodell die Entkopplung in drei Komponenten[1] Model, View und Presenter auf Basis des MVP-Entwurfsmusters mit Anpassung an die Eigenheiten und Erfordernisse der Regelsprache.

Model

Verwaltet die Daten und die dahinterstehende Logik. Ist typischerweise nahe bzw. in der Applikationsschicht. Ein Model kann dabei mehrere Daten verwalten, welche vom IDM als indizierte Attribute betrachtet werden und somit Skalare, Vektoren und Matrizen beinhalten können.

Zum Beispiel ein ***Record***-Objekt mit benutzerdefinierten Attributen und Methoden.

View

Visuelle Repräsentation der Daten für die Darstellung, Eingabe oder Modifikation. Das Datenschema ist das gleiche wie beim Model: indizierte Attribute die somit Skalare, Vektoren oder Matrizen beherbergen.

Zum Beispiel ein ***Edittext***-Objekt das ein Format für die Ein- und Ausgabe verwendet.

[1] Die Komponenten werden mit ihrer englischen Bezeichnung verwendet um damit eine Unterscheidung zum Begriff Modell zu ermöglichen.

Presenter

Die Steuerlogik um die Bindung zwischen Model und View zu regeln oder um mehrere Views zu koppeln.

Zum .Beispiel in Form von Ereignisregeln um beim Beenden der Editierung den neuen Datenwert am View dem Model zuzuweisen.

Kurzer Begriffshinweis: In dieser Dokumentation gibt es für den Begriff „Datenmodell“ quasi zwei Bedeutungen. Erstens das vom MVP abgeleitete Konzept sowie zweitens ein konkretes Model.

Im Gegensatz zu bestehenden MVP-Konzepten in JAVA oder QT erlaubt der Ansatz im IDM die transparente Behandlung ohne eine Kopplung zu reglementieren oder die Unterscheidung zwischen Inhalt bzw. Selektion vorzuschreiben. Ob ein Model also den Inhalt eines Edittextes, die Farbe einer speziellen Tabellenzelle oder das aktive Element einer Liste steuert, bleibt dem Anwendungs- bzw. Dialogprogrammierer überlassen.

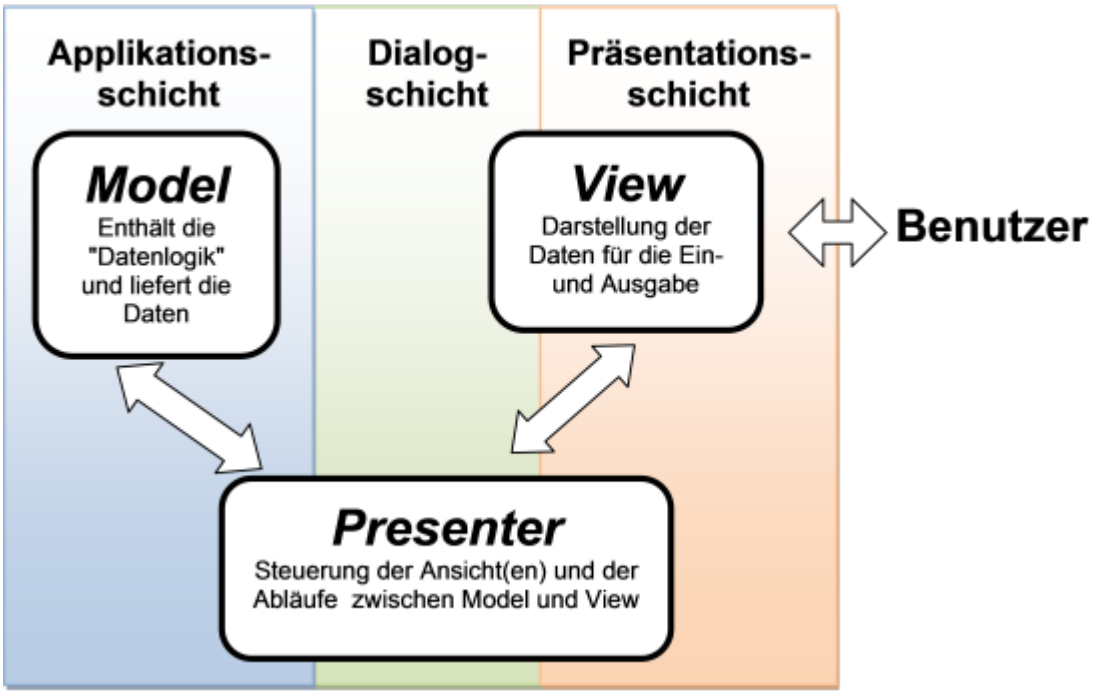


Abbildung 31: *Architekturmuster Model-View-Presenter*

Im IDM werden dabei die Kopplung zwischen Model und View sowie die wichtigsten Steuerfunktionen des Presenters durch einige wenige Attribute geregelt:

Tabelle 14: *Datenmodellattribute*

Attribut	Bedeutung
.datamodel	Verweis an der View auf das Model

Attribut	Bedeutung
.dataget	Definiert die Kopplung eines View-Attributs mit einem Model-Attribut für die Anzeige
.dataset	Definiert die Kopplung eines View-Attributs mit einem Model-Attribut für die Zuweisung
.dataindex	Erlaubt die Steuerung der Aggregation zwischen View- und Model-Attributen
.datamap	Erlaubt die Zuordnung von Attributen zwecks Auflösung von „Mehrdeutigkeiten“
.dataoptions	Steueroptionen der Presenter-Logik

Ein **Beispiel** für die Kopplung einer Dialogoberfläche (View) mit einem Datenmodell (Model) für die Anzeige von Personeninformationen mit Name, Geschlecht und Kindernamen kann etwa wie folgt aussehen:

```
dialog D
record RecPerson {
  string  Name   := "Ina Mustermann";
  boolean Female := true;
  string Children[integer];
  .Children[1] := "Anja";
  .Children[2] := "Peter";
}
window Wi {
  .title "Datamodel Example RecPerson";
  .width 200;
  .height 200;
  /* Kopplung aller View-Objekte mit dem Model RecPerson */
  .datamodel RecPerson;

  edittext EtName
  {
    .xauto 0;
    /* .content wird mit RecPerson.Name verknuepft */
    .dataget .Name;
  }

  checkbox CbGender
  {
    .ytop 30;
    .text "Female";
    /* .active wird aus RecPerson.Female befuellt */
    .dataget .Female;
  }

  listbox LbChildren
```

```
  {
    .ytop 60;
    .xauto 0;  .yauto 0;
    /* .content[] wird aus RecPerson.Children[] befuellt */
    .dataget .Children;
  }

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

Der Datenmodell-Ansatz im IDM nimmt dem Anwendungsprogrammierer einen Teil der Presenter-Logik ab, weshalb das obige Beispiel auch ohne weitere Regeln problemlos funktioniert. Das Befüllen der Oberflächenobjekte „EtName“, „CbGender“ und „LbChildren“ übernimmt der IDM durch die gesetzte Kopplung über die Attributdefinitionen *.datamodel* und *.dataget*. Weiter macht sich das obige Beispiel die Eigenschaft der zusätzlichen Wertweitergabe vom Vater zum Kind zunutze, welche für die Attribute *.datamodel* und *.dataoptions* gilt.

Das folgende Schaubild dient dazu diese Automatismen zu verstehen, sowie die Steuerungs- und Einflussmöglichkeiten aufzuzeigen, welche der Dialogprogrammierer für die Komponenten View und Presenter hat sowie der Applikationsprogrammierer für die Model-Komponente.

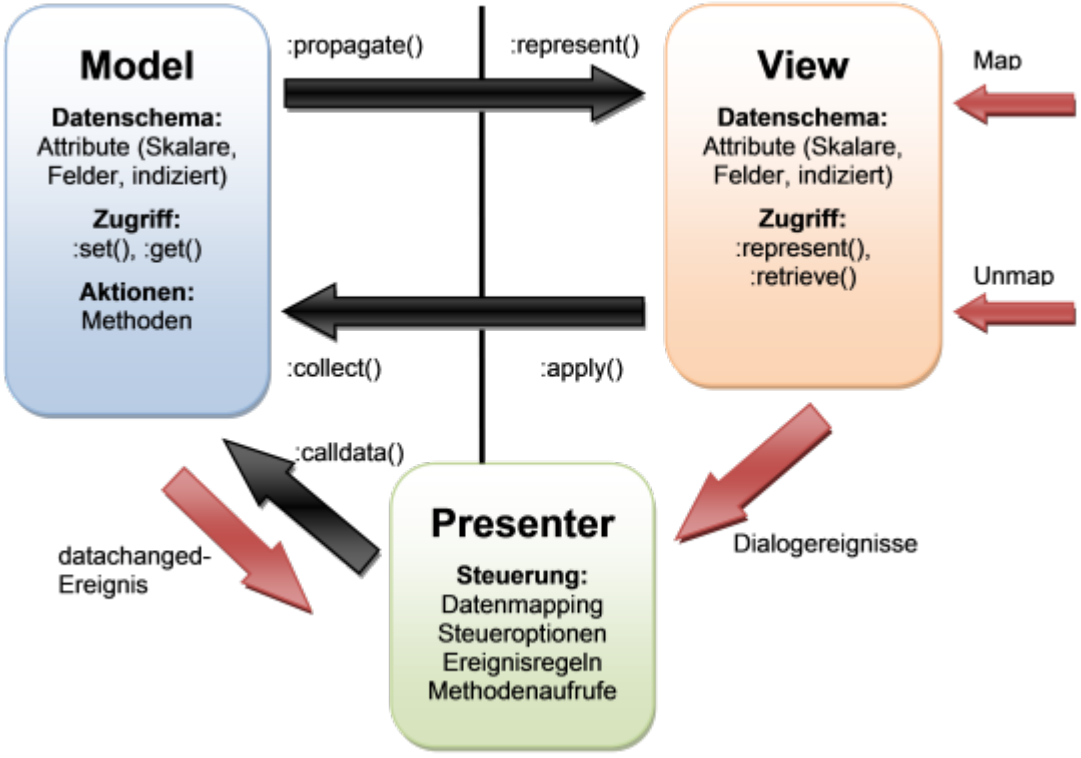


Abbildung 32: *Kopplungen zwischen Model, View und Presenter*

***Tabelle 15:** Weitergabe von Änderungen zwischen Modell, View und Presenter*

Methode, Funktion	Komponente	
:represent()	View	View-Objekt wird mit den Daten des Model befüllt und ggf. wird ein Datenwert wird für die Anzeige aufbereitet
:apply()	View	View-Objekt weist die Daten an das Model zurück, ggf. werden Datenwerte für Zuweisung aufbereitet
Dialogereignis	View, Presenter	Änderungen der View werden signalisiert
:propagate()	Model	Daten der Model-Komponente werden an die verknüpften View-Objekte zur Anzeige weitergegeben
:collect()	Model	Die Model-Komponente holt sich alle Werte aus den verknüpften View-Objekten
DM_DataChanged() bzw. Attributzuweisung	Model	Die Model-Komponente signalisiert eine Änderung an den Daten
.dataoptions[]	Presenter, Model	Steuerungsoptionen für die automatische Synchronisation zwischen Model und View

Die Presenter-Logik im IDM umfasst aktuell vier Automatismen:

1. Wird ein View-Objekt mit einer Kopplung zu einem Model sichtbar („Map“), so holt es sich die Daten kurz bevor es sichtbar wird (standardmäßig aktiv).
2. Wird ein View-Objekt mit einer Kopplung zu einem Model unsichtbar („Unmap“), so weist es die Daten dem zugeordneten Model zu (standardmäßig inaktiv).
3. Ein Dialogereignis an einem View-Objekt mit einer Kopplung zu einem Model, welches eine Änderung an einem gekoppelten Attribut anzeigt, weist die Daten dem zugeordneten Model zu (standardmäßig inaktiv).
4. Änderungen an Daten eines Model, z.B. über ein *datachanged*-Ereignis signalisiert, führen zu einer Propagierung der Änderung an die gekoppelten View-Objekte wenn diese sichtbar sind (standardmäßig aktiv).

33.6.3 Kopplung zwischen Model und View

Grundsätzlich ist das Datenmodell so konzipiert, dass dem Model nicht bekannt ist, von welchen View-Objekten es genutzt wird. Das Datenschema ist für Model und View gleich: indizierte Attribute um somit Skalare, Vektoren und Matrizen zu erlauben.

Die Kopplung einer View mit einem Model geschieht über das *.datamodel*-Attribut:

```
object .datamodel[<View-Attribut>] <Model-ID>
```

Für die Datenübertragung vom Model zur View muss zusätzlich noch definiert werden, welches Attribut des Models auszulesen ist.

```
attribute .dataget[<View-Attribut>] <Model-Attribut>
```

Um die Rückübertragung von Daten von der View zum Model zu definieren dient das folgende Attribut:

```
attribute .dataset[<View-Attribut>] <Model-Attribut>
```

So kann exakt festgelegt werden, was im View-Objekt durch ein Model befüllt bzw. zurückreflektiert wird. Durch die Indizierung am *.datamodel*-Attribut ist auch die Nutzung von mehreren Datenmodellen für unterschiedliche Attribute möglich.

View-Attribute sind typischerweise, aber nicht zwangsläufig, reale Attribute des View-Objekts, also z.B. *.content* für einen ***Edittext*** als View-Objekt. Es empfiehlt sich, als Model-Attribute benutzerdefinierte Attribute zu verwenden, um so eine bessere Unterscheidbarkeit zwischen View- und Model-Attributen zu erreichen.

Um die Kopplung zu „aktivieren“ bzw. wirksam werden zu lassen, muss immer das *.datamodel*-Attribut ohne Index gesetzt sein, sowie jeweils eine Kopplung über *.dataget* bzw. *.dataset* (mit oder ohne Index).

Hier ein **Beispiel** für die Nutzung von mehreren Models (Datenmodelle: „RecUsers“, „RecManager“, „VarValid“, „LbUsers“) durch mehrere Views („listbox LbUsers“, „edittext EtName“, „pushbutton PbKill“).

```
dialog D
record RecUsers
{
  string Name[integer];
  .Name[1] := "mueller";
  .Name[2] := "schrade";
  .Name[3] := "maier";
  string Rights[integer];
  .Rights[1] := "guest";
  .Rights[2] := "user";
  .Rights[3] := "root";
}

record RecManager
{
  string CurrUser := "maier";
  boolean IsAdmin := true;
  rule boolean ChangeUser(string Name)
  {
```

```
    variable anyvalue Idx;
    Idx := RecUsers:find(.Name, Name);
    if typeof(Idx) = integer then
      this.CurrUser := Name;
      this.IsAdmin := stringpos(RecUsers.Rights[Idx], "root") > 0;
      return true;
    endif
    return false;
  }
}

color CoError "red";
variable object VarError := true;

window Wi
{
  .title "Datamodel Model-View Coupling";
  .width 200;  .height 200;
  boolean Valid := false;

  listbox LbUsers
  {
    .xauto 0;  .yauto 0;
    .ybottom 60;
    .datamodel RecUsers;
    .datamodel[.activeitem] RecManager;
    .dataget .Name;
    .dataget[.activeitem] .CurrUser;
    on select, .activeitem changed
    {
      EtName.dataindex[.content] := this.activeitem;
      VarError := null;
      RecManager:ChangeUser(EtName.content);
    }

    :represent()
    {
      if Attribute = .activeitem then
        Value := this:find(.content, Value);
      endif
      pass this:super();
    }
  }

  edittext EtName
  {
    .yauto -1;  .xauto 0;
```

```
    .ybottom 30;
    .datamodel LbUsers;
    .datamodel[.bgc] VarError;
    .dataget .content;
    .dataget[.bgc] .value;
    .dataindex[.content] 0;
    on deselect_enter
    {
      if not RecManager:ChangeUser(this.content) then
        VarError := CoError;
      endif
    }
  }

  pushbutton PbKill
  {
    .yauto -1;
    .text "Kill All Processes";
    .sensitive false;
    .datamodel RecManager;
    .dataget[.sensitive] .IsAdmin;
  }

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

Um gebräuchliche Kopplungen zwischen View- und Model-Attributen möglichst einfach zu definieren, kann auf die **Standardvorgaben** des IDM zurückgegriffen werden (im Beispiel bei „EtName“ zu sehen, hier werden *.datamodel* und *.dataget* ohne Index verwendet). Diese Vorgaben sind in folgender Tabelle definiert:

Tabelle 16: *Standardkopplungen von View-Attributen*

Objektklasse	Standard-View-Attribut für *.dataget*	Standard-View-Attribut für *.dataset*
pushbutton statictext messagebox	*.text*	–
checkbox radiobutton timer	*.active*	*.active*
image menuitem	*.text*	*.active*

Objektklasse	Standard-View-Attribut für *.dataget*	Standard-View-Attribut für *.dataset*
filereq	*.value*	*.value*
listbox ***treeview*** ***tablefield***	*.content*	*.activeitem*
notepage	*.title*	–
edittext	*.content*	*.content*
poptext	*.text*	*.activeitem*
progressbar	*.curvalue*	–
scrollbar ***spinbox***	*.curvalue*	*.curvalue*
menubox ***toolbar*** ***window***	*.title*	–

Obiges Beispiel weist noch eine Besonderheit auf. Einmal wird ein sichtbares Objekt als Datenmodell verwendet (die View-Komponente „EtName“ ist gekoppelt mit dem Model „LbUsers“).

Grundsätzlich unterstützt der IDM als Datenmodell alle Objektklassen die benutzerdefinierte Attribute erlauben, außerdem noch globale Variablen sowie Funktionen mit dem Funktionstyp **datafunc**. Eine Kopplung von mehreren Modellen ist ebenso möglich. Bei sichtbaren (visuellen) Objekten sollte man aber eines bedenken: die Benutzerinteraktion bewirkt kein *changed*-Ereignis für Attribute und somit auch keine automatische Änderungsweitergabe an das Model. Dies muss explizit bzw. durch die entsprechende Synchronisationseinstellung geschehen.

Als View-Objekt sind ebenso alle Objektklassen erlaubt, die benutzerdefinierte Attribute erlauben. Allerdings sind die Automatismen für die Synchronisation zwischen Model und View dafür ausgelegt, dass es sich bei den View-Objekten um Instanzen mit visueller Ausprägung handelt.

33.6.4 Reihenfolge und Werteaggregation

Für die Befüllung eines View-Objekts ist eine korrekte Reihenfolge beim Setzen der View-Attribute notwendig. Sie werden vom IDM in einer vordefinierten, klassenspezifischen Reihenfolge (wie in der Attributliste des Objekts) gesetzt, um die Abhängigkeiten der Attribute untereinander zu berücksichtigen. Für einen ***Poptext*** gilt zum Beispiel die Reihenfolge

…, *.itemcount*, …, *.text[]*, …, *.activitem*, …, *.userdata[]*, …

um so die Abhängigkeit von *.text[]*, *.activeitem* und *.userdata[]* vom *.itemcount*-Attribut zu berücksichtigen. Diese Reihenfolge kann aber nur bei der Repräsentation (also einer Synchronisierung die

an der View-Komponente ausgelöst wird, z.B. über **:represent()**) berücksichtigt werden. Der IDM hat keinen Einfluss auf die Reihenfolge wenn die Synchronisierung von der Model-Komponente ausgelöst wird, z.B. bei Nutzung der **DM_DataChanged()**-Funktion.

Falls ein Datenmodell mit allen vier Attributen gekoppelt ist, so erfolgt zuerst das Setzen der Elementanzahl, dann der Texte, dann die Umsetzung des aktiven Elements und zum Schluss das Setzen des *.userdata[]*-Felds.

Grundsätzlich verbietet der IDM nicht die Nutzung von benutzerdefinierten Attributen als View-Attribute, kann aber für diese keine konsistente Reihenfolge gewährleisten. Benutzerdefinierte Attribute und vordefinierte Attribute die nicht zur Objektklasse gehören, werden auf jeden Fall nach den vordefinierten Attributen behandelt, für welche die Objektklasse eine Reihenfolge vorgibt.

Für die Werteaggregation, also die Beziehung oder Relation von Datenwerten zwischen Model und View gibt es dabei unterschiedliche Arten:

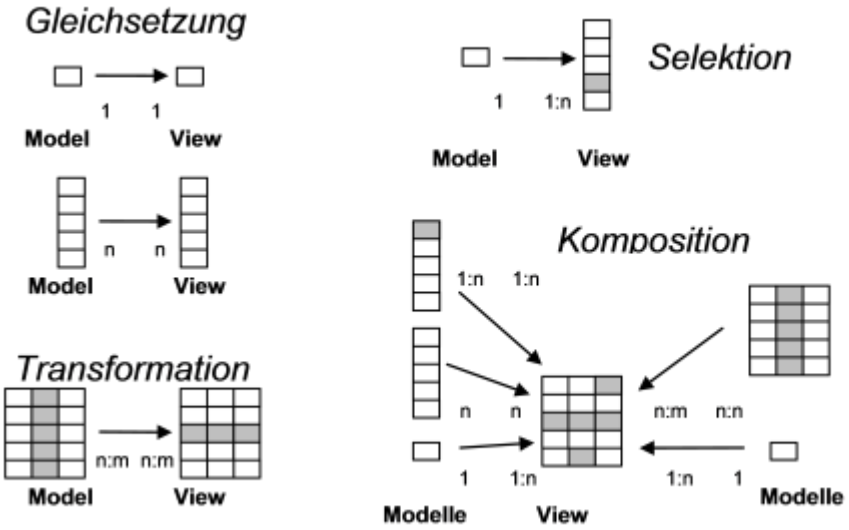


Abbildung 33: *Relationen zwischen Model und View*

Um alle diese Relationsarten zu ermöglichen dienen zwei Attribute:

```
anyvalue .dataindex[<View-Oder-Model-Attribut>] <Index>
attribute .datamap[<View-Attribut>] <View-Attribut>
```

Das *.dataindex[]*-Attribut dient dabei zur Selektion eines Wertes aus einer Werteliste sowie zur Transformation bei indizierten Werten. Dabei sollte beachtet werden, dass die Attribute als Skalar, Feld, assoziatives Feld oder Matrix (zweidimensionales Feld) vorkommen, allerdings der Zugriff immer nur einen Vektor (also ein eindimensionales Feld) ermöglicht.

Bei der Zuweisung von Sammlungen (Wertanzahl *<n>*) an ein vektorielles Attribut (eindimensional oder zweidimensional) finden dabei folgende Transformationen entsprechend dem Index statt:

Tabelle 17: *Indextransformationen bei der Zuweisung von Sammlungen*

Index	Zuweisung
[0,<col>]	*[1,<col>] … [<n>,<col>]*
[<row>,0]	*[<row>,1] … [<row>,<n>]*
[<row>,<col>]	*[<row>,<col>]*
0	*1 … <n>*
<row>	*<row>*
void	*void*
andere	*void*

Das *.datamap[]*-Attribut erlaubt die Vermischung von Datenwerten (aus einem oder verschiedenen Datenmodellen) die genau einem View-Attribut zugeordnet werden sollen. Das hauptsächliche Anwendungsgebiet wird für zweidimensionale Attribute sein, wie man sie am ***Tablefield***-Objekt findet. Man verwendet dazu ein „virtuelles" View-Attribut, welches dann mittels *.datamap[]*-Attribut einem „realen" View-Attribut zugeordnet wird. Als „virtuelles" Attribut kann ein beliebiges vordefiniertes oder benutzerdefiniertes Attribut verwendet werden, es empfiehlt sich unter Umständen aber die Nutzung eines am View-Objekt vorhandenen Attributs um die Reihenfolge bei der kompletten Repräsentation aller Model -Werte zu beeinflussen.

Beispiel

Folgendes Beispiel soll diese verschiedenen Relationsarten veranschaulichen. Beispielsweise wird das View-Objekt „PtMonth" mit dem gesamten Inhalt des Attributs „.MonthName[]" aus dem Model-Objekt „RecDate" gekoppelt. Hingegen zeigt das View-Objekt „StCurMonth" lediglich den 3. Eintrag aus dem Feld „RecDate.FullMonthName[]" an.

Weitaus komplexer ist die Datenkopplung für das View-Objekt „TfCurWeek". Die Wochentagsnamen aus „RecDate.DayName[7]" werden über die Kopplung `.dataindex[.content] [1,0];` auf die Titelzeile (Elemente *[1,1] … [1,7]*) verteilt. Um eine Kopplung der Wochennummern in die Tabelle zur erreichen, nutzen wir das *.field*-Attribut mittels der Definitionen `.dataget[.field] .WeekNr;` und `.dataindex[.field] [0,8];`. Da eine Kopfzeile mittels `.rowheader 1;` gesetzt ist, werden die Wochennummern in die Elemente *[2,8] … [6.8]* geschrieben. Für die Markierung des aktuellen Tages nutzen wird das „virtuelle" Attribut „.text", wobei eine Abbildung `.datamap[.text] .content;` eingerichtet werden muss, um den Markierungstext in das Feld „.content[4,3"] zu leiten.

```
dialog D
record RecDate
{
  integer CurMonth := 3;
  integer CurDay := 23;
  integer CurWeekDay := 3;
```

```
  string DayName[7];
  .DayName[1] := "Mon";
  ...
  string MonthName[12];
  .MonthName[1] := "Jan";
  ...
  string FullMonthName[12];
  .FullMonthName[1] := "January";
  ...
  integer WeekNr[5];
  .WeekNr[1] := 9;
  ...
}

window WiData
{
  .title "Datamodel: index-coupling";
  .width 600;  .height 400;
  .datamodel RecDate;

  poptext PtMonth
  {
    .xauto 0;
    .dataget .MonthName;  /* fills poptext.text[] with "Jan","Feb","Mar",...
*/
    .dataget[.userdata] .FullMonthName;  /* fill full names into .userdata[]
*/
    .dataget[.activeitem] .CurMonth;
  }

  statictext StCurMonth
  {
    .ytop 30;
    .xauto 0;
    .alignment 0;
    .dataget .FullMonthName;
    /* fills statictext.text with "Mar" by Model index */
    .dataindex[.FullMonthName] 3;
  }

  tablefield TfCurWeek
  {
    .xauto 0;  .yauto 0;
    .ytop 60;
    .rowheight[0] 30;  .colwidth[0] 60;
    .colcount 8;  .rowcount 6;
```

```
    .content[1,8] "Week";
    .rowheader 1;
    .dataget[.content] .DayName;
    .dataindex[.content] [1,0];  /* fill "Mon"... to first row via View index
*/
    .dataget[.field] .WeekNr;  /* map to .field attribute to fill week no.
*/
    .dataindex[.field] [0,8];  /* vertical into last column below the header
*/
    .datamodel[.text] Marker;  /* mark a specific day using the
*/
    .dataget[.text] .value;    /* .datamap attribute, because it goes into
*/
    .datamap[.text] .content;  /* the already used View attribute .content!
*/
    .dataindex[.text] [4,3];
  }

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

Das Bildschirmfoto dazu sieht folgendermaßen aus:

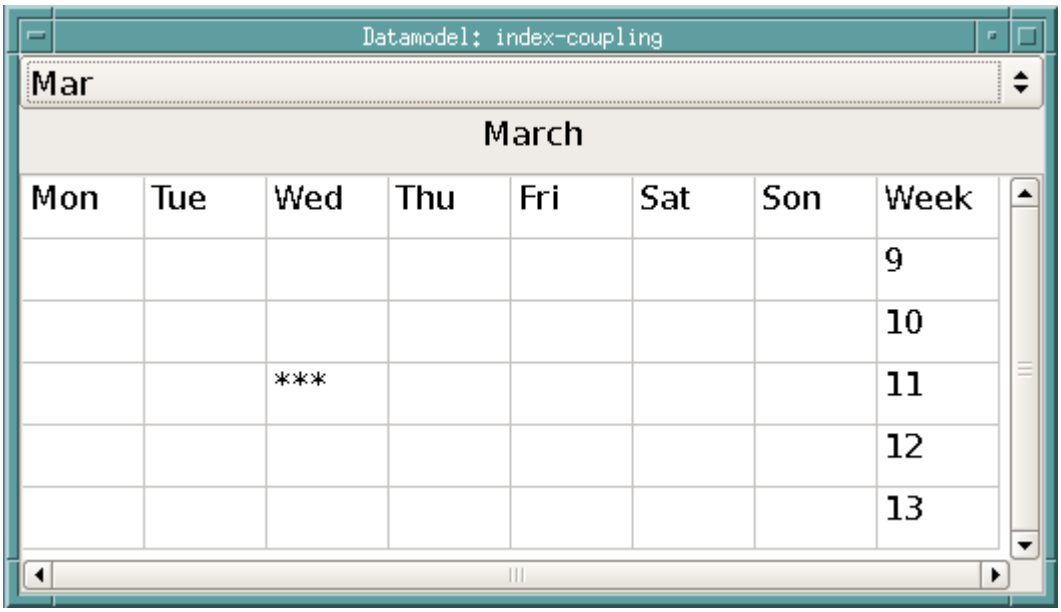


Abbildung 34: *Aus verschiedenen Datenmodellen befülltes Tablefield*

Sehr wichtig ist es folgenden Umstand zu verstehen:

Der Datenaustausch zwischen Model und View funktioniert über die bestehenden Mechanismen des IDM, also **setvalue()**, **getvalue()**, **setvector()** und **getvector()**. Dies impliziert, dass die ausgetauschten Datenwerte entweder skalar oder vektoriell sind.

Wird die Änderung eines Model-Attributs signalisiert (z.B. in einer eigenen Datenfunktion über den Aufruf der **DM_DataChanged()**-Funktion), so liefert diese normalerweise einen Model-Index als Hinweis auf das geänderte Detail mit (typischerweise vom Typ *void*, *integer* oder *index*). Da diese Änderungssignalisierung als *datachanged*-Ereignis vom IDM verarbeitet wird, werden mehrere Änderungen zusammengefasst zur Gesamtänderung (*void*-Index) bzw. überflüssige Ereignisse weggelassen.

Im Zusammenspiel mit der Beeinflussbarkeit der Werteaggregation über das *.dataindex[]*-Attribut hat dies mehrere Konsequenzen und Besonderheiten, die zu beachten sind:

1. Die komplette Befüllung eines zweidimensionalen Attributs (z.B. *.content[]* beim ***Tablefield***) ist problemlos möglich, wenn die Ausrichtung sowie die Spalten- und Zeilenzahl passend zur Datenlänge sind. Eine Anpassung der Tabellengröße erfolgt über die **setvector()**-Funktionalität.
   Die Aktualisierung von Einzelwerteänderungen kann aber nur an der korrekten Zelle im Tablefield erfolgen, wenn das Daten-Model eine Änderungsmeldung mit dem entsprechenden Index abgibt. Eine beliebige Übertragung eines zweidimensionalen Feldes in seiner Zeilen- und Spaltenform an eine bestimmte Stelle in einem Tablefield ist aber nicht möglich, lediglich die Signalisierung von Wertänderungen an einer beliebigen Stelle.
2. Durch die Nutzung von *.dataindex[]* am ***Tablefield*** kann ein skalarer oder vektorieller Datenwert an eine explizite Zelle, Zeile oder Spalte gebracht werden. Eine automatische Zellenerweiterung erfolgt nach den Regeln von **setvalue()** bzw. **setvector()** und hängt dementsprechend auch von der Indextransformation ab. Dabei ist zu beachten, dass benutzerdefinierte Attribute (z.B. Felder) keine automatische Erweiterung erlauben.
3. Analoges gilt auch für Vektor- und Matrixattribute wie sie an ***Listbox***, ***Treeview*** und ***Poptext*** zu finden sind.
4. Der Index für ein Model-Attribut wird genutzt um die unnötige Propagierung von Werteänderungen zu vermeiden (z.B. ist eine Änderung von „Data.Attr[5]“ für View-Objekte uninteressant, die nur mit „Data.Attr[2]“ gekoppelt sind).
5. Bei Nutzung von *void* (Standardwert für *.dataindex[]*) für die Indextransformation erfolgt die Weitergabe des Model-Index, signalisiert durch ein *datachanged*-Ereignis, je nach Attributtyp (skalar, vektoriell, zweidimensional) bis hin zur Präsentation durch **:represent()**. Genutzt unter anderem im Beispiel **randomcolors.dlg** um Farbwerte aus einem assoziativen Feld mit dem *.bgc[]*-Attribut eines ***Tablefield***-Objekts zu koppeln. Allerdings bedeutet dies auch, dass bei Kopplungen mit unterschiedlicher Wertedimension (z.B. ein Feld wird auf einen Skalar abgebildet) sich eine Werteaktualisierung entsprechend dem Index der letzten Änderung ergibt – was meist nicht überraschend ist. Abhilfe schafft hier die Nutzung des korrekten Index für das View- und das Model-Attribut mittels des *.dataindex[]*-Attributs.
6. Ein Setzen von Standardwerten (Index *[0]*, *[0,0]*, *[0,*]* oder *[*,0]*) der beteiligten Model- und View-Attribute ist nicht möglich.

Bei folgenden Beispielen (zu finden im Unterverzeichnis *examples/datamodel/* des IDM-Installationsverzeichnis) lohnt dabei ein Blick in den Quelltext wie auch in das Tracefile bei der Ausführung:

relations.dlg

Zeigt diverse Werteaggregationen von skalaren oder vektoriellen Datenwerten in ein ***Listbox***-Objekt mit und ohne Anpassung des *.content[]*-Felds sowie die Auswirkung bei Einzeländerungen.

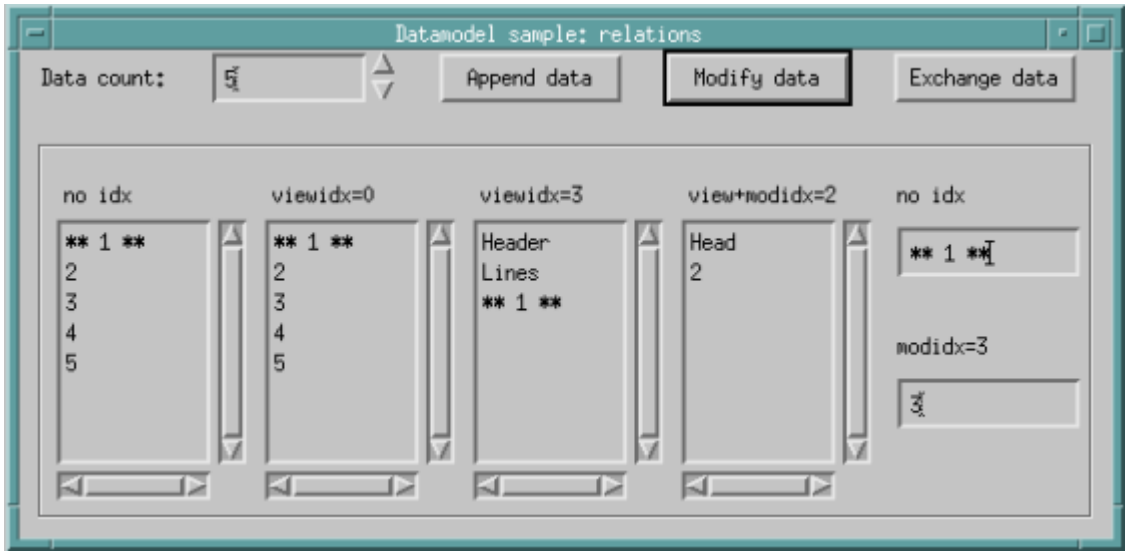


Abbildung 35: *Werteaggregationen für Listenobjekte, Beispiel relations.dlg*

matrixrel.dlg

Zeigt diverse einfache und komplexe Werteaggregationen von verschiedenen Datenwerten in ein ***Tablefield***-Objekt auf.

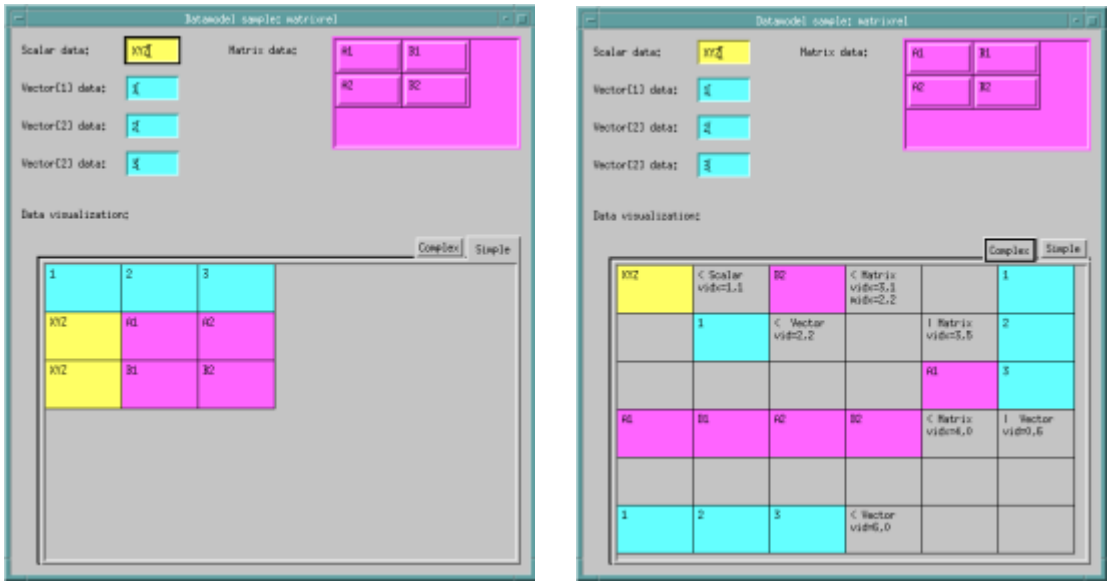


Abbildung 36: *Wertaggregationen für Tablefields, Beispiel matrixrel.dlg*

numbers.dlg

Zeigt die komplette Befüllung eines ***Tablefield***-Objekts mit einer Kopfzeile und inklusive automatischer Anpassung der Zeilenzahl auf.

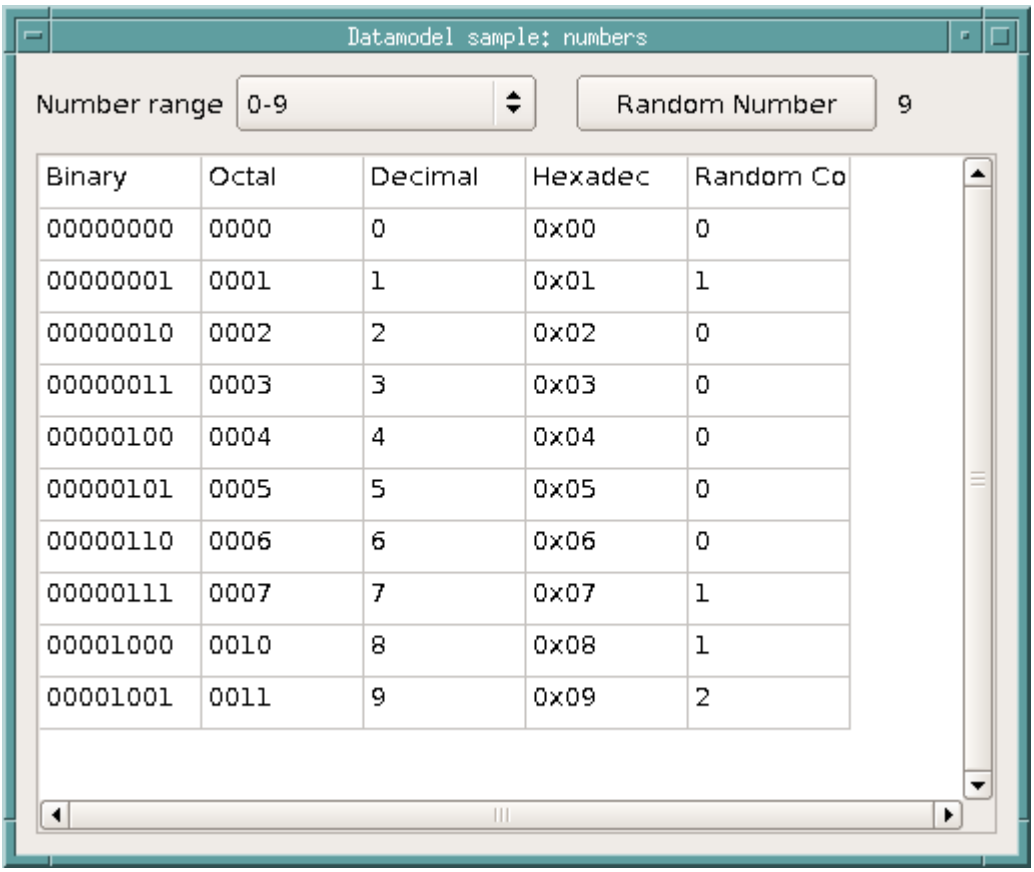


Abbildung 37: *Komplettes Befüllen eines Tablefields, Beispiel numbers.dlg*

puzzle.dlg

Zeigt die Befüllung eines ***Tablefields*** aus einem ***Record*** als Datenmodell oder einer Datenfunktion (lokal bzw. remote) auf.

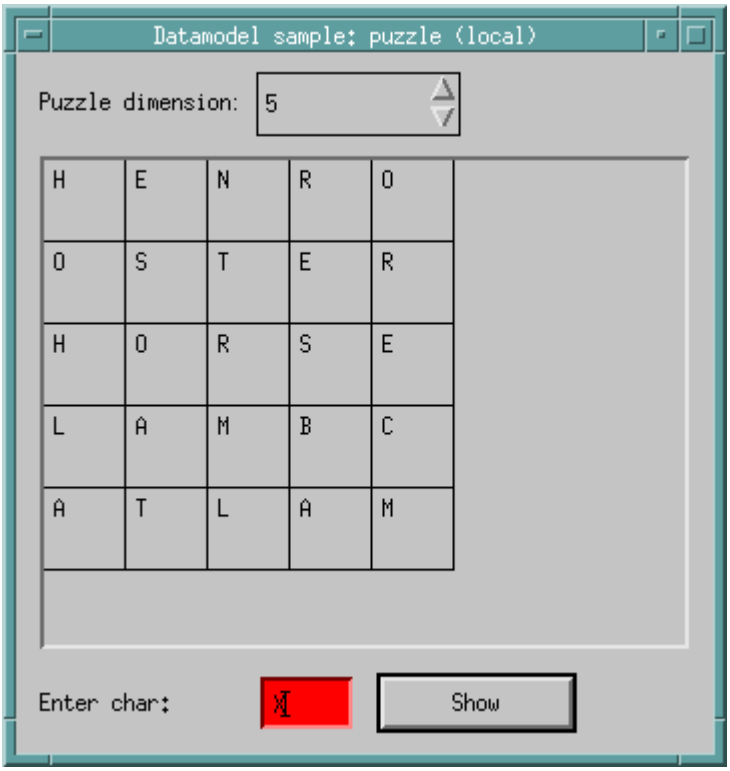


Abbildung 38: *Befüllung eines Tablefields aus Record und Datenfunktion, Beispiel puzzle.dlg*

33.6.5 Synchronisierung zwischen Model und View

Die automatische Synchronisierung zwischen Model und View wird über das *.dataoptions[]*-Attribut gesteuert.

Tabelle 18: *Optionen zur Synchronisierung zwischen Model und View*

Attribut-Index	Standard	Komponente	Bedeutung
dopt_represent_on_map	***true***	View	Direkt vor der Sichtbarschaltung werden von den Model-Komponenten die Datenwerte geholt und dem View-Objekt gesetzt.
dopt_represent_on_init	***false***	View	In der Objektinitialisierung (**:init**-Methode) werden die Datenwerte geholt und am View-Objekt gesetzt.
dopt_apply_on_unmap	***false***	View	Direkt vor der Unsichtbarschaltung werden die Datenwerte vom View-Objekt geholt und den gekoppelten Model-Komponenten zugewiesen.

Attribut-Index	Standard	Komponente	Bedeutung
dopt_apply_on_event	***false***	View	Wird durch eine Benutzerinteraktion ein Dialogereignis erzeugt, welches auf eine mögliche Änderung eines View-Attributs hinweist, so wird dieses den gekoppelten Model-Komponenten zugewiesen.
dopt_propagate_on_start	***true***	Model	Beim Start des Dialogs bzw. Moduls werden die Daten der Model-Objekte an die gekoppelten View-Komponenten weitergegeben.
dopt_propagate_on_changed	***true***	Model	Änderungen an einem Model-Attribut werden an die gekoppelten View-Komponenten weitergereicht.

Ein ganz wichtiges Merkmal des Datenmodells ist, dass Datenänderungen immer „asynchron“ über die Ereignisverarbeitung gemeldet und abgearbeitet werden. Ist z.B. das Datenmodell (Model-Objekt) ein ***Record*** bzw. ein anderes Objekt mit benutzerdefinierten oder vordefinierten Attributen, so wird eine Attributänderung, die über ein *datachanged*-Ereignis signalisiert wird, an die verbundenen View-Objekte gemeldet damit diese sich aktualisieren können. Der Dialogprogrammierer kann zwar über die Operation „::=“ ein *changed*-Ereignis unterbinden, niemals aber ein *datachanged*-Ereignis.

Eine globale Variable kann ebenso als Model-Komponente verwendet werden, erlaubt aber keine Steuerung der Synchronisation. Es erfolgt bei Variablen immer eine Wertepropagierung bei Start des Dialogs bzw. Moduls und bei Änderung des Variablenwertes.

Wird `.dataoptions[dopt_apply_on_unmap] := true;` an einer **Dialogbox**, ***Messagebox*** oder ***Filereq*** über einen **querybox**-Aufruf genutzt, so erfolgt die Übertragung der Datenwerte von der View zum Model nur bei einer positiven Bestätigung (*button_ok* oder *button_yes*).

Ein manueller Eingriff ist in den meisten Fällen nicht notwendig. Falls doch erforderlich, kann eine manuelle Synchronisation über die folgenden Methoden passieren:

Tabelle 19: *Manuell aufrufbare Datenmodellmethoden*

Methode	Komponente	Funktion
:represent()	View	Die Daten aller gekoppelten Model-Attribute (*.dataget*) werden geholt und dem View-Objekt zugewiesen
:apply()	View	Für alle gekoppelten View-Attribute (*.dataset*) werden die Daten geholt und den Models zugewiesen

Methode	Komponente	Funktion
:propagate()	Model	Das Model-Objekt propagiert eine Änderung aller seiner Model-Attribute an die View-Objekte, die eine Kopplung aufweisen
:collect()	Model	Das Model-Objekt holt sich alle Daten aus den gekoppelten View-Objekten um sie seinen Model-Attributen zuzuweisen

Eine Synchronisierung findet nur zwischen Instanzen statt, niemals mit Default-Objekten oder Modellen.

Für eine optimierte Synchronisation von Teilwerten gibt es noch eine weitere Besonderheit. Wird eine Wertänderung signalisiert so wird der Index dieser Wertänderung weitergeleitet sodass nur dieser Einzelwert von der View-Komponente zu aktualisieren ist.

Beispiel

Über ein ***Timer***-Objekt werden zufällige Farben ausgewählt und zufälligen Zellen einer Tabelle zugeordnet.

```
dialog D
color CoRed "red";
color CoYellow "yellow";
color CoGreen "green";
color CoBlue "blue";

timer TiRandomColors {
  .active true;
  .starttime "+00:00:00";
  .incrtime "+00:00:00'100";
  object Color[index] := null;

  on select
  {
    variable index RandIndex := [1 + random(5),1 + random(5)];
    variable object RandColor := D.color[1 + random(D.count[.color])];
    /* change the color at a specific index */
    this.Color[RandIndex] := RandColor;
  }
}

window Wi
 {
  .title "Datamodel - random colors";
  .width 400;  .height 400;

  tablefield Tf
  {
```

```
    .xauto 0;  .yauto 0;
    .rowcount 5;  .colcount 5;
    .colwidth[0] 60;  .rowheight[0] 30;
    .dataoptions[dopt_represent_on_map] false;  /* avoid initial :represent()
*/
    .datamodel TiRandomColors;
    .dataget[.bgc] .Color;
  }
}
```

Bildschirmfoto dazu:

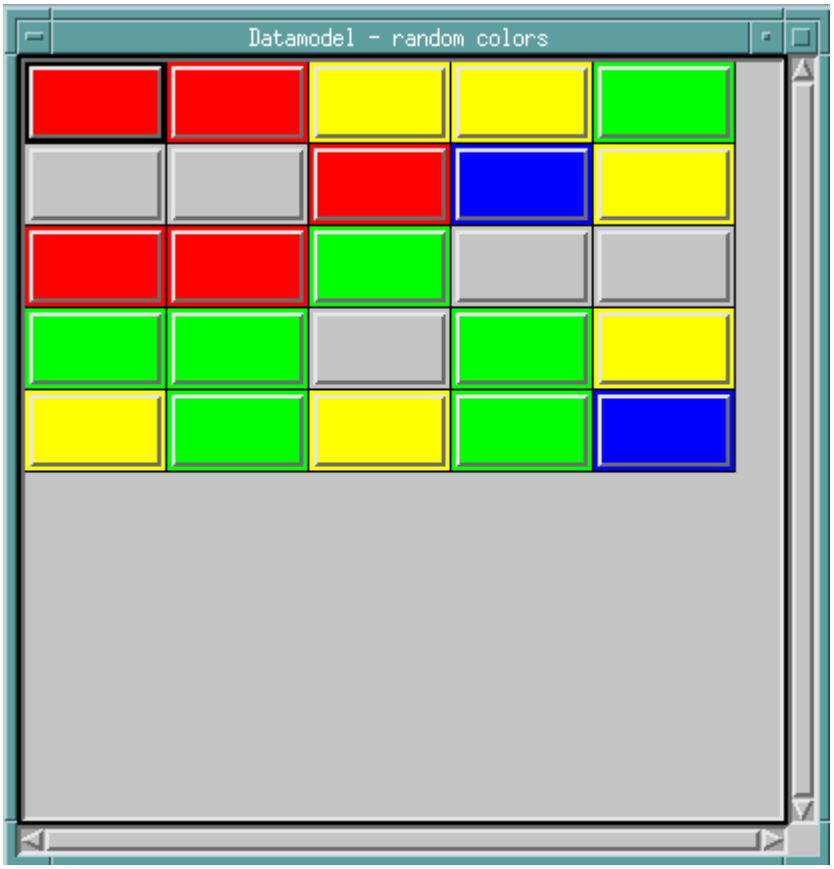


Abbildung 39: *Zufällig gefärbte Tablefield-Zellen mit einem Timer als Datenmodell*

33.6.6 Konvertierung und Konvertierungsmethoden

Bei der Synchronisation zwischen Model- und View-Komponente werden Datenwerte ausgetauscht, was meist einem Setzen oder Erfragen von Attributen (analog zu **setvalue()**, **getvalue()**, **setvector()**, **getvector()**) entspricht. Ein Eingriff des Anwendungsprogrammierers durch Überdefinition der Methoden **:set()** oder **:get()** ist dabei **nicht** möglich.

Allerdings gibt es für die View-Komponente die Möglichkeit, über die überdefinierbaren Methoden **:represent(<Value>, <Attribute>, <Index>)** und **:retrieve(<Attribute>, <Index>)** Einfluss zu nehmen.

Beim Setzen von Attributen an der View-Komponente wird normalerweise, in der Standardbehandlung durch den IDM, eine Typkonvertierung in den Zieldatentyp durchgeführt. Im nachfolgenden Beispiel wird aufgezeigt, wie durch Überdefinition eine Sonderbehandlung für das View-Attribut *.activeitem* durchgeführt wird.

Beispiel

Eine Flughafenliste wird in der ***Listbox*** „Lb“ angezeigt. Die überdefinierten Methoden **:represent()** und **:retrieve()** sorgen dabei für eine korrekte Umsetzung von und zu einem IATA-Kürzel.

```
dialog D
record RecAirport
{
  string Name[4];
  string IATA[4];
  .Name[1] := "Hartsfield Jackson Atlanta International";
  .IATA[1] := "ATL";
  .Name[2] := "Dubai International";
  .IATA[2] := "DXB";
  .Name[3] := "Stuttgart";
  .IATA[3] := "STR";
  .Name[4] := "Orkney Islands";
  .IATA[4] := "KOI";
  string NearestIATA := "STR";
  string Selected := "";
}

window Wi
{
  .title "Airports";
  .width 200;  .height 200;

  listbox Lb
  {
    .xauto 0;  .yauto 0;
    .ybottom 30;
    .dataoptions[dopt_apply_on_event] true;
    .datamodel RecAirport;
    .dataget .Name;
    .dataget[.userdata] .IATA;
    .dataget[.activeitem] .NearestIATA;
    .dataset[.activeitem] .Selected;

    :represent()
    {
      if Attribute = .activeitem then
        Value := this:find(.userdata, Value);
      endif
```

```
      pass this:super();
    }

    :retrieve()
    {
      if Attribute = .activeitem then
        return this.userdata[this.activeitem];
      endif
      pass this:super();
    }
  }

  edittext Et
  {
    .yauto -1;  .xauto 0;
    .editable false;
    .datamodel RecAirport;
    .dataget .Selected;
  }

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

Bildschirmfoto dazu:



Abbildung 40: *Fenster des Beispieldialogs zum Überschreiben von* ***:represent()*** *und* ***:retrieve()***

33.6.7 Aktionen

Um Manipulationsfunktionen der Model-Komponente allgemein für die Presenter-Logik zugänglich zu machen, dient das Konzept der Aktionen. Dies sind Methoden mit Parametern, welche von den Datenmodellen zur Verfügung gestellt werden können. Aufrufbar sind diese über die **:calldata()**-Methode.

33.6.8 Ablaufverfolgung

Folgende Trace-Codes wurden zusätzlich eingeführt um die Synchronisationsabläufe zwischen Datenmodellen (Models) und Präsentationsobjekten (Views) sichtbar zu machen und um mögliche Anwendungsfehler aufzudecken.

Trace-Code	Bedeutung
[CD]	„Call Data“ um den Aufruf der Synchronisation zwischen View und Model zu verfolgen. Tracing erfolgt typischerweise für **:get()**und **:set()** am Model-Objekt sowie **:represent()** und **:apply()** am View-Objekt.
[CE]	„Data Changed Event“ – Ereignis das eine Änderung am Datenmodell signalisiert, typischerweise durch das Setzen eines Attributs oder durch den Aufruf der Applikationsfunktion **DM_DataChanged()** ausgelöst.

33.6.9 Restriktionen

Von IDM abgelehnt werden Änderungen an der Datenkopplung oder der Steuerung der Synchronisation (Attribute *.datamodel*, *.dataget*, *.dataset*, *.datamap*, *.dataindex*, *.dataoptions*) während man innerhalb einer überdefinierbare Methode **:represent()** oder **:retrieve()** bzw. eines Aufrufs einer Datenfunktion ist.

Ebenso vom IDM abgelehnt werden Änderungen an den Attributen *.visible* oder *.mapped*, wenn man diese innerhalb der Synchronisation der View-Komponente ausführt und der IDM sich gerade bei der Umsetzung der Sichtbar- oder Unsichtbarkeitsschaltung befindet.

33.6.10 Attributklassifikation und Datenmodellattribute

Die neuen Attribute *.datamodel*, *.dataget*, *.dataset*, *.dataindex*, *.datamap* und *.dataoptions* definieren eine Klasse von Attributen, welche die Kopplung der View- und Model-Komponenten festlegen und die Synchronisation steuern.

33.6.10.1 .datamodel

	Bezeichner	Datentyp
Regelsprache	**.datamodel[attribute]**	object
C	AT_datamodel	DT_object
COBOL	AT-datamodel	DT-object
Klassifizierung	Datenmodellattribut	

	Bezeichner		Datentyp
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Mit diesem Attribut kann die Kopplung des Objekts (View-Komponente) zu einem Datenmodell (Model-Komponente) definiert werden. Das Attribut ist an der View-Komponente zu definieren.

Als Index kann ein Attribut verwendet werden, welches das View-Attribut bezeichnet für das das Datenmodell verwendet werden soll. Ohne Index gilt das Datenmodell für alle View-Attribute.

Für eine aktive Kopplung ist mindesten die Definition eines Datenmodells ohne Index sowie eine Kopplung des View-Attributs mit einem Model-Attribut über *.dataget* oder *.dataset* nötig.

Vorrang vor der normalen Vererbung des Attributwertes hat noch die Überdeckung durch einen gesetzten Wert an dem am nächsten liegenden Vaterobjekt (ohne Dialog bzw. Modul).

33.6.10.2 .dataget

	Bezeichner		Datentyp
Regelsprache	**.dataget[attribute]**		attribute
C	AT_dataget		DT_attribute
COBOL	AT-dataget		DT-attribute
Klassifizierung	Datenmodellattribut		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Mit diesem Attribut wird die Kopplung eines View-Attributs, anzugeben als Index, zu einem Model-Attribut, anzugeben als Wert, für das Holen des Wertes von der Model-Komponente und das Repräsentieren durch die View-Komponente definiert.

Ohne Index wird das klassenspezifische Standardattribut (siehe Tabelle im Kapitel „Kopplung zwischen Model und View“) verwendet. Das Attribut ist an der View-Komponente zu definieren.

Der Datenaustausch erfolgt entsprechend den Synchronisationsregeln der beteiligten Model-Komponente und View-Komponente und ist nur aktiv, wenn auch ein Datenmodell im .datamodel-Attribut gesetzt ist.

33.6.10.3 .dataset

	Bezeichner		Datentyp
Regelsprache	**.dataset[attribute]**		attribute
C	AT_dataset		DT_attribute
COBOL	AT-dataset		DT-attribute
Klassifizierung	Datenmodellattribut		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Mit diesem Attribut wird die Kopplung eines View-Attributs, anzugeben als Index, zu einem Model-Attribut, anzugeben als Wert, für das Holen des Wertes von der View-Komponente und das Zuweisen an der Model-Komponente definiert.

Ohne Index wird das klassenspezifische Standardattribut (siehe Tabelle im Kapitel „Kopplung zwischen Model und View“) verwendet. Das Attribut ist an der View-Komponente zu definieren.

Der Datenaustausch erfolgt entsprechend den Synchronisationsregeln der beteiligten Model-Komponente und View-Komponente und ist nur aktiv, wenn auch ein Datenmodell im .datamodel-Attribut gesetzt ist.

33.6.10.4 .dataindex

	Bezeichner		Datentyp
Regelsprache	**.dataindex[attribute]**		attribute
C	AT_dataindex		DT_attribute
COBOL	AT-dataindex		DT-attribute
Klassifizierung	Datenmodellattribut		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Dieses Attribut definiert zusätzlich zur Kopplung von Attributen zwischen Model-Komponente und View-Komponente einen Index der den Zugriff auf das Attribut weiter verfeinert um so die unter-

schiedlichen Relationsarten (siehe hierzu Kapitel „Reihenfolge und Werteaggregation“) zu ermöglichen.

Als Index kann ein beliebiges View- oder Model-Attribut verwendet werden. Dieses Attribut ist an der View-Komponente zu setzen.

33.6.10.5 .datamap

	Bezeichner		Datentyp
Regelsprache	**.datamap[attribute]**		attribute
C	AT_datamap		DT_attribute
COBOL	AT-datamap		DT-attribute
Klassifizierung	Datenmodellattribut		
Index-Datentyp	attribute		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Dieses Attribut definiert eine zusätzliche Abbildung von Attributen bei der Kopplung von Attributen zwischen View- und Model-Komponente. Diese Definition kann notwendig werden, wenn mehrere Datenmodelle ihre Datenwerte in ein einziges Attribut der View-Komponente schreiben sollen um so die Relationsart „Mischung“ (siehe hierzu auch Kapitel „Reihenfolge und Werteaggregation“) zu erlauben.

Als Index kann ein beliebiges View- oder Model-Attribut verwendet werden. Dieses Attribut ist an der View-Komponente zu setzen.

33.6.10.6 .dataoptions

	Bezeichner		Datentyp
Regelsprache	**.dataoptions[enum]**		boolean
C	AT_dataoptions		DT_boolean
COBOL	AT-dataoptions		DT-boolean
Klassifizierung	Datenmodellattribut		
Index-Datentyp	enum		
Zugriff	get, set	Standardwert	–
changed-Ereignis	nein	Vererbung	ja

Dieses Attribut definiert die automatische Synchronisation zwischen Model- und View-Komponente. Für Details siehe Kapitel „Synchronisierung zwischen Model und View“.

Attribut-Index	Standard	Komponente	Bedeutung
dopt_represent_on_map	***true***	View	Direkt vor der Sichtbarschaltung werden von den Model-Komponenten die Datenwerte geholt und dem View-Objekt gesetzt.
dopt_represent_on_init	***false***	View	In der Objektinitialisierung (**:init**-Methode) werden die Datenwerte geholt und am View-Objekt gesetzt.
dopt_apply_on_unmap	***false***	View	Direkt vor der Unsichtbarschaltung werden die Datenwerte vom View-Objekt geholt und den gekoppelten Model-Komponenten zugewiesen.
dopt_apply_on_event	***false***	View	Wird durch eine Benutzerinteraktion ein Dialogereignis erzeugt, welches auf eine mögliche Änderung eines View-Attributs hinweist, so wird dieses den gekoppelten Model-Komponenten zugewiesen.
dopt_propagate_on_start	***true***	Model	Beim Start des Dialogs bzw. Moduls werden die Daten der Model-Objekte an die gekoppelten View-Komponenten weitergegeben.
dopt_propagate_on_changed	***true***	Model	Änderungen an einem Model-Attribut werden an die gekoppelten View-Komponenten weitergereicht.

Dieses Attribut ist an allen Objektklassen vorhanden, die benutzerdefinierte Attribute erlauben, aber unter Umständen nicht mit allen Indizes. Nur für visuelle Objekte stehen die Indizes *dopt_apply_on_unmap*, *dopt_represent_on_map* und *dopt_apply_on_event* zur Verfügung.

33.6.11 Datenmodellmethoden

33.6.11.1 :apply()

Beim Aufruf dieser Methode an einer View-Komponente werden alle Datenwerte von View-Attributen, die mit Model-Komponenten gekoppelt sind, ausgelesen (durch einen **:retrieve()**-Aufruf) und den korrespondierenden Model-Komponenten zugewiesen.

Durch Angabe der optionalen Parameter kann auch die Einschränkung auf ein spezielles View-Attribut bzw. auf einen speziellen indizierten Einzelwert erfolgen. Ohne Parameter erfolgt auch ein **:apply ()**-Aufruf an allen Kindern und Kindeskindern.

Normalerweise sollte ein Aufruf dieser Methode nicht notwendig werden, da die Synchronisation zwischen View und Model über das Attribut *.dataoptions[]* gesteuert wird.

Die Kopplung zur Model-Komponente muss über die Attribute *.datamodel* und *.dataset* erfolgt sein.

Es erfolgt keine Fehlermeldung, wenn das Objekt keine gekoppelten Attribute besitzt oder das optional angegebene Attribut nicht existiert oder nicht gekoppelt ist.

Definition

```
void :apply
(
  { attribute Attribute input { , anyvalue  Index input } } |
  { anyvalue  Index     input }
)
```

Parameter

*attribute **Attribute** input*

Dieser optionale Parameter bezeichnet das View-Attribut welches ausgelesen und der zugehörigen Model-Komponente zugewiesen werden soll.

*anyvalue **Index** input*

Dieser optionale Parameter bezeichnet den Indexwert der für die Werteholung vom View-Objekt zu verwenden ist.

Beispiel

Im folgenden Beispieldialog erfolgt die Zuweisung des Inhaltsstrings ans Datenmodell über die Nutzung der **:apply()**-Methode am ***Edittext*** oder am ***Fenster***. Der ***Statictext*** am unteren Fensterrand zeigt den aktuellen Wert des Datenmodells „VarString“ an.

```
dialog D
accelerator AcF5 "F5";
variable string VarString := "Freitag";

window Wi
{
  .title ":apply demo";
  .width 200;

  child edittext Et
  {
    .xauto 0;
    .xright 80;
    .datamodel VarString;
```

```
    .dataset .value;
    .content "Samstag";
    .toolhelp "Press F5 to apply";

    on key AcF5
    {
      this:apply(.content);
    }
  }

  child pushbutton Pb
  {
    .xauto -1;
    .width 80;
    .text "Apply";

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

  child statictext St
  {
    .xauto 0;
    .yauto -1;
    .datamodel VarString;
    .dataget .value;
  }

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

33.6.11.2 :collect()

Beim Aufruf dieser Methode an einer Model-Komponente werden alle Datenwerte von den View-Komponenten erfragt, die gekoppelte Attribute aufweisen. Dazu erfolgt ein **:retrieve()**-Aufruf für die korrespondierenden View-Attribute um die Datenwerte zu ermitteln und sie dem Datenmodell zuzuweisen.

Normalerweise sollte ein Aufruf dieser Methode nicht notwendig werden, da die Synchronisation zwischen View und Model über das Attribut *.dataoptions[]* gesteuert wird.

Die Kopplung zur Model-Komponente muss über die Attribute *.datamodel* und *.dataset* erfolgt sein.

Es erfolgt keine Fehlermeldung wenn das Objekt keine Referenzierung durch eine View-Komponente oder keine gekoppelten Attribute besitzt.

Definition

```
void :collect
(
)
```

Beispiel

```
dialog D
record Rec
{
  .dataoptions[dopt_propagate_on_start] false;
  string  FirstName := "";
  string  LastName  := "";
  boolean Female    := false;
}

timer Ti
{
  .starttime "+00:00:02";
  .incrtime "+00:00:01";
  .active true;

  on select
  {
    Rec:collect();  /* read data explictly (from EtFirst/LastName&CbFemale)
*/
  }
}

window Wi
{
  .title ":collect demo";
  .width 200;  .height 200;
  .xleft 250;
  .datamodel Rec;
  .dataoptions[dopt_represent_on_map] false;

  statictext
  {
    .text "First Name"; .width 100;
  }

  edittext EtFirstName
  {
    .content "Ina";
    .xauto 0;
    .xleft 100;
```

```
    .dataset .FirstName;
  }

  statictext
 {
    .text "Last Name";  .width 100;  .ytop 30;
  }

  edittext EtLastName
  {
    .ytop 30;
    .xauto 0;  .xleft 100;
    .content "Bausch";
    .dataset .LastName;
  }

  checkbox CbFemale
  {
    .ytop 60;
    .text "Female";
    .active true;
  }

  statusbar Sb
  {
    statictext StFirstName
    {
      .width 100;
      .dataget .FirstName;
    }

    statictext StLastName
    {
      .dataget .LastName;
      .text "Update in 2 secs";
    }
  }

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

33.6.11.3 :propagate()

Beim Aufruf dieser Methode an einer Model-Komponente werden alle gekoppelten View-Komponenten dazu aufgefordert, ihre View-Attribute zu aktualisieren, die mit diesem Model-Objekt gekoppelt sind.

Normalerweise sollte ein Aufruf dieser Methode nicht notwendig werden, da die Synchronisation zwischen View und Model über das Attribut *.dataoptions[]* gesteuert wird.

Die Kopplung muss über die Attribute *.datamodel* und *.dataget* an der View-Komponente erfolgt sein.

Es erfolgt keine Fehlermeldung wenn das Model-Objekt keine gekoppelten Attribute besitzt.

Definition

```
void :propagate
(
)
```

Beispiel

```
dialog D

timer Ti
{
  .starttime "+00:00:03";
  .incrtime "+00:00:01";
  .active true;
  string Time := "??:??";
  integer Min:=0;
  integer Sec:=0;
  integer Counter := 100;
  .dataoptions[dopt_propagate_on_start] false;
  .dataoptions[dopt_propagate_on_changed] false;

  on select
  {
    this.Counter := this.Counter-1;
    if this.Counter<0 then
      exit();
      return;
    endif
    if this.Sec=59 then
      this.Min := (this.Min + 1) % 60;
    endif
    this.Sec := (this.Sec + 1) % 60;
    this.Time := sprintf("%02d:%02d", this.Min, this.Sec);
    this:propagate();  /* propagate explicitly every second */
  }
}

window WiTime
{
  .width 200;
  .title ":propagate demo";
```

```
  .datamodel Ti;
  .dataoptions[dopt_represent_on_map] false;

  statictext StMin
  {
    .xauto 0;
    .dataget .Time;
  }

  statictext StCounter
  {
    .xauto 0;
    .ytop 50;
    .dataget .Counter;
    .text "Please wait...";

    :represent()
    {
      if Attribute=.text then
        Value := "Time to exit: "+Value;
      endif
      return this:super();
    }
  }

  on close { exit(); }
}

window WiMinSec
{
  .title ":propagate demo - min/sec";
  .width 200;  .height 200;
  .xleft 250;

  tablefield Tf
  {
    .xauto 0;  .yauto 0;
    .rowcount 2;  .colcount 2;
    .rowheader 1;
    .colwidth[0] 50;  .rowheight[0] 30;
    .content[1,1] "Min";
    .content[1,2] "Sec";
    .dataoptions[dopt_represent_on_map] false;
    .datamodel Ti;
    .dataget[.field] .Sec;
    .dataget[.content] .Min;
    .dataindex[.field] [1,2];
```

```
    .dataindex[.content] [2,1];
  }

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

33.6.11.4 :represent()

Diese Methode hat zweierlei Funktion.

1. Beim Aufruf ohne Parameter werden an der View-Komponente alle gekoppelten Attribute aktualisiert. Dies wird ebenso für alle Kinder und Kindeskinder ausgeführt.
   Durchgeführt wird dies, indem für jedes View-Attribut der Datenwert von der korrespondierenden Model-Komponente erfragt wird und durch Aufruf der internen und überdefinierbaren **:represent ()**-Methode (mit Parametern *Value*, *Attribute*, *Index*) die Repräsentation durch das Objekt erfolgt. Die Kopplung zur Model-Komponente muss über die Attribute *.datamodel* und *.dataset* erfolgt sein.

   Wichtig
   Ein direkter Aufruf mit Parametern wird vom IDM nicht unterstützt und ist für zukünftige Erweiterungen vorbehalten.
2. Die überdefinierbare **:represent()**-Methode erlaubt einen Eingriff des Dialogprogrammierers vor der eigentlichen Zuweisung des Datenwertes an das Präsentationsobjekt. Dadurch können notwendige Konvertierungen oder Transformationen zwischengeschaltet werden. Die überdefinierte Methode sollte mit `pass this:super();` abgeschlossen werden um das normale Verhalten des IDM zu gewährleisten bzw. um die automatische Typkonvertierung auszunutzen.

   Hinweis
   Die überdefinierbare Methode **:set()** wird bei der Zuweisung von Datenwerten an das Präsentationsobjekt nicht aufgerufen.

Definition

```
void :represent
(
)
```

Überdefinierbare Methode

```
void :represent
(
  anyvalue  Value     input,
  attribute Attribute input,
  anyvalue  Index     input
)
```

Parameter

anyvalue *Value* input

Dieser Parameter beinhaltet den Datenwert der präsentiert werden soll.

attribute *Attribute* input

Dieser Parameter bezeichnet das View-Attribut in dem der Datenwert repräsentiert werden soll.

anyvaluet *Index* input

Dieser Parameter bezeichnet den Indexwert für die Präsentation des Datenwertes im View-Attribut.

Beispiel

```
dialog D
default statictext { .sensitive false; }

record Rec
{
  .dataoptions[dopt_propagate_on_start] false;
  string Name := "Hugo Walter";
  string ZIP  := "O7123";
}

window Wi
{
  .title ":represent demo";
  .width 200;  .height 150;
  .datamodel Rec;
  .dataoptions[dopt_represent_on_map] false;

  statictext
  { .text "Name"; .width 100; }

  edittext EtName
  {
    .xauto 0;  .xleft 100;
    .dataget .Name;
    .dataset .Name;

    on deselect_enter
    {
      this:apply();
    }
  }

  statictext
  { .text "ZIP";  .width 100;  .ytop 30; }
```

```
edittext EtZIP
{
  .ytop 30;  .xauto 0;  .xleft 100;
  .dataget .ZIP;
  .dataset .ZIP;
  .format "XNNN";

  on deselect_enter
  {
    this:apply();
  }
}

statictext
{ .text "Country";  .width 100;  .ytop 60; }

poptext PtCountry
{
  .ytop 60;
  .xauto 0;  .xleft 100;
  .dataget[.activeitem] .ZIP;
  .dataset[.activeitem] .ZIP;
  .text[1]  "???";  .text[2]  "NRW";
  .text[3]  "BER";  .text[4]  "HH";
  .text[5]  "HRO";  .text[6]  "HAN";
  .text[7]  "NS";   .text[8]  "SH";
  .text[9]  "BW";   .text[10] "S";
  .text[11] "BAY";

  :represent()
  {
    /* convert the XNNN-string into an index */
    if Attribute=.activeitem then
      Value := 2 + atoi("0" + substring("" + Value, 2, 1));
    endif
    pass this:super();
  }

  :retrieve()
  {
    if Attribute = .activeitem then
      if this.activeitem > 1 then
        return sprintf("%s%d%02d",
                       substring(this.text[this.activeitem], 1, 1),
                       this.activeitem - 2,
                       random(100));
      else
```

```
        return "";
      endif
    endif
    pass this:super();
  }

  on select
  {
    this:apply();
  }
}

pushbutton PbRepresent
{
  .text "Represent";
  .yauto -1;
  .width 100;  .xleft 50;

  on select
  {
    this.window:represent();  /* explicit represent of all views */
  }
}

statusbar Sb
{
  statictext StInfo
  {
    .dataget .Name;
    .dataget[.At] .ZIP;
    string Name := "";
    string ZIP := "";

    :represent()
    {
      /* merge .Text & .At into a single string */
      case Attribute
      in .text:
        this.Name := toupper(Value);
      in .At:
        this.ZIP := Value;
      in .text, .At:
        this.text := sprintf("%s @ %s", this.Name, this.ZIP);
        return;
      endcase
      pass this:super();
    }
```

```
    }
  }

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

33.6.11.5 :retrieve()

Diese überdefinierbare Methode wird bei der Synchronisation zwischen View- und Model-Komponten am Präsentationsobjekt aufgerufen um den Datenwert für ein View-Attribut zu ermitteln. Dies geschieht auch implizit bei Aufruf der Methoden **:apply()** oder **:collect()**.

Wichtig

Ein direkter Aufruf wird von IDM nicht unterstützt und ist für zukünftige Erweiterungen vorbehalten.

Das Standardverhalten der Methode ist, dass das entsprechende Attribut als Einzelwert oder aggregierter Gesamtwert zurückgeliefert wird. Dabei erfolgt für die Ermittlung der Datenwerte des Präsentationsobjekts kein Aufruf der überdefinierbaren Methode **:get()**. Es werden also immer die konkreten Attributwerte im Standardverhalten erfragt. Indizierte Attribute liefern dabei beim Zugriff auf den Gesamtwert den Datenwert als Vektor zurück (siehe auch die Funktionen **setvector()** und **getvector()**).

Durch Überdefinition dieser Methode kann das Standardverhalten umgangen bzw. ergänzt werden, um so eine Anpassung der Werte zu erlauben.

Die Kopplung zur Model-Komponente muss über die Attribute *.datamodel* und *.dataset* erfolgt sein.

Definition

```
anyvalue :retrieve
(
  attribute Attribute input,
  anyvalue  Index     input
)
```

Parameter

attribute *Attribute* input
: Dieser Parameter bezeichnet das View-Attribut dessen Datenwert erfragt werden soll.

anyvalue *Index* input
: Dieser Parameter bezeichnet den Indexwert, der für den Zugriff auf den Datenwert des View-Attributs am Präsentationsobjekt benutzt werden soll.

Rückgabewert

Datenwert entsprechend der Attribut- und Indexangabe.

Beispiel

Siehe Methode **:represent()**

33.6.11.6 :calldata()

Beim Aufruf dieser Methode werden alle Datenmodelle, die am aufgerufenen Objekt und seinen Kindern über das *.datamodel*-Attribut definiert sind, aufgesammelt und eine „Aktion“, für jedes unterschiedliche Datenmodell jeweils nur einmal, in Form einer Methode mit Argumenten aufgerufen.

Mit dieser Methode kann also ohne die definierten Model-Objekte im Detail zu kennen an einem View-Objekt eine „Aktion“ an den beteiligten Datenmodellen ausgelöst werden.

Ein Rückgabewert oder eine Rückkopplung, falls z.B. die Methode an den Datenmodellen nicht vorhanden ist, ist nicht vorgesehen.

Definition

```
void :calldata
(
      method   Method input
  { , anyvalue Arg1   input }
  ...
  { , anyvalue Arg15  input }
)
```

Parameter

method *Method* input
Dieser Parameter bezeichnet die Methode, die an den beteiligten Datenmodellen aufzurufen ist.

*anyvalue **Arg1** input*

…

*anyvalue **Arg15** input*
Diese optionalen Parameter beinhalten die Argumente, die für den Methodenaufruf benutzt werden.

Beispiel

```
dialog D

record RecPart
{
  string Name[5] := "";
  .Name[1] := "car";
  .Name[2] := "wheel";
  .Name[3] := "seat";
  .Name[4] := "front pane";
  .Name[5] := "break pedal";
```

```
  rule void Add()
  {
    variable integer Idx := this.count[.Name] + 1;
    this.count[.Name] := Idx;
    this.Name[Idx] := "part#" + Idx;
  }
}

record RecLevel
{
  integer Level[5] := 0;

  rule void Add()
  {
    this.count[.Level] := this.count[.Level] + 1;
  }

  rule void Indent(integer Idx)
  {
    if Idx>0 andthen Idx<=this.count[.Level] then
      this.Level[Idx] := (this.Level[Idx]+1) % 5;
    endif
  }
}

window Wi
{
  .height 200;
  .title ":calldata demo";
  .datamodel RecPart;

  treeview Tv
  {
    .xauto 0;
    .yauto 0; .ybottom 30;
    .datamodel[.level] RecLevel;
    .dataget[.content] .Name;
    .dataget[.level] .Level;
    .open[0] true;
  }

  pushbutton PbIndent
  {
    .yauto -1;
    .text "Indent >";
```

```
    on select
    {
      this.window:calldata(:Indent, Tv.activeitem);
    }
  }

  pushbutton PbAdd
  {
    .xauto -1;  .yauto -1;
    .text "Add";

    on select
    {
      this.window:calldata(:Add);
    }
  }

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

33.6.12 Datenfunktion

33.6.12.1 Regelsprache

Definition

```
{ export | reexport } function datafunc { <Sprache> } <Funktionsname> ();
```

Die Parameter einer Datenfunktion sind vom ISA Dialog Manager fest definiert und können vom Benutzer nicht verändert werden.

Diese Funktionen können im *.datamodel*-Attribut bei allen Objekten angegeben werden, die auch benutzerdefinierte Attribute unterstützen. Eine Datenfunktion kann dabei die Rolle eines Datenmodells (Model-Komponente) erfüllen um die Präsentationsobjekte mit Datenwerten zu versorgen. Aufgerufen wird die Funktion wenn eine Synchronisierung der Datenwerte erfolgen soll, also entweder um Datenwerte vom Datenmodell zu holen oder um sie zuzuweisen.

Die Datenfunktion kann nur in C/C++ geschrieben werden und wird für die Applikationssprache COBOL nicht unterstützt. Die Implementierung der Datenfunktion kann aber auf der DDM-Serverseite liegen.

Vergleichen Sie hierzu bitte auch das Kapitel „C-Schnittstelle“ sowie die Attribute *.datamodel*, *.dataget*, *.dataset* und *.dataoptions*.

33.6.12.2 C-Schnittstelle

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

```
DM_Boolean DML_default DML_CALLBACK <Funktionsname> (DM_DataArgs *args);
```

wenn die Funktion als

```
{ export | reexport } function datafunc <Funktionsname> ();
```

im Dialogskript definiert ist

oder als

```
DM_Boolean DML_c DML_CALLBACK <Funktionsname> (DM_DataArgs *args);
```

wenn die Funktion als

```
{ export | reexport } function c datafunc <Funktionsname> ();
```

im Dialogskript definiert ist.

Der Parameter dieser Funktion ist fest vom ISA Dialog Manager vorgegeben und kann nicht verändert werden. Sie repräsentiert ein Datenmodell (Model-Komponente) welches die Präsentationsobjekte (View-Komponente) mit Datenwerten versorgt oder diese speichert und verwaltet.

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

33.6.12.3 Beispiel

Dialogdatei

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

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

  edittext Et
  {
    .datamodel FuncData;
```

```
    .dataget .text;
    .dataset .text;
    .xauto 0;
    .xright 80;
  }

  pushbutton PbAdd
  {
    .text "Add";
    .xauto -1;
    .width 80;

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

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

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

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

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

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

C-Datei

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

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

#include "datafuncfm.h"

#define DIALOGFILE "~:datafunc.dlg"

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

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

```
    }
    break;
  case MT_set:
    switch(args->attribute)
    {
    case AT_text:
      if (args->data.type == DT_string)
      {
        if (DM_ValueChange(&data_vec, NULL, &args->data, DMF_AppendValue))
          DM_DataChanged(data_id, AT_content, NULL, DMF_Verbose);
      }
      break;
    default:
      break;
    }
    break;
  default:
    break;
  }
}

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

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

    /* reverse the string */
    cp = data.value.string;
    if (cp)
    {
      len = strlen(cp);
      if (len>2)
      {
        len--;
        for (i=0; len>0 && i<len/2; i++)
        {
          ch = cp[i];
          cp[i] = cp[len-i];
```

```
          cp[len-i] = ch;
        }
      }
    }
    if (DM_ValueChange(&data_vec, &index, &data, 0) && data_id)
      DM_DataChanged(data_id, AT_content, &index, DMF_Verbose);
  }
}

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

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

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

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

  data.type = DT_string;
  data.value.string = "^ Enter a string";
```

```
  DM_ValueChange(&data_vec, NULL, &data, DMF_AppendValue);
  data.value.string = "and press 'Add'";
  DM_ValueChange(&data_vec, NULL, &data, DMF_AppendValue);

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

  /* install table of application functions */
  if (!BindFunctions_D (dialogID, dialogID, 0))
    DM_TraceMessage ("There are some functions missing.", 0);

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

  return (EXITOK);
}
```

33.6.12.4 Datenfunktions-Struktur DM_DataArgs

Diese Struktur wird immer als Parameter an Datenfunktionen übergeben.

```
typedef struct
{
  DM_ID        object;
  DM_Method    task;

  DM_Attribute attribute;
  DM_String    identifier;

  DM_Value     index;
  DM_Value     data;
  DM_Value     retval;
} DM_DataArgs;
```

Bedeutung der Elemente

DM_ID object

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

DM_Method task

In diesem Element wird der Grund des Aufrufs angegeben. Hier sind zur Zeit die folgende Werte möglich: *MT_get* oder *MT_set*.

DM_Attribute attribute

Für die Aufrufgründe *MT_get* und *MT_set* steht in diesem Element das Attribut auf das die Funktion anzuwenden ist.

DM_String identifier

In diesem Element steht der Attributbezeichner aus dem *attribute*-Element in String-Form um die Implementierung bei benutzerdefinierten Attributen zu erleichtern.

DM_Value index

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

DM_Value data

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

DM_Value retval

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

33.6.13 Applikationsfunktion DM_DataChanged

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

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

Parameter

-> DM_ID object

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

-> DM_Attribute attribute

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

-> DM_Value *index

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

-> DM_Options options

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

Rückgabewert

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

DM_FALSE Es konnte kein Ereignis erzeugt werden.

Beispiel

Siehe „Datenfunktion“

33.7 Kompatibilität

Das Netzprotokoll hat sich geändert, die alten Versionen sind mit dieser Version nicht mehr kompatibel.

C-, C++- und COBOL-Quelltexte müssen neu kompiliert werden, da sich interne Strukturen geändert haben

Das Binärformat hat sich geändert. Die Binärdateien müssen neu generiert werden.

33.8 Unix

Neu: Die Plattform RHEL7.0 (Kernel 3.10.0, 32- & 64-Bit, gcc4.8.2, GLIBC_2.14) mit Qt 4.8.5 wird unterstützt.

Neu: Die zusätzliche Installationsoption **-desktop** integriert den IDM in den Desktop.

Auf Linux-Maschinen mit GNOME-Desktop und Standardpfaden für **.desktop**-Einträge, MIME-Typen und Icons kann zusätzlich die Option **-desktop** benutzt werden um eine Integration von Editor, Debugger und Hilfe ins Startmenü zu ermöglichen sowie IDM-spezifische Dateitypen (MIME-Typen) einzutragen. Dazu sind Root-Rechte notwendig. Für den Programmstart aus dem Anwendungsmenü müssen aber die IDM-Applikationen im Standardsuchpfad sein. Ein explizites Definieren von `IDM_HOMEDIR` ist dazu nicht nötig.

Das Installationsskript **install.sh** deinstalliert eine vorhandene IDM-Version inklusive Dokumentation und Beispielen über das Deinstallationsskript **lib/IDM/uninstall.sh**.

33.9 Motif

Neu: Die Attribute *.alignment*, *.spacing* und *.tilestyle* am ***Image***-Objekt werden nun unterstützt. Unter Motif ist der Standardwert für *.spacing* am ***Image***-Objekt *2*.

Neu: Das ***Statusbar***-Objekt wird nun auch unter Motif unterstützt.

Änderung: Die Standardhöhenberechnung (bei *.height = 0*) der ***Spinbox*** erfolgt nun unter Berücksichtigung des gesetzten Fonts. Die Höhenberechnung ist so ausgelegt, dass ein ***Edittext*** mit gleichem Font und mit *.borderwidth = 0* hineinpasst und der Text vertikal beschnitten wird.

33.10 Windows

Änderung: Die Optionen *.options[opt_wntsizebug_compat]* und *.options[opt_w2kprefsize_compat]* werden nicht mehr ausgewertet. Sollten Dialogskripte noch nicht angepasst sein, dann müssen das neue Rahmenattribut *.borderstyle* mit der Startoption **-IDMborder5_compat** ausgeschaltet werden. Außerdem muss *.options[opt_wntsizebug_compat]* nun explizit der Wert *true* zugewiesen werden, da der neue Standardwert *false* ist.

33.10.1 Änderung von .toolkit_version unter Microsoft Windows

Das Attribut *.toolkit_version* am ***Setup***-Objekt liefert unter MICROSOFT WINDOWS nicht mehr die Windows-Version für die der IDM kompiliert wurde, sondern die Version der „Common Controls DLL“ (**comctl32.dll**). Diese DLL bestimmt das Aussehen der Oberflächenobjekte, sodass nun ein Wert zurückgeliefert wird, der Aufschluss über das Aussehen und das Layout (z. B. Rahmenbreiten) der Oberfläche gibt.

Die Version wird als Dezimalzahl kodiert:

Majorversion * 100 + Minorversion

Außerdem können noch folgende vordefinierte Werte vorkommen:

0	Keine „Common Controls DLL“ vorhanden.
400	Version konnte nicht bestimmt werden, da „Common Controls DLL“ älter als Version *4.71* ist.
471	„Common Controls DLL“ hat Version *4.71* oder Version konnte nicht bestimmt werden, da ein Fehler beim Aufruf von **DllGetVersion** auftrat.
582	„Common Controls DLL“ hat Version *5.82* oder Version ist *6.00* oder höher und „Visual Styles“ sind ausgeschaltet.
>= *600*	„Visual Styles“ sind eingeschaltet.

33.10.2 Anmerkung zum Window-Objekt

Die Rahmenbreite des Fensters kann unterschiedlich sein, je nachdem welche Subsystem-Version beim Linken der Anwendung angegeben wurde. Beobachtet wurde ein Unterschied unter WINDOWS 7 bei Fenstern mit *.sizeable = false*.

Die Linker-Option **/SUBSYSTEM:WINDOWS,5.01** bei 32-Bit-Programmen bzw. **…,5.02** bei 64-Bit-Programmen erzeugt Rahmenbreiten wie unter WINDOWS XP. In VISUAL STUDIO kann die Linker-Option gesetzt werden, indem in den Projekteigenschaften unter *„Linker“* → *„System“* bei *„Mindest erforderliche Version“ 5.01* bzw. *5.02* eingetragen wird. Weitere Informationen können den Hilfen zu Linker und VISUAL STUDIO von MICROSOFT entnommen werden.

33.11 Kern

12228: Ein Absturz, der beim kompletten Löschen einer mit *scanf*-Format und */tzp*-Optionen formatierten Eingabe bei besonderen Zahlenkonstellationen auftrat, wird nun vermieden.

12056 (Änderung): Es konnte zu Kompilierfehlern kommen, da die IDM-Include-Dateien Schlüsselwörter umdefiniert hatten.

Achtung

Es muss weiterhin auf die korrekte Reihenfolge der Include-Dateien geachtet werden.

Um Probleme mit Schlüsselwörtern zu vermeiden, wurden potentielle Schlüsselwörter und veraltete Schlüsselwörter aus den Include-Dateien **IDMuser.h**, **IDMcpp.h** und **IDMusw.h** entfernt.

Schlüsselwort	IDM-Definition
pascal	DML_pascal
cdecl	DML_c
near	(1)
far	(1)
huge	(1)

(1) Es gibt keine IDM-Definition hierfür.

Außerdem beginnen nun alle IDM-Datentypen mit „DM_“. Dadurch fallen folgende Datentypen weg:

Alter Datentyp	Neuer Datentyp
PTR	*DM_Pointer*
FPTR	*DM_Pointer*
NPTR	*DM_Pointer*

Alter Datentyp	Neuer Datentyp
schar	*DM_Int1*
uchar	*DM_UInt1*
ushort	*DM_UInt2*
uint	*DM_UInt*
ulong	*DM_UInt4*
boolean	*DM_Boolean*
DM_boolean	*DM_Boolean*
SOCKET	*DM_Socket* (1)

(1) Unter UNIX wurde *int* statt *SOCKET* verwendet, es sollte auch *DM_Socket* verwendet werden.

Die alten Datentypen können durch die Definition des Präprozessorsymbols `DM_5_COMPAT_TYPES` eingeschaltet werden.

Alle Konstanten beginnen nun auch mit „DM_“. Die folgenden Konstanten sollten sukzessive ersetzt werden:

Alte Konstante	Neue Konstante
TRUE	*DM_TRUE*
FALSE	*DM_FALSE*
FAILURE	*DM_FAILURE*

Die alten Konstanten können durch die Definition des Präprozessorsymbols `DM_5_COMPAT_CONSTANTS` eingeschaltet werden. Sollte es Probleme mit *TRUE*, *FALSE* oder *FAILURE* geben, so kann die Definition durch das Präprozessorsymbol `DM_OMIT_BOOL_CONSTANTS` vermieden werden.

Nicht mehr verwendet werden folgende Präprozessorsymbole, wobei die neuen bereits im IDM 5 verwendet wurden:

Altes Symbol	Aktuelles Symbol
DOS	WIN32
WSIWINXP	WSIWIN=0x501
WSIWIN32	(1)

Altes Symbol	Aktuelles Symbol
WSIWINVISTA	(2)

(1) IDM für WINDOWS 2000 wird nicht mehr unterstützt.

(2) IDM für WINDOWS VISTA gab es nicht.

11987: **DM_GetValue()** gibt jetzt einen korrekten UTF16-Leerstring zurück, wenn das abgefragte Attribut auf *null* gesetzt ist.

11353: Der IDM erzwingt nun beim Interface-Schreiben dass ein gesetzter Label (geerbter Label nicht ausreichend) bei exportierten Objekten (direkt unterhalb des Moduls) vorhanden ist.

Das Setzen eines Labels bei exportierten Objekten ist zwingend erforderlich um so eine korrekte Interface-Erzeugung und Modul-Nachladung zu ermöglichen. Falls der IDM beim Interface-Schreiben auf ein exportiertes Objekt ohne gesetzten Label trifft, so wird eine Fehlermeldung ausgegeben.

11319 (Änderung): Das Attribut *.options[opt_old_select]* am ***Tablefield*** hat nun den Standardwert *false*.

9108 (Änderung): Das Attribut *.options[opt_old_select]* am ***Poptext*** hat nun den Standardwert *false*.

Neu: Die Slot-Verwaltung wurde erweitert, sodass ein ungültiger Zugriff auf ein zerstörtes Objekt unter Umständen erkannt und vom Dumpstate (**dumpstate()**) bzw. der **debug()**-Ausgabe ausgeben wird. Dies funktioniert nur für die letzten 250 zerstörten Objekte sicher, wenn kein Anlegen dazwischenliegt. Da das Objekt zerstört wurde, kann nur die Klasse geliefert werden.

Die Dumpstate-Funktionalität gibt nun im SLOTS-Abschnitt auch IDM-Objekte aus, auf die nach ihrer Zerstörung noch ein ungültiger Zugriffsversuch erfolgte. Diese sind typischerweise an der Eigenschaft *„refs=0“*, *„lock=0“*, *„drop=1“* und *„object="Unknown ID"“* zu erkennen. Die Ausgabe dieser ungültigen Zugriffe kann aber nur für die letzten 250 zerstörten Objekte erfolgen und wird durch zwischenzeitlich angelegte Objekte beeinträchtigt. Ein Zugriff auf eine ungültige Objekt-ID wird vom IDM als Hinweis genommen eine Dumpstate-Ausgabe beim Beenden der Anwendung durchzuführen bzw. den SLOTS-Abschnitt einzuschließen.

Neu: Der Dumpstate-Ausgabe wurde ein HINTS-Abschnitt hinzugefügt.

Neu: Für eine bessere Problemanalyse enthält die Warnung *„Queue Full“* nun Details über das Ereignis.

33.12 Editor

Änderung: Im Regelvokabular sind eingebaute Funktionen und Methoden nun aufgetrennt.

Index

=

=> 318

3

3D-Rahmen 273

A

.acc_label 190, 198

.acc_text 190, 197

alias 203

allowundefined 287

Anwendungsfunktion 203

append() 148

Application

  SSH 160

  SSL 160

:apply() 393

asynchroner Methodenaufruf 142

AT-acc-label 198

AT-acc-text 197

AT-allowundefined 287

AT-borderstyle 279

AT-calendaralignment 288

AT-certificatefile 162

AT-colcount 304

AT-colorname 269

AT-coltitle 305

AT-colwidth 305

AT-content 306

AT-cursorname 269

AT-dataget 390

AT-dataindex 391

AT-datamap 392

AT-datamodel 389

AT-dataoptions 392

AT-dataselect 244

AT-dataselectattr 246

AT-dataselectcount 248

AT-dataselecttype 247

AT-dataset 391

AT-fontname 270

AT-format 288

AT-masterapplication 54, 76

AT-maxvalue 291

AT-mincolwidth 307

AT-minvalue 291

AT-options 102

AT-password 162

AT-picheight 308

AT-picture 308

AT-picwidth 309

AT-privatekeyfile 101

AT-propscale 43

AT-publickeyfile 102

AT-real-modified 292

AT-rowcount 310

AT-scalestyle 42

AT-searchpath 126

AT-selected 311

AT-shortdaynames 293
AT-smallpicheight 311
AT-smallpicture 312
AT-smallpicwidth 313
AT-style 293, 314
AT-tiledpi 43
AT-today 294
AT-todaymarker 295
AT-trailingdates 295
AT-username 163
AT-value 296
AT-weeknumbers 298
AT-xmargin 44
AT-ymargin 44
AT_acc_label 198
AT_acc_text 197
AT_allowundefined 287
AT_borderstyle 279
AT_calendaralignment 287
AT_certificatefile 162
AT_colcount 304
AT_colorname 269
AT_coltitle 305
AT_colwidth 305
AT_content 306
AT_cursorname 269
AT_dataget 390
AT_dataindex 391
AT_datamap 392
AT_datamodel 389
AT_dataoptions 392
AT_dataselect 244
AT_dataselectattr 246
AT_dataselectcount 248
AT_dataselecttype 247
AT_dataset 391
AT_fontname 270
AT_format 288
AT_masterapplication 54, 76
AT_maxvalue 291
AT_mincolwidth 307
AT_minvalue 291
AT_options 102
AT_password 162
AT_picheight 308
AT_picture 308
AT_picwidth 309
AT_privatekeyfile 101
AT_propscale 43
AT_publickeyfile 102
AT_real_modified 292
AT_rowcount 310
AT_scalestyle 42
AT_searchpath 126
AT_selected 311
AT_shortdaynames 293
AT_smallpicheight 311
AT_smallpicture 312
AT_smallpicwidth 313
AT_style 293, 314
AT_tiledpi 43
AT_today 294
AT_todaymarker 295
AT_trailingdates 295

AT_username 163
AT_value 296
AT_weeknumbers 298
AT_xmargin 44
AT_ymargin 44
AU 199
automatischer Start 208

B

Bauprozess
  Optimierung 108
Beispiel
  Datenmodell 241, 369, 372, 377, 380
-bindir 122, 125
BindThruLoader 201
border_compat 274
bordercolor 273
borderstyle 273, 279
borderwidth 273
Build 108, See also *Bauprozess*
builder 93
Builder-Prozess 108
Builder-Prozess-Modus 93
-buildermodlimit 109
-buildersetdirty 109

C

Caching
  doccursor 244
  Doccursor 241
calendaralignment 287
Call Data (CD) 389
:calldata() 388, 405
Canvas-Funktion
  Qt 252
CCR_event 253
CD (Call Data) 389
CE (Data Changed Event) 389
.certificatefile 162
clean 135
-cleancompile 121, 125
-cleancompile1 125
-cobbasename 110
-cobname 110
COBOL
  Unicode 201
.codepage 203
Codepage 203
colcount 304
:collect() 395
colorname[integer] 269
coltitle[integer] 305
colwidth[integer] 305
-compile 121, 124
-compile1 125
concat() 237
config 144
configurable 144
connect
  SSL 160
content[index] 306
countof() 324
-cpyname 110
cursorname[integer] 269

D

Data Changed event (CE) 389
datachanged 371, 384
datafunc 407
dataget 372
dataget[attribute] 390
dataindex 376
dataindex[attribute] 391
datamap 376
datamap[attribute] 392
datamodel 371
Datamodel-Attribut
  Datentyp 241, 247
  Kardinalität 241, 248
datamodel[attribute] 389
dataoptions 244, 383
dataoptions[enum] 392
dataselect[attribute] 244
dataselectattr[attribute] 246
dataselectcount[attribute] 248
dataselecttype[attribute] 247
dataset 372
dataset[attribute] 391
Dateierweiterungen
  Registrierung 221
Datenfunktion 382, 407
Datenmodell 367
  Beispiel 241, 369, 372, 377, 380
  Doccursor 240
  Listbox 381
  Standardkopplungen 374
  tablefield 375
  Tracing 389
  XML 240
Datentyp
  Datamodel-Attribut 241, 247
  hash 318
  list 317
  matrix 318
  refvec 318
  vector 317
datetime 282
Debugger
  Attribute überwachen 227
  automatischer Start 208
  Object View 226
  Objektansicht 226
  Objekte überwachen 227
delete() 150
DM-value-long-first 201
DM-value-long-second 201
DM_5_COMPAT_CONSTANTS 418
DM_5_COMPAT_TYPES 418
DM_CanvasUserArgs
  Qt 252
DM_Control 196
DM_ControlEx 119, 196
  use 125
DM_DataArgs 413
DM_DataChanged 414
DM_IndexReturn 365
DM_Initialize
  Locale 211

DM_LoadProfile 145
DM_OMIT_BOOL_CONSTANTS 418
DM_PicInfo 270
DM_SendMethod 143
DM_SHARED_MODULES 84
DM_String 344
DM_StringChange 362
DM_StringInit 360
DM_StringReturn 364
DM_Value 343
DM_ValueChange 349
DM_ValueCount 354
DM_ValueGet 352
DM_ValueIndex 357
DM_ValueInit 345
DM_ValueReturn 358
DMcob_CreateFromModel 172
DMcob_FatalAppError 201
DME_ArgMayInvalid 82
DME_BuiltinMayUndef 82
DME_ChildAttrMayUndef 82
DME_FunctionMayUndef 82
DME_ImportNotUsed 79
DME_MethodMayUndef 82
DME_ScopeMayClash 82
DME_TypeMayClash 82
DMF-CreateInvisible 172
DMF-CreateModel 172
DMF_FC_RecordStub 103
DMF_PCREBinding 136
DMF_UIAutomationMode 196
DMFSearchPath 119
dmw_datetime 282
dmw_listview 298
doccursor
  Caching 244
Doccursor
  Caching 241
  Datenmodell 240
Dock Pattern 190
dopt_apply_on_event 384, 393
dopt_apply_on_unmap 383, 393
dopt_cache_data 241, 244
dopt_propagate_on_changed 384, 393
dopt_propagate_on_start 384, 393
dopt_represent_on_init 383, 393
dopt_represent_on_map 383, 393
DT_hash 343
DT_index 343
DT_list 343
DT_matrix 343
DT_refvec 343
DT_vector 343
dynamisierte Funktionsaufrufe 202

E

Editor
  kontextsensitive Hilfe 215
EITK_analyzer 114
Ereignis
  datachanged 371
exchange() 153
exec
  SSH 161

ExpandCollapse Pattern 189

F

Farbverlauf 265
Fehleranalyse 110
Fehlerhandler 114
Fehlermeldung
ormat 110
find() 332
finish 135
font raster 57
fontname[integer] 270
Fontraster 57
for-Schleife 82
foreach 342
format 288
Format 212
Reguläre Ausdrücke 142
Formatstring 289
Funktionsaufruf
dynamisiert 202

G

Geometrie
Rahmen 276
getToolkitDataEx 316
getvector() 338
Gradient 265
Grid Pattern 187
GridItem Pattern 188

H

hash 318
HP UFT 193
HTML 253

I

Identifikator 3
IDM_ANALYZERCONFIG 82
IDM_BORDER5_COMPAT 278
IDM_FLOWGRAPH 114
IDM_FONTRASTER_COMPAT 57
IDM_HOMEDIR 217
IDM_SEARCHPATH 119, 123
IDM_WRITEONCHANGE 108
-IDManalyseseverity 87, 113
IDManalyzerconfig 82
-IDManalyzerconfig 82
-IDMborder5_compat 278
IDMconnect 164
IDMcpp.h 417
-IDMdbg 225
-IDMDBGargs 208
-IDMdisable_error 98
IDMdisable_errors 83
-IDMdisable_errors 83
-IDMerroranalysis 87
-IDMflowgraph 114
IDMfontraster_compat 57
IDMsearchpath 119
-IDMsearchpath 123
-IDMssl_certfile 164

-IDMssl_options 103
-IDMssl_privkeyfile 164
-IDMtellport 164
IDMtransport
SSL 163
-IDMusepathmodifier 124
IDMuser.h 417
IDMusw.h 417
-IDMwriteonchange 108
-ifdir 122, 125
import
Vergleich mit use 120
Import
unbenutzt 79
Include-Datei 417
indexat() 325
init 135
insert() 155
Installation
IDM für Windows 221
itemcount() 323

J

join
string 331
join() 330

K

Kalenderwoche 297
Kardinalität
Datamodel-Attribut 241, 248
keys() 327
Kompatibilität
Rahmen 274
Konfigurationsdatei 144
kontextsensitive Hilfe
Editor 215
Kontrollflussanalyse 114

L

level 179
libssh 159
Linienförmiger Rahmen 273
list 317
Listbox
Datenmodell 381
listview 298
loadprofile() 146
Locale 209

M

masterapplication[enum] 76
.masterapplication[enum] 76
matrix 318
max() 337
maximale Rekursionstiefe 223
maxvalue 290
Methodenaufruf
asynchron 142
Micro Focus Visual COBOL 201
min() 336
mincolwidth[integer] 307
minvalue 291
missing return 114

Mnemonics 263
Model 367
Model-View-Presenter 367
Modul
  Beenden 135
  Initialisierung 135
MVP 367

N

-noif 125

O

Object View 226
Objektansicht 226
  Attributänderungen 230
  Attribute filtern 229
  Attribute überwachen 227
  indizierte Attribute 228
  Objektanzeige 226
  Objekte suchen 227
  Objekte überwachen 227
  Objektliste 226
Objektanzeige
  this 229
OpenSSL 159
opt_addevents 253
opt_animated 262
opt_auto_close 179
opt_cert_required 103
opt_fontraster_compat 57
opt_graphicsview 252
opt_hscroll 166
opt_html 253
opt_nested_docks 262
opt_no_ssl_v2 103
opt_scroll_pixels 205
opt_tabbed_docks 262
opt_verify_peer 103
opt_window_size 262
opt_xmborder_compat 275
optimierter Bauprozess 108
Option
  bindir 125
  cleancompile 125
  cleancompile1 125
  compile 124
  compile1 125
  IDManalyzerconfig 82
  IDMdisable_errors 83
  IDMsearchpath 123
  IDMusepathmodifier 124
  ifdir 125
  noif 125
  -bindir 125
  -buildersetdirty 109
  -buildermodlimit 109
  -cleancompile 125
  -cleancompile1 125
  -cobbasename 110
  -cobname 110
  -compile 124
  -compile1 125
  -cpyname 110
  -IDManalyseseverity 87

-IDManalyzerconfig 82
-IDMDBGargs 208
-IDMdisable_error 98
-IDMdisable_errors 83
-IDMerroranalysis 87
-IDMflowgraph 114
-IDMsearchpath 123
-IDMssl_options 103
-IDMusepathmodifier 124
-IDMwriteonchange 108
-ifdir 125
-noif 125
-recompile 125
-recompile1 125
recompile 125
recompile1 125
.options[enum] 102

P

Paket-Pfad 117
.password 162
PCRE 135
PCRE-Bibliothek
Anbindung 135
Version 135
picheight 308
picture[integer] 308
picwidth 309
Presenter 368
Logik 371
.privatekeyfile 101
profile 144
:propagate() 397
.publickeyfile 102

Q

Qt
Treeview 177
queryhelp() 147

R

Rahmen
Geometrie 276
Kompatibilität 274
Tablefield (Motif) 275
Rahmen aus 273
RangeValue Pattern 186
raster wdith 57
Rasterbreite 57
real_modified 292
-recompile 121, 125
-recompile1 125
Record-Parameter 202
Redefinition
:represent() 400
:retrieve() 404
Redistribution von IDM-Anwendungen 221
reference string 57
Referenzstring 57
refvec 318, 320
reg-Datei 170
regex() 137
regex_count 140
regex_eval 140

regex_locate 140
regex_match 140
regex_unmatch 140
regex_vars 140
Registrierung von Dateierweiterungen 221
Reguläre Ausdrücke 135
  Format 142
Rekursionstiefe
  IDM für Windows 223
  maximale 223
  Stackgröße 223
:represent() 400
:retrieve() 404
return 114
rowcount 310
RSH 92

S

saveprofile() 146
Schlüssel 159
Scroll Pattern 186
ScrollItem Pattern 187
.searchpath 119, 126
selected[integer] 311
Selection Pattern 188
SelectionItem Pattern 188
Selektionsmuster 240, 244
semantische Prüfung 111
sendmethod() 143
setvector() 340
shortdaynames 292
showitem 255
smallpicheight 311
smallpicture[integer] 312
smallpicwidth 313
sort() 334
split() 239
SSH 92, 159
  Schlüssel 159
  Zertifikat 159
SSL 159
  Schlüssel 159
  Zertifikat 159, 162
Stackgröße
  Rekursionstiefe 223
Standardkopplungen (Datenmodell) 374
start 135
strreplace() 234
style 293, 314
Suchpfad 118, 123
  setzen 119
Systemformat 289

T

Table Pattern 190
tablefield
  Datenmodell 375
Tablefield
  Rahmen (Motif) 275
TableItem Pattern 190
Testlizenz
  Ablaufdatum 249
this
  Objektanzeige 229

tile
text 199
today 294
todaymarker 294
Toggle Pattern 189
Toolkit-Rahmen 273
toolkit_version 416
top_most 107
Tracefile Analyzer 114
Tracing 93
Datenmodell 389
trailingdates 295
Transform Pattern 187
transparente Farbe 251
transport
SSL 160
Treeview 179
Qt 177

U

UC_API 315
UC_I_querykey 231
UC_InqArg 232, 316
UC_InqName 231, 315
UC_Inquire 317
UC_Querykey 232
UC_Replace 316
UI-Stil 251
UI Automation 180
Control Pattern 181, 183
Control Type 180
Objektidentifikation 190
Property 183
Tracing 199
Umgebungsvariable
IDM_FLOWGRAPH 114
IDM_HOMEDIR 217
IDM_WRITEONCHANGE 108
unbenutzter Import 79
Unicode 201
:unuse() 122, 128
use 115
Syntax 117
Vergleich mit import 120
Use-Pfad 115, 118, 124
Syntax 117
:use() 122, 127
.username 163
-userregistry 170

V

value 296
Value Pattern 186
valueat() 326
values() 328
vector 317
Verweisoperator 318
View 367
Visual COBOL 201

W

Warnung
Format 110
missing return 114

weeknumbers 297
Window Pattern 189
+writeheader 202-203
-writeole 170
+/-writetrampolin 104, 110

X

XML
    Datenmodell 240

Z

Zertifikat 159
    SSL 162