ISA Dialog Manager

ENTWICKLUNGSUMGEBUNG

A.06.03.b

Dieses Handbuch enthält eine Übersicht über den ISA Dialog Manager und die Dokumentation. Außerdem sind Start-optionen, Umgebungsvariablen, Konfigurationsdatei, Ablauf-verfolgung (Tracing) und Hilfen zur Fehleranalyse beschrieben.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Deutschland

Darstellungskonventionen

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

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

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

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

Beschreibungsmodus

Alle Schlüsselwörter sind fett und unterstrichen, z.B.

variable **integer** **function**

Indizierung von Attributen

Syntax für indizierte Attribute:

[I]

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

Identifikatoren

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

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

Die maximale Länge eines Identifikators beträgt *31* Zeichen.

Beschreibung der zugelassenen Identifikatoren in Backus-Naur-Form

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

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

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

<Ziffer> ::= 1 | 2 | 3 | … 9 | 0

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

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

Inhalt

1 Übersicht

Der Dialog Manager ist ein Entwicklungswerkzeug zur Erstellung grafischer Benutzerschnittstellen. Dieses Werkzeug kann von Ihnen als eine systemunabhängige Schnittstelle zwischen einem Anwendungsprogramm und einem Fenstersystem der jeweiligen Zielumgebung eingesetzt werden.

Der Dialog Manager basiert auf einem Benutzerschnittstellen-Modell, das

- eine Präsentationsschicht,
- eine Dialogschicht und
- eine Anwendungsschicht

enthält.

Alle Aspekte der Präsentationsschicht, d.h. das Aussehen des Dialoges an der Benutzerschnittstelle, können durch das Setzen von Objektattributwerten manipuliert werden (siehe Handbücher „Editor“, „Objektreferenz“ und „Attributreferenz“).

Die Dialogkontrolle wird durch die Änderung von Attributwerten und durch das Aufrufen von Anwendungsfunktionen definiert.

Funktionen der Anwendung werden z.B. durch Referenzierung des Funktionsnamens und der benötigten Parameter aufgerufen (siehe Handbücher „C-Schnittstelle - Funktionen“ bzw. „COBOL-Schnittstelle“).

1.1 Erläuterung der Struktur und der Komponenten

Die DM-Eingabedatei ist in zwei große Datengruppen gegliedert.

- Die erste Datengruppe, der **Definitionsteil**, enthält die Definitionen für die Dialogidentifikatoren, Ressourcen, Defaultobjekte, Vorlagen und Objekte.
- Die zweite Datengruppe enthält die **Regelbasis**. Diese Regeln beschreiben den Ablauf des Dialoges (siehe Handbuch „Regelsprache“).

Der Definitionsteil muss hierarchisch gegliedert sein, d.h. es können nur Referenzen zu bereits definierten Ressourcen oder Objekten gemacht werden. Wenn also das Objekt "Fenster" die Hintergrundfarbe "Blau" verwenden soll, so muss "Blau" vorher definiert werden.

Diese Gliederung hat zur Folge, dass die Definitionen in folgender Reihenfolge vorzunehmen sind:

- Dialog
- Ressourcen
- Defaultobjekte

- Vorlagen bzw. Modelle
- Objekte
- Regeln

Da die Regelabarbeitung durch auftretende Ereignisse gesteuert wird, ist innerhalb der Regeldefinition keine besondere Gliederung oder Reihenfolge notwendig.

1.2 Software

Die **Standard-Entwicklungsumgebung** des ISA Dialog Managers umfasst

- Simulationskomponente und Laufzeitbibliotheken
- interaktiver WYSIWYG-Editor
- C- und C++-Schnittstelle
- XML-Schnittstelle
- Debugger und Profiler
- Tracefile Analyzer (nicht verfügbar für den IDM FÜR MOTIF)

Im folgenden sind die zusätzlich erhältlichen **Optionen** aufgeführt:

- Programmierschnittstellen
  - COBOL-Schnittstelle
  - COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL
- Kommunikation
  - Distributed Dialog Manager (DDM)
  - OLE-Schnittstelle
- USW-Schnittstelle für die Erweiterung des IDM um benutzerdefinierte Widgets
- Eclipse Plugins
  - IDM Eclipse Plugin für die Erstellung und Bearbeitung von Dialogskripten
  - ISA Makefile Generator

1.3 Dokumentation

Installationsanleitung

Diese Anleitung beschreibt die Systemvoraussetzungen für den ISA Dialog Manager und wie er auf den verschiedenen Plattformen installiert wird.

Kurzreferenz

Diese Referenz bietet eine kurze Übersicht über Objekte, Ressourcen, Attribute, Methoden, Anweisungen der Regelsprache und eingebaute Funktionen zum Nachschlagen.

Release Notes IDM 6
Release Notes IDM 5
Release Notes IDM 4

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

1.3.1 Attribute, Objekte und Ressourcen

Attributreferenz

In diesem Handbuch sind alle vordefinierten Attribute der ISA Dialog Manager Objekte beschrieben. Es enthält die Definitionen und Datentypen der Attribute für die Regelsprache und die Programmiersprachen C und COBOL.

Objektreferenz

In diesem Handbuch sind alle Objekte des ISA Dialog Managers beschrieben. Zu jedem Objekt sind seine vordefinierten Attribute und Methoden sowie die unterstützten Ereignisse und Kindobjekte aufgeführt.

Ressourcenreferenz

In diesem Handbuch sind alle Ressourcen des ISA Dialog Managers beschrieben. Ressourcen sind Objekte wie Cursors, Farben, Texte und Schriftarten über die bestimmte Merkmale des Aussehens oder Verhaltens von IDM Objekten festgelegt werden können.

1.3.2 Programmierung

1.3.2.1 Eingebaute Regelsprache

Handbuch Regelsprache

Dieses Handbuch erläutert die Regelsprache des ISA Dialog Managers in der das dynamische Verhalten der Benutzeroberfläche programmiert werden kann. Im Handbuch sind beispielsweise Ereignis- und Regelverarbeitung, Datentypen, Sprachumfang der Regelsprache, Syntax von Anweisungen und eingebaute Funktionen beschrieben.

Handbuch Programmiertechniken

Dieses Handbuch vermittelt grundlegende Techniken für die Entwicklung von Benutzerschnittstellen mit dem ISA Dialog Manager. Zu den Themen gehören Modularisierung, Verwendung von Modellen, objektorientierte Programmierung und das Datenmodell.

Methodenreferenz

In diesem Handbuch sind alle vordefinierten Methoden der ISA Dialog Manager Objekte erläutert. Es enthält die Methodendefinitionen mit ihren Parametern und Rückgabewerten. Außerdem ist beschrieben, welche Methoden überschrieben (redefiniert) werden können.

Handbuch Benutzerdefinierte Attribute und Methoden

Dieses Handbuch erläutert die Verwendung von benutzerdefinierten Attributen und Methoden.

Handbuch XML-Schnittstelle

Dieses Handbuch beschreibt die Schnittstelle zur Verarbeitung von XML-Daten (Extensible Markup Language) mit dem ISA Dialog Manager. Die dafür vorhandenen Objekte mit ihren Attributen und Methoden werden erklärt.

1.3.2.2 Programmierschnittstellen (Application Programming Interfaces, APIs)

Handbuch C-Schnittstelle - Grundlagen

In diesem Handbuch wird die prinzipielle Struktur der Schnittstelle des ISA Dialog Managers für in C entwickelte Anwendungen dargestellt. Das Handbuch beschreibt die Datentypen sowie das Übersetzen und Linken der Anwendungen.

Handbuch C-Schnittstelle - Funktionen

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

Handbuch C++-Schnittstelle

In diesem Handbuch ist die Schnittstelle des ISA Dialog Managers für in C++ entwickelte Anwendungen dargestellt. Es beschreibt die Klassen zur Anbindung des IDM und deren Methoden. Die Funktionsweise von Code-Generator und Code-Mischer wird erklärt.

Handbuch COBOL-Schnittstelle

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

1.3.2.3 Kommunikation und Integration

Handbuch Distributed Dialog Manager (DDM)

In diesem Handbuch ist die Netzwerkoption des ISA Dialog Managers (Distributed Dialog Manager, DDM) beschrieben. Mit ihr können verteilte Anwendungen entwickelt werden, bei denen Benutzeroberfläche und Anwendungen auf verschiedenen Rechnern in einem Netzwerk laufen.

Handbuch Benutzerdefinierte Widgets (USW)

Die USW-Schnittstelle (User Supplied Widget) ist eine Option des ISA Dialog Managers mit der selbst entwickelte Widgets in den IDM integriert werden können. Das Handbuch erläutert die Entwicklung von benutzerdefinierten Objektklassen und ihr Zusammenspiel mit dem IDM.

Handbuch OLE-Schnittstelle

Dieses Handbuch beschreibt die OLE-Schnittstelle des ISA Dialog Managers, die als Option des IDM für Microsoft Windows erhältlich ist. OLE (Object Linking and Embedding) ist eine Technik mit der Objekte miteinander kommunizieren und ineinander eingebettet werden können. Das Handbuch erläutert, wie OLE-Clients und -Server mit dem IDM implementiert werden.

Handbuch Automatisches Testen und Barrierefreiheit

Dieses Handbuch beschreibt Funktionen, die der ISA Dialog Manager bereitstellt um das automatische Testen der Benutzeroberfläche und die Entwicklung barrierefreier Anwendungen zu unterstützen.

1.3.3 Entwicklungsumgebung

Handbuch Entwicklungsumgebung (dieses Handbuch)

Dieses Handbuch enthält eine Übersicht über den ISA Dialog Manager und die Dokumentation. Außerdem sind Startoptionen, Umgebungsvariablen, Konfigurationsdatei, Ablaufverfolgung (Tracing) und Hilfen zur Fehleranalyse beschrieben.

Editor-Handbuch

In diesem Handbuch ist der grafische Editor des ISA Dialog Managers beschrieben mit dem Dialoge interaktiv erstellt und geändert werden können.

Debugger-Handbuch

In diesem Handbuch ist der Debugger des ISA Dialog Managers beschrieben. Der Debugger ist ein Werkzeug für das Auffinden von Fehlern in Dialogen.

Profiler-Handbuch

In diesem Handbuch ist der Profiler des ISA Dialog Managers beschrieben. Der Profiler ist ein Werkzeug mit dem sich das Laufzeitverhalten von Dialogen analysieren lässt.

Tracefile Analyzer Handbuch

In diesem Handbuch ist der Tracefile Analyzer des ISA Dialog Managers beschrieben, der verschiedene Hilfsmittel für die Auswertung von Tracedateien bietet.

2 Startoptionen

Beim Start des Dialog Managers bzw. einer mit dem Dialog Manager erstellten Anwendung können in der Kommandozeile folgende Parameter angegeben werden:

-IDMbinerror <boolean>

Wird diese binäre Option auf *false* gesetzt (`-IDMbinerror=false`), so werden Fehlermeldungen beim binären Lesen unterdrückt.

-IDMcallstack <boolean>

Mit dieser Option kann die Verwaltung des Callstack, der die Aufrufliste von Regeln, Methoden, Builtin-, Applikations- und Schnittstellenfunktionen speichert um sie beim Dumpstate auszugeben, unterbunden werden.

Grundeinstellung

true

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMcatchexceptions <boolean>

Mit dieser Option kann die Einrichtung eines Exception-Catcher unterbunden werden.

Grundeinstellung

true

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMcolor <integer>

Mit Hilfe dieser Option können die aktuell zu verwendenden Farben auf die angegebene Variante umgestellt werden.

-IDMconfigfile <filename>

Bei Angabe dieser Option wird die angegebene Datei als Konfigurationsdatei verwendet und diese Option aus dem *argv*-Parameter entfernt. Fehlt diese Option, wird geprüft, ob die Umgebungsvariable `IDM_CONFIGFILE` gesetzt ist. Falls dies der Fall ist, wird die dort angegebene Datei als Konfigurationsdatei verwendet.

Wurde eine Konfigurationsdatei angegeben und kann diese Datei geöffnet werden, so werden die dort angegebenen Optionen vor die restlichen auf der Kommandozeile angegebenen Optionen gesetzt.

Die Konfigurationsdatei enthält eine Liste von Kommandozeilenoptionen. Als Trennzeichen zwischen den einzelnen Optionen und ihren Parametern sind folgende erlaubt: BLANK („ “), TAB („\t“), RETURN („\n“).

Die so erhaltenen Optionen werden behandelt als ob sie alle auf der Kommandozeile angegeben wurden, d.h. auch die Anwendung erhält sie.

Anmerkung

-IDMerrfile kann damit nicht gesetzt werden.

-IDMconsole

Diese Option bewirkt, dass unter MICROSOFT WINDOWS – analog zu Unix – Log-File und Fehlermeldungen in den Standard-Ausgabekanal (STDOUT) geschrieben werden. Läuft der Prozess nicht innerhalb einer Konsole, so wird eine neue Konsole für mögliche Ausgaben geöffnet.

Verfügbarkeit

Versionen ab A.05.02.f; Plattform MICROSOFT WINDOWS

-IDMcp_appl <codepage>
-IDMcp_format <codepage>
-IDMcp_input <codepage>
-IDMcp_io <codepage>
-IDMcp_output <codepage>

Optionen zum Setzen der Codepage für verschiedene Operationen. Siehe Kapitel „Startoptionen zum Definieren von Codepages“.

-IDMcursor <integer>

Mit Hilfe dieser Option können die aktuell zu verwendenden Cursor auf die angegebene Variante umgestellt werden.

-IDMdumpstate <enum>

Mit Hilfe dieser Option lässt sich die Ausgabe von DM Zustandsinformationen (Dumpstate) beeinflussen. Durch den *<enum>*-Parameter wird die Ausgabe bestimmter Informationen erzwungen.

Parameter

Wert (enum)	Bedeutung
dump_all	Alle Abschnitte werden gekürzt herausgeschrieben. Entspricht der Ausgabe bei einem FATAL ERROR.
dump_error	Die Abschnitte ERRORS, CALLSTACK und EVENTS werden gekürzt herausgeschrieben. Dies ist auch die normale Ausgabe im Falle eines EVAL ERRORS.

Wert (enum)	Bedeutung
dump_events	Der Abschnitt THISEVENT/EVENT QUEUE wird ungekürzt herausgeschrieben.
dump_full	Alle Abschnitte werden ungekürzt herausgeschrieben.
dump_locked	Der Abschnitt SLOTS wird ungekürzt herausgeschrieben. Zusätzlich werden für gesperrte (locked) Objekte deren Attributwerte herausgeschrieben.
dump_memory	Der Abschnitt MEMORY wird ungekürzt herausgeschrieben.
dump_none	Nichts passiert (kein Herausschreiben)
dump_process	Der Abschnitt PROCESS wird ungekürzt herausgeschrieben.
dump_short	Alle Abschnitte (außer SLOTS) werden gekürzt herausgeschrieben.
dump_slots	Der Abschnitt SLOTS wird ungekürzt herausgeschrieben.
dump_stack	Der Abschnitt CALLSTACK wird ungekürzt herausgeschrieben.
dump_usage	Der Abschnitt USAGE wird ungekürzt herausgeschrieben.
dump_uservisible	Der Abschnitt VISIBLE OBJECTS wird ungekürzt für alle sichtbaren Toplevel-Objekte inklusive deren Kindobjekte, vordefinierten und benutzerdefinierten Atttribute herausgeschrieben.
dump_visible	Der Abschnitt VISIBLE OBJECTS wird ungekürzt herausgeschrieben.

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMdumpstateseverity <string>

Die Ausgabe eines Dumpstate erfolgt normalerweise wenn ein EVAL ERROR oder ein FATAL ERROR auftritt. Mit dieser Option kann zusätzlich eine Dumpstate-Ausgabe bei anderen Fehler- und Meldungsarten erzwungen werden.

Parameter

Erstes Zeichen	Dumpstate-Ausgabe…
E	bei Fehlermeldungen [E: …]
F	bei fatalen Fehlermeldungen [F: …]
I	bei informativen Meldungen [I: …], Warnungen und Fehlermeldungen

Erstes Zeichen	Dumpstate-Ausgabe…
O	nur für EVAL ERROR
W	bei Warnungen [W: …] und Fehlermeldungen

Grundeinstellung

„F“

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMenv <Variable>=<Wert>

Der Dialog Manager erlaubt es, Umgebungsvariablen von der Kommandozeile aus zu setzen. Dies ist insbesondere für den DM unter MS Windows interessant. Unter MS Windows ist es eigentlich nicht möglich, Umgebungsvariablen für alle Anwendungen von MS Windows aus zu setzen.

Mit der Kommandozeilenoption **-IDMenv <Variable>=<Wert>** können diese Variablen für die DM-Anwendung gesetzt werden. Die Umgebungsvariablen von MS Windows werden damit für die DM-Anwendung überschrieben. Die Umgebungsvariablen werden nur temporär und nur für die DM-Anwendung gesetzt.

Hinweis

Zwischen dem Variablennamen, dem Gleichheitszeichen und dem Variablenwert dürfen sich keine Leerzeichen befinden.

Diese Option kann auch mehrfach angegeben werden, um verschiedene Variablen setzen zu können.

Beispiel

Windows

```
idm -IDMenv BINARY=c:\idm\bin -IDMenv IF=c:\idm\if dialog.dlg
```

Unix

```
idm -IDMenv binary=/home/user1/bin -IDMenv IF=/usr/idm/interface
dialog.dlg
```

-IDMerrfile <filepath>|none

Mit dieser Option kann eine Fehlerdatei gesetzt werden. Diese Datei enthält alle Fehler, die während dem Ablaufen der Applikation aufgetreten sind.

Durch Setzen der Umgebungsvariable `IDM_LOGFILE` auf einen Dateipfad kann dasselbe bewirkt werden.

Die Startoption überschreibt die Umgebungsvariable wenn beide gesetzt sind.

Bei Startoption und Umgebungsvariable kann *none* anstelle eines Dateipfads angegeben werden, um das Anlegen einer Fehlerdatei zu unterbinden.

Im Dateinamen können Platzhalter verwendet werden (siehe Kapitel „Platzhalter in Dateinamen“).

Hinweis

Bei gleichzeitiger Nutzung der Option **–IDMtracefile** (oder deren Umgebungsvariable) kann die hier angegebene Fehlerdatei auch ohne Inhalt bleiben oder sie wird überhaupt nicht erzeugt. Eventuelle Fehlermeldungen werden in diesem Fall in die Tracedatei herausgeschrieben.

-IDMerrwinfile <filepath>|none

Diese Option leitet Fehlermeldungen, die als Dialogfenster angezeigt oder in der Konsole ausgegeben werden, in die angegebene Datei um. Damit können beispielsweise (insbesondere unter Microsoft Windows) Serverprozesse ohne notwendige Nutzerinteraktion (Bestätigung eines Fehlerdialogs) weiterlaufen.

Alternativ kann auch die Umgebungsvariable `IDM_ERRWIN` gesetzt werden.

Die Startoption überschreibt die Umgebungsvariable wenn beide gesetzt sind.

Bei Startoption und Umgebungsvariable kann *none* anstelle eines Dateinamens angegeben werden. Dies unterbindet sowohl die Anzeige von Fehlerdialogen und Fehlerausgaben in die Konsole, als auch das Anlegen einer Datei zum Protokollieren der Fehler.

Im Dateinamen können Platzhalter verwendet werden (siehe Kapitel „Platzhalter in Dateinamen“).

Hinweis

Bei gleichzeitiger Nutzung der Optionen **-IDMtracefile** oder **-IDMerrfile** (oder deren Umgebungsvariablen) kann die hier angegebene Datei auch ohne Inhalt bleiben oder sie wird überhaupt nicht erzeugt. Eventuelle Fehlermeldungen werden in diesem Fall in die Datei, welche bei der jeweils anderen Option angegeben ist, herausgeschrieben.

-IDMfatalneterrors <boolean>

Mit `-IDMfatalneterrors true` kann ein kompatibles Verhalten des DISTRIBUTED DIALOG MANAGERS (DDM) zu den IDM-Versionen vor A.05.01.d eingestellt werden, mit einem sofortigen Abbruch auf Client- und Server-Seite bei Netzwerk-, Protokoll- und Versionsfehlern. Das heißt, außer für lokale Applikationen werden keine *start*- und *finish*-Ereignisse mehr ausgelöst und es erfolgt kein Aufruf von **AppFinish** mehr.

Wenn die Benutzung einer Startoption nicht möglich ist, kann anstelle von **-IDMfatalneterros** die Option *DMF_FatalNetErrors* der Funktion **DM_Initialize** verwendet werden. *DMF_FatalNetErrors* hat Vorrang vor **-IDMfatalneterrors**.

Siehe auch

C-Funktion DM_Initialize

-IDMfont <integer>

Mit Hilfe dieser Option können die aktuell zu verwendenden Zeichensätze auf die angegebene Variante umgestellt werden.

-IDMformat <integer>

Mit Hilfe dieser Option kann die aktuell verwendete Variante von Formaten eingestellt werden.

-IDMindent <indent>[:<tabsize>[:<traceindent>]]

Diese Option bestimmt die Einrückungstiefe von Quelltexten und optional des Tracing. Zusätzlich kann die Ersetzung von Leerzeichen durch Tabulatoren eingestellt werden.

Für *<indent>*, das die Einrückungstiefe von Quelltexten definiert, und für das optionale *<traceindent>*, das die Einrückung für die Tracedatei bestimmt, sind ganze Zahlen *>=0* erlaubt. Für das optionale *<tabsize>*, das die Ersetzung von Leerzeichen durch Tabulatoren steuert, sind nur die Werte *0* und *8* zulässig.

Um z.B. eine Einrückung von drei Zeichen ohne Tabulator-Ersetzung zu bekommen ist *„3:0“* anzugeben, mit *„8:8“* erfolgt die Einrückung jeweils um einen Tabulator.

Hinweis

Die Einrückung für das Tracing kann auch am ***Setup***-Objekt eingestellt werden.

-IDMkeyboard <integer>

Über diese Option kann die aktuell verwendete Variante von Accelerators eingestellt werden.

-IDMlanguage <integer>

Mit Hilfe dieser Option kann die aktuell zu verwendende Sprache auf die angegebene Variante umgestellt werden.

-IDMno_yi_monitoring

Mit dieser Startoption kann das Aufrufen von Monitorfunktionen unterbunden werden, die mit **YiRegisterUserEventMonitor** installiert werden. Die Option kann bei der Fehlersuche hilfreich sein, wenn der Verdacht besteht, dass eine Monitorfunktion Fehler verursacht.

Anstelle der Startoption kann auch die Umgebungsvariable IDM_NO_YI_MONITORING angegeben werden.

Die Startoption überschreibt die Umgebungsvariable wenn beide gesetzt sind.

Alternativ kann die Option *.options[opt_yi_monitoring]* des ***Setup***-Objekts auf *false* gesetzt werden. Wurden Monitorfunktionen durch Startoption oder Umgebungsvariable ausgeschaltet, lassen sie sich nicht durch Setzen von *.options[opt_yi_monitoring] = true* einschalten.

-IDMobjdump_fkey <func_key_no>

Diese Startoption bewirkt, dass der Quelltext des aktiven Fensters in die Trace-Datei geschrieben wird, wenn die Funktionstaste mit der Nummer *<func_key_no>* gedrückt wird. Die Ausgabe des Quelltextes ist in der Trace-Datei mit den Trace-Codes *„DC“* und *„DR“* gekennzeichnet.

-IDMscale <integer>

Die Option **-IDMscale** bestimmt mit welcher Skalierung die Anwendung dargestellt werden soll. Dabei schaltet ein Wert von *0* die Skalierung aus. Die Angabe erfolgt in %.

Grundeinstellung:

Die aktuelle vom System verwendete Skalierung.

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

Einschränkung WINDOWS

Über diese Startoption kann die DPI-Awareness lediglich an- oder ausgeschaltet werden. Diese Option sollte unter Microsoft Windows allerdings nicht verwendet werden. Da DPI-Awareness eine Eigenschaft der Anwendung ist, sollte diese nur über eine Manifest-Datei spezifiziert werden. Das Verwenden der Startoption unter Windows resultiert in einer Warnung in der Trace- oder Logdatei.
Für Testzwecke kann die DPI-Awareness eingeschaltet (Wert: *1*) oder ausgeschaltet (Wert: *0*) werden, sollte jedoch nie im laufenden Betrieb umgestellt werden.

Einschränkung QT

Ob ein hier gesetzter Skalierungsfaktor auch wirklich angewandt wird, ist stark von der Desktop-Umgebung und deren Unterstützung für HighDPI abhängig.

Verfügbarkeit

Ab IDM-Version A.06.03.a

Siehe auchKapitel„HighDPI UnterstützungSupport“im Handbuch„Programmiertechniken“

-IDMsearchpath <searchpath>

Diese Option setzt den Suchpfad über den Dialog-, Modul-, Interface- und Binärdateien bei Imports mit **use** 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.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

Kapitel „Suchpfad für Interface-, Modul-, Dialog- und Binärdateien“ im Handbuch „Programmiertechniken“

-IDMserver

Mit Hilfe dieser Option kann die aktuelle Version der Fenstersystem-Schnittstelle abgefragt werden.

Diese Option funktioniert nur in den Motif-Versionen des Dialog Managers. In den Windows-Versionen liefert sie eine Fehlermeldung.

-IDMshowerror

Gibt auch interne, vom IDM behandelte Fehler in der Trace-Datei aus. Diese Fehler können normalerweise ignoriert werden. In manchen Situationen können sie aber Hinweise auf die Ursache anderer Fehler liefern.

-IDMsource <integer>

Mit Hilfe dieser Option kann die aktuelle Variante der Source-Definition für die Drag&Drop-Operationen eingestellt werden.

-IDMstrace

Das Tracefile wird im Safety-Modus verwendet. Zusätzlich muss das Tracing über die Option **-IDMtracefile <filepath>** eingeschaltet sein.

Um das Mitlaufen des Tracefiles auch bei längeren Anwendungssitzungen mit möglichst geringen Performanzeinbußen und Ressourcenverbrauch zu ermöglichen, wird in diesem Modus ein Tracing in einen begrenzten Ringpuffer, der im Speicher gehalten wird, durchgeführt. Der Inhalt des Ringpuffers wird beim Beenden der Anwendung in der Datei gespeichert.

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMstracefile <filepath>

Diese Option ist eine Kurzform für die Kombination der Optionen **-IDMtracefile <filepath>** zum Einschalten des Tracing und **-IDMstrace** zur Einstellung des Safety-Modus.

Im Dateinamen können Platzhalter verwendet werden (siehe Kapitel „Platzhalter in Dateinamen“).

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMstraceopts <string>

Diese Option aktiviert das Safety-Tracing und definiert gleichzeitig die Einstellungen dafür. Zusätzlich muss das Tracing über die Option **-IDMtracefile <filepath>** eingeschaltet sein.

Der String-Parameter beeinflusst die Einstellungen wie folgt:

Schablone	Einstellung
c<integer>	Bytes pro Zeile
h	Hierarchische Ausgabe: Erhaltung möglichst vieler Hierarchiestufen
l<integer>	Zeilenanzahl
r	Rotierende Ausgabe (Standard): Älteste Zeile wird durch neueste ersetzt
s<integer>	Längengrenze von Strings

Durch Konkatenation können mehrere Einstellungen gleichzeitig vorgenommen werden. Dabei ist die Reihenfolge beliebig.

Anstelle dieser Option kann auch die Umgebungsvariable `IDM_STRACEOPTS` genutzt werden. Die Startoption überschreibt die Umgebungsvariable wenn beide gesetzt sind.

Beispiel

Durch die Option **-IDMstraceopts l300c80** wird ein Safety-Tracing mit *300* Zeilen und *80* Bytes pro Zeile aktiviert.

Verfügbarkeit

IDM-Versionen A.05.01.g3, A.05.01.h, ab A.05.02.e

-IDMtarget <integer>

Mit Hilfe dieser Option kann die aktuelle Variante der Target-Definition für die Drag&Drop-Operationen eingestellt werden.

-IDMtile <integer>

Mit Hilfe dieser Option können die aktuell zu verwendenden Muster auf die angegebene Variante umgestellt werden.

-IDMtiledpi <integer>

Über diese Startoption wird der für Tiles anzuwendende DPI-Wert angegeben. Dieser DPI-Wert spielt dann eine Rolle, wenn Bilder für einen bestimmten DPI-Wert erstellt wurden und auch dementsprechend angezeigt werden sollen. Standardmäßig wird von *96* DPI ausgegangen.

Verfügbarkeit

Ab IDM-Version A.06.03.a

Siehe auchKapitel„HighDPI UnterstützungSupport“im Handbuch„Programmiertechniken“

-IDMtracefile <filename>

Diese Option schreibt eine Protokoll-Datei, in der alle Funktionsaufrufe des ISA Dialog Managers, alle Aufrufe vom ISA Dialog Manager an die Applikation und alle ausgeführten Regeln mitprotokolliert werden (siehe Kapitel „Tracing (Ablaufverfolgung)“ für weitere Informationen).

Durch Setzen der Umgebungsvariablen `IDM_TRACEFILE` auf einen Dateipfad kann dasselbe bewirkt werden.

Die Startoption überschreibt die Umgebungsvariable wenn beide gesetzt sind.

Im Dateinamen können Platzhalter verwendet werden (siehe Kapitel „Platzhalter in Dateinamen“).

-IDMtracetime <integer>

Diese Option schreibt zusätzlich Zeitstempel in die Protokoll-Datei, in der alle Funktionsaufrufe des Dialog Managers, alle Aufrufe vom Dialog Manager an die Applikation und alle ausgeführten Regeln mitprotokolliert werden. Dadurch ist es möglich, den absoluten oder relativen Zeitbedarf von Funktionen oder Regeln zu erkennen und dadurch entsprechende Tuningmaßnahmen einzuleiten.

Wertebereich

0

Im Tracefile werden keine Zeiten aufgeführt

1

Dieser Wert kennzeichnet den Starttime-Modus. Bei diesem Modus werden alle Start- und Endezeiten mitprotokolliert, der Zeitverbrauch für einen einzelnen Aufbau kann dann aus der Differenz berechnet werden. Dabei wird in diesem Modus ausschließlich die System- und die Benutzerzeit betrachtet.
In diesem Modus erscheinen im Tracefile am Zeilenanfang die Zeiten im Format [hh:mm:ss:uuu]:

- hh = Stunden
- mm = Minuten
- ss = Sekunden
- uuu = Millisekunden

2

Dieser Wert kennzeichnet den Tracetime-Modus. In diesem Modus wird der Zeitunterschied zum letzten mitprotokollierten Aufruf gegeben. In diesem Modus kann also relativ einfach erkannt werden, wie viel Zeit für einzelne Aktionen verbraucht wird.
In diesem Modus erscheinen im Tracefile die Zeitdifferenz zur letzten Trace-Ausgabe im Format [ss:uuu] am Zeilenanfang:

- ss = Sekunden
- uuu = Millisekunden

3

Dieser Wert kennzeichnet den Realtime-Modus. In diesem Fall wird für jede zu protokollierende Aktion die reale Zeit (Uhrzeit) im Tracefile abgelegt.
In diesem Modus erscheinen im Tracefile die Realzeit im Format [hh:mm:ss] am Zeilenanfang:

- hh = Stunden
- mm = Minuten
- ss = Sekunden

-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*.

Verfügbarkeit

Ab IDM-Version A.06.02.g

-IDMversion

Über diese Option kann die aktuell verwendete Version des Dialog Manager Laufzeitsystems (Runtime System) erfragt werden.

Die Version wird in der Form

k.vn.uv.pl<Zusatzinfo>

ausgegeben. Darin bedeuten:

k	Kennzeichnung der Versionsart (Großbuchstabe, in der Regel „A“)
vn	Versionsnummer, Major Release (zwei Ziffern)
uv	Unterversionsnummer, Minor Release (zwei Ziffern)
pl	Patchlevel (Kleinbuchstabe und optional eine Ziffer)

<Zusatzinfo> nur für interne Zwecke

2.1 Platzhalter in Dateinamen

Wenn für Startoptionen oder Umgebungsvariablen angegeben ist, dass Platzhalter im Dateinamen verwendet werden können, stehen dafür folgende Platzhalter zur Verfügung:

%Y	Jahr
%O	Monat
%D	Tag
%H	Stunde
%M	Minute
%S	Sekunde
%J	Tag des Jahres
%C	Eindeutige Zahl
%A	Anwendungsname
%T	ttyID
%U	UserID
%P	ProcessID

Die Platzhalter werden dann zur Laufzeit durch die entsprechenden Werte ersetzt.

2.2 Startoptionen zum Definieren von Codepages

Mit den Codepage-Optionen kann die Zeichencodierung definiert werden, die der IDM bei verschiedenen Operationen nutzt. Folgende Optionen stehen zur Verfügung, bei denen für <codepage> die in Kapitel „Codepage-Bezeichner“ genannten Bezeichner verwendet werden können:

Option	Beschreibung
-IDMcp_appl <codepage>	Definiert die Codepage, in der Anwendungsfunktionen Strings interpretieren und zurückgeben. Der bessere Weg ist allerdings, diese Codepage innerhalb der Anwendung mit DM_Control zu setzen.

Option	Beschreibung
-IDMcp_format <codepage>	Definiert die Codepage, in der Formatfunktionen Strings interpretieren und zurückgeben. Der bessere Weg ist allerdings, diese Codepage innerhalb der Anwendung mit DM_Control zu setzen.
-IDMcp_input <codepage>	Definiert die Codepage, die der IDM für die Interpretation von Dialog-, Modul-, Interface- und Init-Dateien verwendet, sofern die Dateien kein Erkennungsmerkmal (siehe Kapitel „Erkennungsmerkmale für Dateien“) haben.
-IDMcp_io <codepage>	Abkürzung für das gleichzeitige Setzen der Input- und Output-Codepage (**-IDMcp_input** und **-IDMcp_output**).
-IDMcp_output <codepage>	Definiert die Codepage, die der IDM für Ausgabedateien verwendet. Die Option bezieht sich auf Dialoge, Module, Interfaces, Init-, Log- und Trace-Dateien sowie stdout und stdin.

2.2.1 Codepage-Bezeichner

Bei den Codepage-Optionen des IDM können für den Parameter <codepage> folgende Bezeichner eingesetzt werden:

acp	ascii	cp437	cp850	cp1252
dec169	hp15	iso6937	iso8859	no_conv
utf8	utf16	utf16b	utf16l	winansi

2.2.2 Erkennungsmerkmale für Dateien

Text-Dateien können innerhalb der ersten 8 Bytes ein Erkennungsmerkmal enthalten, das die Zeichencodierung der Datei angibt. Ist ein Erkennungsmerkmal vorhanden, hat es für den IDM Vorrang vor einer Codepage-Option.

Der IDM erkennt folgende Erkennungsmerkmale am Dateianfang:

Erkennungsmerkmal	Zeichencodierung
Bytefolge 0xfeff	UTF-16 BE (Big Endian)
Bytefolge 0xffef	UTF-16 LE (Little Endian)
// UTF8 oder // utf8	UTF-8

Erkennungsmerkmal	Zeichencodierung
// 8859	ISO-8859-1
// 1252	CP1252

Die hexadezimalen Bytefolgen „feff“ und „ffef“ sind sogenannte „Byte Order Marks“ (BOM) die im Unicode-Standard festgelegt sind. Sie definieren die Reihenfolge des Bytes innerhalb eines Zeichencodes: „feff“ steht für die Reihenfolge „Big Endian“ mit den höherwertigen Bytes zuerst, „ffef“ für die Reihenfolge „Little Endian“ mit den niederwertigen Bytes zuerst.

Wenn mit **-IDMcp_output** oder **-IDMcp_io** die Ausgabe-Codepage auf UTF-8, UTF-16, ISO-8859 oder CP1252 gesetzt ist, schreibt der IDM das entsprechende Erkennungsmerkmal an den Dateianfang von Dialogen, Modulen und Interfaces.

Hinweis

Text-Editoren zeigen die Byte Order Marks „feff“ und „ffef“ in der Regel nicht an. Man kann sie aber sehen, wenn man die Dateien mit einem Hex-Editor öffnet.

2.3 Startoptionen im Simulationsprogramm

Folgende Optionen sind nur im **Simulationsprogramm** gültig:

+application <applicationID>

Diese Option ist nur zusammen mit den Optionen **+writefuncmap** und **+writeheader** gültig!

Bei Angabe dieser Option werden die Funktionsprototypen der im Objekt *applicationID* enthaltenen Funktionen herausgeschrieben.

-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.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

+/-builder

Diese Option bewirkt, dass der IDM bzw. PIDM im Builder-Modus läuft. Falls noch nicht geschehen, wird er als IDM Builder-Prozess im Hintergrund gestartet. Alle Anfragen zum Schreiben von Interface-, Binär-, Funcmap- oder Trampolin-Dateien werden an diesen IDM Builder Prozess weitergereicht. Der Prozess arbeitet im **Shared-Modules-Modus** um Zeit beim Nachladen von importierten Modulen zu sparen.

Für **-builder** erfolgt die Erkennung von Server- und Client-Modus anhand der Aktionen (**-writebin**, **-writeexport**…) während **+builder** explizit den Client-Modus aktiviert. Bei **-builder** wird automatisch vom Client-Modus ausgegangen, wenn es mit einer Aktion verwendet wird. Ohne Aktion wird im Builder-Prozess (Server) gearbeitet.

Verfügbarkeit

Versionen ab A.05.02.f; Plattformen MICROSOFT WINDOWS, UNIX/LINUX

-builderid <Builder-Bezeichner>

Normalerweise haben IDM Builder-Prozess und IDM bzw. PIDM, die den Prozess im Builder-Modus nutzen, denselben Vaterprozess (mit der gleichen Identifikationsnummer). Dies kann durch Angabe eines eigenen Builder-Bezeichners in Form eines Dateinamens **ohne** vorangestellten Pfad umgangen werden.

Verfügbarkeit

Versionen ab A.05.02.f; Plattformen MICROSOFT WINDOWS, UNIX/LINUX

-builderstop

Mit dieser Option kann ein laufender IDM Builder-Prozess vor Erreichen seiner Timeout-Zeit gestoppt werden.

Verfügbarkeit

Versionen ab A.05.02.f; Plattformen MICROSOFT WINDOWS, UNIX/LINUX

-buildertimeout <secs>

Mit dieser Option wird die Wartezeit des IDM bzw. PIDM im Builder-Prozess-Modus in Sekunden definiert. Nach Erreichen dieser Zeit beendet sich der IDM Builder-Prozess. Nach der eingestellten Wartezeit scheitern Verbindungsversuche und Aufrufe.

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.

Verfügbarkeit

Versionen ab A.05.02.f; Plattformen MICROSOFT WINDOWS, UNIX/LINUX

-classname <classname>

Diese Option beschränkt das Schreiben der Klassendefinitionen von USW-Klassen für das IDM-ECLIPSE-PLUGIN auf die angegebene Klasse.

Sie ist nur zusammen mit der Option **-writeclassdef** gültig.

-cleancompile
-cleancompile1

Diese Option löscht alle Interface- und Binärdateien für die geladenen Dialog- und Moduldateien sowie alle mit **use** importierten Untermodule.

Die Option **-cleancompile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-cleancompile1** performs the action only for the loaded dialog or module and **not** for submodules imported per **use**.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

Kapitel „Kompilieren von Interface- und Binärdateien für Imports mit use“ im Handbuch „Programmiertechniken“

-cobbasename <basefilename>

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.

Kann nur in Verbindung mit der Option **+/-writetrampolin** verwendet werden.

Verfügbarkeit

Ab IDM-Version A.06.02.g

-cobname <basefilename>

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 „.cbl“ wenn eine der Optionen **-ufcob**, **-mfviscob** oder **-mfviscob-u** angegeben wird, ansonsten „.cob“.

Kann nur in Verbindung mit der Option **+/-writetrampolin** verwendet werden.

Verfügbarkeit

Ab IDM-Version A.06.02.g

-compile
-compile1

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.

Um die Zieldateien auf jeden Fall, ohne Prüfung des Dateidatums, neu zu erzeugen, kann die Option **-recompile** verwendet werden.

To recreate the target files in any case, without checking the file date, the **-recompile** option can be used.

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.

Die Option **-compile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-compile1** performs the action only for the loaded dialog or module and **not** for submodules imported per **use**.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

Kapitel „Kompilieren von Interface- und Binärdateien für Imports mit use“ im Handbuch „Programmiertechniken“

-cpyname <basefilename>

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“.

Kann nur in Verbindung mit der Option **+/-writetrampolin** verwendet werden.

Verfügbarkeit

Ab IDM-Version A.06.02.g

-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.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

-mfviscob

Erzeugt COBOL Copy-Strecken für MICRO FOCUS VISUAL COBOL.

Kann nur in Verbindung mit den Optionen **+writeheader** und **+/-writetrampolin** verwendet werden.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

-mfviscob-u

Erzeugt COBOL Copy-Strecken für MICRO FOCUS VISUAL COBOL mit Unterstützung für *National Character* (Unicode, UTF-16).

Kann nur in Verbindung mit den Optionen **+writeheader** und **+/-writetrampolin** verwendet werden.

Verfügbarkeit

Nur COBOL-Schnittstelle für MICRO FOCUS VISUAL COBOL.

-noif

Mit dieser Option wird das Generieren der Interface-Dateien unterbunden.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

+/-profile <filepath>

Mit Hilfe dieser Option wird die angegebene Konfigurationsdatei geladen.

-profile	Konfigurationsdatei wird vor **DM_StartDialog** geladen
+profile	Konfigurationsdatei wird nach **DM_StartDialog** geladen

Siehe auch

Funktion DM_LoadProfile im Handbuch „C-Schnittstelle - Funktionen“

-recompile
-recompile1

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.

This option causes the interface and binary files to be written for the loaded module or dialog and for all loaded submodules imported per **use**.

Anders als bei der Option **-compile** werden die Zieldateien **immer** neu erzeugt. Es findet keine Datumsprüfung statt.

Unlike the option **-compile**, the target files are **always** regenerated. There is no date check.

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.

Possibly necessary directories resulting from the Use Path are created if required. The output directories of the target files can be controlled using the **-ifdir** and **-bindir** options.

Da die erzeugten Interface- und Binärdateien vorgegebene Dateiendungen (*.if* und *.bin*) besitzen, ist die Erzeugung dieser Dateien gemischt mit den Quellen möglich.

Since the generated interface and binary files have predefined file extensions (*.if* and *.bin*), it is possible to create these files mixed with the sources.

Achtung

Vorhandene Dateien werden überschrieben.

Existing files will be overwritten.

Die Option **-recompile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-recompile1** performs the action only for the loaded dialog or module and **not** for submodules imported per **use**.

Verfügbarkeit

Ab IDM-Version A.06.02.g

Siehe auch

Kapitel „Kompilieren von Interface- und Binärdateien für Imports mit use“ im Handbuch „Programmiertechniken“

+searchsymbol <Symbolname>

Ein Modul wird in einem Pfad gesucht, der über Umgebungsvariablen definiert wird. Wird eine entsprechende Datei in einem Verzeichnis gefunden, wird diese geladen und die weitere Suche abgebrochen.

Damit nun diese Umgebungsvariable beim Laden der Module beachtet wird, muss beim Erstellen der Interface-Datei zusätzlich die Option **+searchsymbol IDMLIB** angegeben werden, damit das Symbol „IDMLIB“ allen Dateinamen vorangestellt wird.

Allgemeine Syntax

```
idm +writeexport <Exportdatei-Name> +searchsymbol  <Umgebungsvariable>
  <Modul-Name>
```

Beispiel

```
idm +writeexport color.if +searchsymbol IDMLIB color.mod
```

Siehe auch

Kapitel „Modularisierung“ im Handbuch „Programmiertechniken“

-ufcob

Erzeugt COBOL Copy-Strecken für MICRO FOCUS COBOL.

Kann nur in Verbindung mit den Optionen **+writeheader** und **+/-writetrampolin** verwendet werden.

-userregistry

Diese Option kann zusätzlich zu **-writeole** angegeben werden, um ein OLE-Control nur für den aktuellen Benutzer zu registrieren. Die Registrierungsdaten werden dann unter HKEY_CURRENT_USER in die Windows-Registry geschrieben.

Verfügbarkeit

Ab IDM-Version A.06.01.g

+writebin <binaryfile>

Mit Hilfe dieser Option kann aus der ASCII-Dialog-Datei eine Binär-Datei erstellt werden.

Der Vorteil dieser Binär-Datei liegt vor allem in der wesentlich kürzeren Ladezeit und in der Nicht-Änderbarkeit durch den Endanwender.

-writeclassdef <xmlfile>

Durch diese Option wird in die angegebene Datei für alle registrieren USW-Klassen eine Klassendefinition geschrieben, die vom IDM-ECLIPSE-PLUGIN gelesen werden kann. Die Klassendefinition wird vom IDM-ECLIPSE-PLUGIN beispielsweise benötigt, damit Attribute von USW-Klassen bei der Eingabeunterstützung vorgeschlagen werden.

+/-writedialog <filename>

Diese Option schreibt einen Dialog bzw. ein Modul in seinem aktuellen Zustand als Textdatei nach „filename“. Die Zeichenkodierung (Codepage) der Datei kann mit der Option **-IDMcp_output** bzw. **-IDMcp_io** definiert werden (siehe Kapitel „Startoptionen zum Definieren von Codepages“). Ist keine dieser Optionen angegeben, dann wird der Dialog bzw. das Modul als ASCII-Datei gespeichert.

+writeexport <interfacefile>

Mit dieser Option kann der Entwickler automatisch die Interface-Datei aus seinem Modul erstellen. Dabei werden Kommentare, die mit ! ! vor den eigentlichen Objekten stehen, mit in die Interface-Datei übernommen. Auf diese Art und Weise kann eine Kommentierung der Dialogskripte auch für den Anwender eines Moduls verfügbar gemacht werden.

Siehe auch

Kapitel „Modularisierung“ im Handbuch „Programmiertechniken“

+writefuncmap <basefilename>

Diese Option erzeugt eine Header-Datei und eine C-Datei. Die Header-Datei enthält die Funktionsprototypen für alle im IDM-Dialog deklarierten Funktionen (siehe auch Option **+writeproto**) und den Prototyp für die Anbindungsfunktion. Die C-Datei enthält eine Funktionstabelle mit den im Dialog deklarierten Funktionen und die Anbindungsfunktion **BindFunctions_<Dialogbezeichner>**, die zur Anbindung der Funktionen an den Dialog aufgerufen werden muss.

Siehe auch

Handbuch „C-Schnittstelle - Grundlagen“

+writeheader <basefilename>

Die Startoption **+writeheader** kann verwendet werden, um Prototypen und Record-Definitionen für dynamisch angebundene Funktionen zu erzeugen.

Die Kommandozeile für den Simulator hat folgenden Aufbau:

```
pidm [+application <Name der Applikation>] +writeheader <basefilename>
  <dialogfile>
```

Wenn C-Funktionen vorhanden sind wird eine Header-Datei mit dem Suffix **.h** erstellt welche die entsprechenden Funktionstypen und Record-Definitionen beinhaltet. Sind COBOL-Funktionen mit Record-Parametern vorhanden, so werden die Copy-Strecken in die Datei mit der Endung **.cpy** geschrieben. Sind keine entsprechenden Funktionen vorhanden, erfolgt auch keine Generierung einer Datei.

Die Startoption ist eine Mischung der Startoptionen **+writefuncmap** und **+/-writetrampolin**, nur ohne Generierung von C- und COBOL-Code. Sie eignet sich somit nur für die dynamische Anbindung von Applikationsfunktionen.

Verfügbarkeit

Ab IDM-Version A.06.01.d

-writeole <basefilename>

Mit dieser Option generiert der ISA Dialog Manager die notwendigen Dateien, um einen OLE-Server beim System zu registrieren. Es werden eine **idl**-Datei zur Erzeugung der Typbibliothek und eine **reg**-Datei mit den Registrierungseinträgen erzeugt. Die Registrierungseinträge werden unter HKEY_LOCAL_MACHINE in die Windows-Registry eingetragen, sodass die Registrierung für alle

Benutzer gültig ist.

Um ein OLE-Control nur für den aktuellen Benutzer zu registrieren, kann zusätzlich die Option -userregistry angegeben werden.

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.

Siehe auch

Kapitel „Generierung der Schnittstelleninformation“ im Handbuch „OLE-Schnittstelle“

+writeproto <filename>

Diese Option erzeugt eine Datei mit Funktionsprototypen für alle Funktionen, die im IDM-Dialog deklariert sind. Die erzeugte Datei kann als C-Header-Datei für die Applikation verwendet werden um die Korrektheit von Funktionsaufrufen zu überprüfen.

Hinweis

Verwenden Sie die Option **+writefuncmap**, um nicht nur die Header-Datei, sondern außerdem eine C-Datei mit der Funktionstabelle und der entsprechenden Anbindungsfunktion zu generieren.

+writerefs <filename>

Mit dieser Option werden die Referenzierungen aller in einem Dialog oder Modul definierten Modelle in die angegebene Datei geschrieben. Dadurch kann geprüft werden, ob die Modelle im Dialog oder Modul tatsächlich verwendet werden. Benutzte Modelle haben mindestens eine Referenz.

Es ist zu beachten, dass Modelle ohne Referenzierungen eventuell trotzdem für Objekte verwendet werden, die innerhalb von Regeln dynamisch erzeugt werden. Daher sollte die Ausgabe dieser Option lediglich als Hinweis auf möglicherweise ungenutzte Modelle betrachtet werden.

Beispiel

```
idm +writerefs TestRefs.txt Test.dlg
```

+/-writetrampolin <basefilename>

Mit dieser Option generiert der ISA Dialog Manager die C- und COBOL-Module, die notwendig sind, um Dialoge in den Records verwendet werden mit einer Anwendung zu verbinden. Die erzeugten Dateien müssen mit der Anwendung übersetzt und gelinkt werden. Je nach Art der Funktionen, die Records verwenden, werden automatisch die entsprechenden Header- und Copy-Dateien für C und/oder COBOL generiert.

Syntax des Kommandos

```
idm -writetrampolin <ausgabedatei> <dialogdatei>
```

Zusammen mit **+/-writetrampolin** können folgende Optionen angegeben werden:

- -ufcob, -mfviscob und -mfviscob-u um Copy-Dateien für MICRO FOCUS COBOL bzw. MICRO FOCUS VISUAL COBOL zu erzeugen.
- **-cobbasename**, **-cobname** und **-cpyname** um die Basisnamen der erzeugten Dateien anzupassen.

3 Konfigurationsdatei

In der Konfigurationsdatei können konfigurierbare Variablen und Records vom Anwender umdefiniert werden.

Die Konfigurationsdatei kann mit DM_LoadProfile geladen werden.

Innerhalb der Konfigurationsdatei sind beliebig viele Einträge der folgenden Form erlaubt:

Syntax

```
<Variable> := <Wert> ;
<Record> . <Attribut> { [<Index>] } := <Wert> ;

// <Kommentar bis zum Zeilenende>
/* <Kommentar auch mehrzeilig> */
```

Die <Variablen> müssen mit **config** als konfigurierbar gekennzeichnet sein.

Bei den <Records> muss das <Attribut> *.configurable true* sein und das gewünschte Attribut darin als DM-internes oder als benutzerdefiniertes Attribut vorhanden sein.

Optional dürfen diese Attribute auch indiziert angesprochen werden. Sie müssen dann jedoch entsprechend im jeweiligen Record definiert worden sein.

Als <Wert> sind alle konstanten Werte erlaubt, die auch bei der Variablendefinition mit Initialisierung und bei der Initialisierung von benutzerdefinierten Attributen eingesetzt werden dürfen. Die Datentypen der jeweiligen Werte müssen selbstverständlich mit den jeweiligen Datentypen der Variablen und der Record-Attribute übereinstimmen.

Beispiel

Dialogskript

```
dialog Example

config variable string Name := "Jack";

color C1 "red";
color C2 "green";

model window WinMod
{
  .bgc C1;
}

record Settings
{
  .configurable true;
```

```
  boolean Mode[3];
    .Mode[1] := true;
    .Mode[2] := false;
    .Mode[3] := false;

  object Windowcolor shadows WinMod.bgc;
}
```

Konfigurationsdatei

```
// configuring a variable
Name := "Peter";
// configuring a record
Settings.Mode[2] := true
// the model is configured directly with shadows
Settings.Windowcolor := C2;
```

4 Umgebungsvariablen

Das Laden externer Information sowohl aus der Regelsprache als auch aus der Programmschnittstelle (siehe dazu auch die Funktionen DM_LoadDialog bzw. DM_LoadProfile im Handbuch „C-Schnittstelle - Funktionen“) sollte immer über Umgebungsvariablen erfolgen. Dazu gehören:

- externe Bilder eines Musters
- Schnittstellen- und Modulbeschreibungen
- Dialogdateien
- Profiles

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

`<Umgebungsvariable>:<Name der Dialogdatei>`

wobei die Umgebungsvariable ein beliebiger Pfad sein kann, unter dem die Dialogdatei gesucht werden soll. Dieser wird zum Ladezeitpunkt durchsucht und die erste aufgefundene Datei wird geladen.

Dies ist sehr leicht auf dem Zielrechner konfigurierbar.

Beispiel

Regelsprache:

```
tile Checkbox "IDM_IMAGEPATH:Check.gif";
```

Vor dem Start des Programms muss die Umgebungsvariable `IDM_IMAGEPATH` auf das Verzeichnis gesetzt werden, wo sich die GIF-Datei befindet.

- MICROSOFT WINDOWS

  ```
  set IDM_IMAGEPATH =d:\appl\bilder
  ```

- UNIX

  ```
  setenv IDM_IMAGEPATH /usr/local/appl/bilder
  ```

5 IDM Builder-Prozess

5.1 Verfügbarkeit

» Plattformen: Der Builder-Prozess-Modus ist für MICROSOFT WINDOWS und UNIX bzw. LINUX verfügbar, aber nicht für VMS.

» Versionen: ab A.05.02.f

5.2 Der Builder-Prozess-Modus

Bei komplexen, modularisierten IDM-Anwendungen wird der Zeitbedarf für das Erstellen der Anwendung mit **make** oder anderen Buildmanagement-Werkzeugen maßgeblich durch die Ladezeiten der Module und Dialoge bestimmt.

Jeder Aufruf der Simulation (IDM bzw. PIDM) zum Erstellen einer Interface-, Binär-, Funcmap- oder Trampolin-Datei lädt notwendigerweise die angegebene Dialog- oder Moduldatei. Besteht die Anwendung aus einer Vielzahl von Modulen mit wiederum vielen, tiefen Import-Abhängigkeiten, werden lange Ladezeiten hauptsächlich durch das Nachladen der importierten Module verursacht.

Eine deutliche Reduzierung dieser Nachladezeit kann durch Nutzung des IDM bzw. PIDM im **Builder-Prozess-Modus** erreicht werden.

Dabei läuft ein IDM Simulations-Programm im Hintergrund im Builder-Prozess-Modus als **Server** und wartet auf Anfragen um Interface-, Binär-, Funcmap- und Trampolin-Dateien zu erstellen. Dieser IDM Builder-Prozess läuft dabei im **Shared-Modules-Modus**, der normalerweise durch Setzen der Umgebungsvariable DM_SHARED_MODULES aktiviert wird. Dabei erfolgt die Wiederverwendung von importierten Modulen anhand der Gleichheit des Interface-Namens und nicht wie sonst üblich über den Import-Bezeichner und den übergeordneten Import. Dies führt zu einer deutlichen Reduzierung der Ladezeit bei Modulen mit vielen Imports.

Die während der Anwendungserstellung notwendigen IDM- bzw. PIDM-Aufrufe verbinden sich lediglich als **Client** an den IDM Builder-Prozess. Dabei werden die notwendigen Argumente an den Server geschickt und der Rückgabestatus weitergereicht.

Der IDM Builder-Prozess muss nicht manuell gestartet werden, sondern startet bei korrekter Verwendung (Option **+builder**) von selbst. Nach einer einstellbaren Timeout-Zeit (Option **-buildertimeout <secs>**) beendet er sich automatisch. Er kann aber auch vor Erreichen der Timeout-Zeit manuell gestoppt werden (Option **-builderstop**).

Dadurch sollte die bisherige Verwendung des IDM bzw. PIDM relativ einfach und problemlos auf die Nutzung im Builder-Prozess-Modus umstellbar sein.

5.2.1 Beispiel

Makefile mit einfachem modularisiertem Dialog

```
PIDM=pidm
IDMARGS=-IDMenv MODPATH=if;.
IDMARGS_WRITEBIN=$(IDM_ARGS) +searchsymbol MODPATH –writebin
IDMARGS_WRITEIF=$(IDM_ARGS) –writeexport
all:: bin/dialog.dlg
bin/dialog.dlg: dialog.dlg if/defaults.if
    $(PIDM) $(IDMARGS_WRITEBIN) $@ dialog.dlg
if/defaults.if: defaults.mod
    $(PIDM) $(IDMARGS_WRITEIF) $@ dialog.dlg
bin/defaults.mod: defaults.mod
    $(PIDM) $(IDMARGS_WRITEBIN) $@ defaults.mod
```

Umgestelltes Makefile

Die Aktivierung des Builder-Prozess-Modus ist unterstrichen

```
PIDM=pidm
IDMARGS=-IDMenv MODPATH=if;. +builder
# Windows:
# IDMARGS=-IDMconsole -IDMenv MODPATH=if:. +builder
IDMARGS_WRITEBIN=$(IDM_ARGS) +searchsymbol MODPATH –writebin
IDMARGS_WRITEIF=$(IDM_ARGS) –writeexport
all:: bin/dialog.dlg builderstop
bin/dialog.dlg: dialog.dlg if/defaults.if
    $(PIDM) $(IDMARGS_WRITEBIN) $@ dialog.dlg
if/defaults.if: defaults.mod
    $(PIDM) $(IDMARGS_WRITEIF) $@ dialog.dlg
bin/defaults.mod: defaults.mod
    $(PIDM) $(IDMARGS_WRITEBIN) $@ defaults.mod
builderstop::                       # optional
    $(PIDM) –builderstop
```

5.2.2 Nutzungshinweise

Für die Wiederverwendung des gleichen IDM Builder-Prozesses (also des gleichen IDM bzw. PIDM im Builder-Prozess Server-Modus) muss die Prozessnummer des Vaterprozesses des IDM- bzw. PIDM-Clients identisch sein. Ansonsten kann mit der Option **-builderid** gearbeitet werden und der IDM Builder-Prozess manuell gestartet und gestoppt werden. Sinnvollerweise sollten IDM-spezifische Bauschritte direkt nacheinander erfolgen um die Wiederverwendung des IDM Builder-Prozesses innerhalb der einstellbaren Timeout-Zeit zu ermöglichen.

Initial läuft normalerweise kein IDM Builder-Prozess. Er wird mit der Option **+builder** automatisch gestartet und enthält dabei die Umgebung des startenden Prozesses. Bei der Kommunikation

zwischen Builder-Client und -Server werden die mit **-IDMenv** gesetzten Umgebungsvariablen und das Arbeitsverzeichnis des Builder-Clients weitergegeben. Diese Variablen überdecken damit die im IDM Builder-Prozess gesetzten Variablen. Das Umsetzen des Arbeitsverzeichnisses sorgt für eine identische Situation bei Client und Server. Kommt keine Verbindung zwischen Client und Server zustande, führt dies zu typischen Fehlermeldungen der Art *„… can't open/connect builder pipe …"*.

Der IDM Builder-Prozess ist nicht dafür gedacht, den Build zu parallelisieren. Zu einer Zeit kann sich immer nur ein Client mit dem IDM Builder-Prozess verbinden.

Wird im IDM Builder-Prozess während des Ladevorgangs die maximal mögliche Anzahl von Module erreicht (ca. 4.000 Module können gleichzeitig geladen sein), kommt es zu einem Neustart des IDM Builder-Prozesses und der Meldung *„[I: restart build server]"*. Dabei gehen die geladenen, wieder verwendbaren Import-Module verloren.

5.2.3 Hinweise zur Nutzung mit dem IDM Eclipse Plugin

Für die Nutzung des Builder-Prozess-Modus mit dem MAKEGEN PLUGIN sollten die Optionen **-IDMconsole** und **+builder** verwendet werden. Sie ermöglichen es den Eclipse CDT (C/C++ Development Tooling) bei Makefile-Projekten, IDM-Fehlermeldungen aus IDM- bzw. PIDM-Aufrufen durch den Fehlerparser des IDM ECLIPSE PLUGINS zu prüfen.

5.2.4 Besonderheiten

- Muss erst ein IDM Builder-Prozess gestartet werden, kommt es zu einer kurzen Verzögerung da auf den Prozessstart gewartet werden muss.
- Beim Bau-Lauf können Wartezeiten entstehen, die hervorgerufen werden durch das verzögerte Beenden des Builder nach einem Timeout bzw. durch das Warten des Make-Prozesses auf das Beenden aller Kindprozesse bzw. das Schließen der Ausgabekanäle. Der IDM Builder-Prozess kann in diesem Fall einfach durch Aufruf von IDM bzw.PIDM mit der Option **-builderstop** beendet werden.
- Die Kommunikation zwischen IDM bzw. PIDM im Builder-Prozess Client-Server-Modus geschieht über eine **benannte Pipe** (unter UNIX typischerweise im */tmp*-Verzeichnis). Entsprechende Sicherheitseinstellungen und Zugriffsrechte sollten also gesetzt sein. Ein forciertes Beenden von IDM bzw. PIDM – z.B. mit `kill -9` – sollte vermieden werden um das Aufräumen der Pipe-Dateien sicherzustellen.
- Bei der Nutzung von **+builder** ist das Umleiten der Ausgabe von Trace- oder Log-File (z.B. über die Optionen **-IDMerrfile** und **-IDMtracefile**) wirkungslos, da der IDM Build-Prozess ohne Umleitung gestartet wird.
- Unter Microsoft Windows erscheinen Fehlermeldungen des IDM bzw. PIDM normalerweise als Messagebox und andere Ausgaben in der IDM Log-Datei. Um Fehlermeldungen und Ausgaben in einer Konsole zu erhalten, kann die Option **-IDMconsole** verwendet werden. Diese Option wird beim Start an den IDM Builder-Prozess weitergegeben. Damit sollte ein Bau-Prozess möglich sein, dessen relevante Fehlermeldungen und Ausgaben an **stdout** bzw. **stderr** weitergereicht werden.

5.3 Startoptionen des Builder-Prozesses

IDM und PIDM

- **+/-builder**
  Starten des IDM bzw. PIDM im Builder-Modus
- **-builderid <Builder-Identifikator>**
  Angabe eines Identifikators für den IDM Builder-Prozess
- **-builderstop**
  Beenden des IDM Builder-Prozesses
- **-buildertimeout <secs>**
  Wartezeit nach der sich der Builder-Prozess automatisch beendet

Siehe

Kapitel „Startoptionen im Simulationsprogramm“ für die Beschreibung der Optionen.

IDM-Bibliothek

- **-IDMconsole** (MICROSOFT WINDOWS)
  Umleiten von Log-File und Fehlermeldungen nach STDOUT

Siehe

Kapitel „Startoptionen“ für die Beschreibung der Optionen.

6 Fehlermeldungen des IDM

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 weitere 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 eine „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 Hilfen zur Fehleranalyse

Verfügbarkeit

Die in diesem Kapitel beschriebenen Hilfen zur Fehleranalyse stehen für die DM Versionen A.05.01.g3 und A.05.01.h sowie ab A.05.02.e zur Verfügung.

7.1 Überblick

Die hier beschriebenen Hilfsmittel haben das Ziel, Entwickler in besonderen Situationen, wie Interpreter-Fehlermeldungen, Abstürzen[1] einer Anwendung oder Assfail (Fatal Error) Meldungen durch zusätzliche Informationen bei der Analyse und Einordnung von Fehlern zu unterstützen. Die folgende Tabelle enthält eine grobe Klassifizierung möglicher Fehler nach typischen Fehlerstellen und ihren wahrscheinlichen bzw. möglichen Auswirkungen. Die Auswirkung, dass ein Fehler unentdeckt bleibt, zu einer veränderten oder fehlerhaften Funktionalität führt oder in eine andere Komponente hineingetragen wird, ist in der Tabelle nicht explizit erwähnt.

Fehlerstelle	Beschreibung	Auswirkung
Regelcode	Fehler im Regelcode einer IDM-Anwendung, z.B. ungültige Zugriffe auf ein Attribut oder Objekt, Typfehler oder uninitialisierte Werte.	*** Eval Error im Log- oder Tracefile [E: …] oder [W: …] Meldungen, YE-Fehlercodes im Tracefile
Applikationsfunktion	Verschiedene Fehlerarten, z.B. Zugriffsverletzung, beschädigte Speicherverwaltung, die sich in letzter Konsequenz unter Umständen aber erst bei der Regelverarbeitung oder in der IDM-Bibliothek zeigen.	eventuell Absturz
IDM-Bibliothek	Innerhalb des IDM sind fehlerhafte Zustände normalerweise durch eine Assertion abgesichert. Allerdings kann sich ein Fehler durchaus auf die Regelverarbeitung auswirken oder wie ein Applikationsfehler zeigen.	FATAL ERROR im Log- oder Tracefile [E: …] oder [W: …] Meldungen im Tracefile

[1] Mit einem Absturz ist dabei eine unbehandelte oder nicht abfangbare Fehlersituation gemeint, z.B. ungültiger Speicherzugriff, Division durch *0*.

Fehlerstelle	Beschreibung	Auswirkung
Externe Funktionen	Externe, vom IDM unabhängige Funktionen, z.B. in einem anderen Thread oder einer anderen Bibliothek, kann Fehler enthalten und diese z.B. durch Überschreiben von Speicher oder fehlerhafte Synchronisierung in andere Bereiche hineintragen.	z.B. Absturz

Im Folgenden werden die Erweiterungen zur Fehleranalyse und ihr Einsatzzweck erläutert.

Anmerkungen

» Grundsätzlich sollen die Erweiterungen keinen Debugger oder ein Werkzeug wie „Dr. Watson“ unter MICROSOFT WINDOWS ersetzen. Sie sind als Ergänzung zu verstehen, um sinnvolle Informationen zu liefern, die von den genannten Werkzeugen nicht geliefert werden.

» Wenn zuvor bereits schwerwiegende Fehler aufgetreten sind, kann die Richtigkeit der ausgegebenen Informationen nicht vollständig garantiert werden.

7.1.1 Safety-Tracing

Um den Nutzen des Tracing im IDM auch auf Situationen auszudehnen, in denen ein Fehler erst nach längerer Laufzeit einer Anwendung auftritt, kann der Safety-Modus des Tracing benutzt werden.

Normalerweise führt das ständige Mitlaufen des Tracing zu einer deutlichen Performanzeinbuße und einer sehr großen, unhandlichen Tracedatei. Im Safety-Modus erfolgt das Tracing in einen Ringpuffer mit begrenzter Zeilenzahl und Zeilenlänge, der im Arbeitsspeicher gehalten wird. Erst beim Beenden einer Anwendung oder bei einem Fehlerereignis wird dieser Puffer in die Tracedatei geschrieben.

Weitere Informationen zum Safety-Tracing sind bei den Startoptionen **-IDMstrace**, **-IDMstracefile** und **-IDMstraceopts** im Kapitel „Startoptionen“ sowie im Kapitel „Safety-Tracing“ zu finden.

7.1.2 Dumpstate

Bei Abstürzen, einem „FATAL ERROR“ oder einer „Error in Eval“ Meldung hat man ohne Tracing bei Anwendungen, die Binärdateien und die IDM-Laufzeitblibliothek nutzen, nur wenige Anhaltspunkte. Der Dumpstate stellt eine Zustandsinformation von IDM-Spezifika dar. Dazu gehören die Aufrufliste (Callstack), gemerkte Fehlercodes (Errorstack), Ereignisse sowie Hinweise über die Anzahl von IDM-Objekten und die Speichernutzung durch den IDM.

Primäres Ziel ist es, in einer Fehlersituation möglichst viele Information auszugeben, um so eine Bewertung des Fehlers zu ermöglich und die Ursachenforschung zu erleichtern. Es können nur Informationen gesammelt werden, die dem IDM bekannt sind, keine Informationen über Applikations- oder Fremdfunktionen. Das Schreiben des Dumpstate kann auch programmatisch über die Regelsprache oder eine DM-Schnittstellenfunktion erreicht werden.

Wichtiger Hinweis

Sicherheitsrelevante Informationen, z.B. Passwörter, die in ein Login-Fenster eingegeben werden, sollten nach ihrer Verwendung von der Anwendung gelöscht werden, damit sie nicht über einen Dumpstate ausgegeben und dadurch zugänglich gemacht werden.

Weitere Informationen zur Dumpstate-Ausgabe sind bei den Startoptionen **-IDMcallstack**, **-IDMdumpstate** und **-IDMdumpstateseverity** im Kapitel „Startoptionen“ sowie im Kapitel „Dumpstate (Zustandsinformationen)“ zu finden.

7.2 Exception-Catcher

Um bei einem Absturz das Safety-Tracing wie auch den Dumpstate herausschreiben zu können, wird ein globaler Exception-Catcher für eine Anwendung angemeldet. Der IDM installiert im Bootstrap einen globalen Exception-Catcher (für Structured Exceptions unter MICROSOFT WINDOWS, signal-Handler unter UNIX) für abfangbare Ausnahmen.

Beim Auftreten einer Exception gibt der IDM einen „FATAL ERROR“ mit der Exception-Nummer aus. Wenn notwendig und eingestellt, werden Dumpstate und Safety-Tracefile geschrieben. Der Exception-Catcher reicht die Exception dann wie gewohnt weiter, um so die weitere Behandlung durch das Betriebssystem („core dump“ unter UNIX/LINUX oder das Schreiben einer Dump-Datei unter MICROSOFT WINDOWS durch „Dr. Watson“) zu ermöglichen.

Hinweis

Beim Einrichten eigener Exception-Handler ist zu beachten, dass diese eine Exception weiterreichen.

Die Einrichtung eines Exception-Catcher kann mit der Startoption **-IDMcatchexceptions** (siehe Kapitel „Startoptionen“) unterbunden werden.

8 Dumpstate (Zustandsinformationen)

Verfügbarkeit

Die Dumpstate-Ausgabe steht für die DM Versionen A.05.01.g3 und A.05.01.h sowie ab A.05.02.e zur Verfügung.

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

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

Die Dumpstate-Information ist immer zwischen *„*** DUMP STATE BEGIN ***“* und *„*** DUMP STATE END ***“* eingeschlossen und kann folgende, in den nächsten Unterabschnitten detailliert beschriebenen, Abschnitte aufweisen:

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

Um die Ausgabe möglichst klein zu halten, erfolgt sie normalerweise in gekürzter Form. Grundsätzlich immer erfolgt eine Kürzung von IDM-Strings (in "…") auf maximal 40 Zeichen. Ihre Gesamtlänge wird in [ ] angehängt. Bytegrößen-Angaben erfolgen gekürzt auf Kilo-, Mega oder Gigabyte (k/m/g).

Der Dumpstate wird normalerweise automatisch in die Log- bzw Tracedatei herausgeschrieben wenn eine der in der folgenden Tabelle genannten Situationen eintritt. Dabei beeinflusst die Situation auch den Inhalt des Dumpstates.

Fehlerart	Ausgegebene Abschnitte
Der Regelinterpreter meldet einen „EVAL ERROR“.	Errorstack, Callstack, Events.
Die IDM-Bibliothek erkennt einen eigenen Fehler und meldet einen „FATAL ERROR“.	Alle.
Eine abfangbare Exception (z.B. Access Violation, Stack Overflow, Division by Zero) tritt auf und der IDM Exception-Catcher ist aktiv.	Alle, zusätzlich wird noch der Exception-Code (*„Microsoft Windows“*) bzw. die Signal-Nummer (UNIX) vorneweg ausgegeben. Unter UNIX wird ab IDM-Version A.05.02.e bei einem INT-Signal kein vollständiger Dumpstate mehr ausgegeben, sondern nur noch die Abschnitte Stack, Errors, Process und Events.
Normales Beenden über **DM_ShutDown()** aber Hinweise zu aufgetretenen Fehlern liegen vor.	Entsprechend den Hinweisen.

Wichtiger Hinweis

Sicherheitsrelevante Informationen, z.B. Passwörter, die in ein Login-Fenster eingegeben werden, sollten nach ihrer Verwendung von der Anwendung gelöscht werden, damit sie nicht über einen Dumpstate ausgegeben und dadurch zugänglich gemacht werden.

Das Schreiben des Dumpstate kann auch explizit durch die Builtin-Funktion **dumpstate()** bzw. die Schnittstellenfunktion **DM_DumpState()** erfolgen. Zusätzlich gibt es über die Optionen **-IDMdumpstate** und **-IDMdumpstateseverity** die Möglichkeit, die Ausgabe unabhängig von der Fehlerart zu beeinflussen bzw. die Fehlerart zu definieren, bei der eine Dumpstate-Ausgabe erfolgt.

In Anbetracht der großen Anzahl an komplexen Informationen, die beim Dumpstate ermittelt werden, kann ein Absturz innerhalb der Dumpstate-Funktionalität nicht abgefangen bzw. ausgeschlossen werden.

Es folgt eine detaillierte Beschreibung der einzelnen Abschnitte die im Dumpstate vorkommen.

8.1 Process

Dieser Bereich beinhaltet die Process-ID, die Nummer des Thread in dem der Dumpstate aufgerufen wird sowie die Thread-Nummer des IDM-Haupt-Thread. Wird der Dumpstate nicht im IDM-Haupt-Thread aufgerufen steht an erster Stelle der Hinweis:

```
ATTENTION: not in IDM main thread!
```

Dann ist Vorsicht geboten, deutet es doch auf einen Applikationsfehler, einen Fehler in einer externen Funktion oder die unsachgemäße Benutzung des IDM hin. Es ist zu beachten, dass der Callstack nur die Aufrufliste des IDM-Haupt-Thread beinhaltet!

```
PROCESS:
pid=2984, thread=4080, IDM-thread=4080, date=2009-11-10, time=16:20:38
```

8.2 Errors

Listet alle aktuell gesetzten Fehlercodes auf. Siehe hierzu auch Schnittstellenfunktion DM_QueryError im Handbuch „C-Schnittstelle - Funktionen“.

```
ERRORS:
#0: IDM-E-UnkAttr: Attribute not available for this type of object
[object/thing:639]
```

8.3 Callstack

Der Callstack (Aufrufliste) ist das wichtigste Mittel um zu erkennen, ob bei einem Absturz ein Fehler in einer Applikationsfunktion, einer Regel oder Methode, in der IDM-Bibliothek oder einer externen Funktion aufgetreten ist.

Der Callstack beinhaltet alle Aufrufe von Regeln, Methoden, eingebauten Funktionen, DM-Schnittstellenfunktionen sowie die vom IDM aufgerufenen Applikationsfunktionen. Dazu werden die Parameterwerte ausgegeben. Allerdings kann nicht in jeder Situation eine korrekte String-Codierung vorgenommen werden, um Auswirkungen auf die Performanz möglichst gering zu halten.

Für Regeln werden zusätzlich das this-Objekt, der Dateiname in der sich die Regel befindet, für ASCII-Dialoge die Startzeile der Regel, sowie eine %-Angabe, welche die ungefähre Position im Zwischencode der Regel darstellt, mit ausgegeben. Für die oberste Regel auf dem Stack werden außerdem die Werte der lokalen Variablen (locvars-Liste) sowie der Inhalt des Werte-Stack (valstack), der für die Ausdrucksauswertung benötigt wird, aufgelistet.

```
STACK:
#0: rule D.AfterError, this=dialog D, file=dumping.dlg:68+39%
  locvars=[dialog D, "initvar2", nothing]
  valstack=[100]
#1: rule on dialog start, this=dialog D, file=dumping.dlg:78+35%
#2: DM_StartDialog(dialog D, null)
```

Kürzungen

Es werden nur die obersten 20 und untersten 20 Einträge aufgelistet

Wichtig

Findet sich die Meldung *„ATTENTION: not in IDM main thread!“* am Anfang der Dumpstate-Ausgabe ist Vorsicht geboten. Der Callstack zeigt nur den Aufruf-Stack des IDM-Haupt-Thread an!

8.4 Events

In diesem Abschnitt werden die aktuell abgearbeiteten Ereignisse (THISEVENTS) und die Ereignisse, die noch in einer Warteschlange (EVENT QUEUE) stehen und als nächstes abgearbeitet würden, aufgelistet.

Die Quelle (source) gibt dabei die Ereignisauslösung an: **dialog**, **setval**, **destroy** oder **external**. Sie gibt Aufschluss darüber, ob es sich um ein vom Anwender/System ausgelöstes Ereignis handelt, es durch eine Wertänderung eines Attributs ausgelöst wurde, es sich um das Zerstören eines Moduls oder um ein externes Ereignis handelt.

Das Objekt, das der Empfänger des Ereignisses ist, wird angezeigt, ebenso die Warteschlange, der Datenwert und die Argumente sowie bei THISEVENTS die spezifischen Attribute (z.B. *.x*, *.y*).

```
THISEVENTS:
#0: source=dialog, object=dialog D, event=start
EVENT QUEUE#1:
#0: source=dialog, object=window D.Wi, event=activate
#1: source=external, object=pushbutton D.Wi.Pb, data=1, args=["EvData"]
```

Kürzungen

Maximal 10 Ereignisse werden aufgelistet

8.5 Usage

Dieser Abschnitt ist aus drei Tabellen aufgebaut:

- Informationen über die Anzahl der IDM-Objekte, deren Verteilung auf Defaults, Modelle und Instanzen, ihre Gesamtanzahl und deren ungefährer Speicherbedarf im IDM-Kern (nicht der dargestellten Oberflächenelemente). Diese Tabelle ist aufsteigend nach den Spalten **alloc** und **count** sortiert.
- Informationen über Dialoge und Module. Die Anzahl aller Dialoge und Module wird ausgegeben sowie die Summe der darin enthalten Objekt-Slots, der für die Slotverwaltung benötigte Speicher, die Gesamtanzahl der Bezeichner und der dafür notwendige Speicherplatz. In den **<maximum>**-Zeilen werden die Dialoge bzw. Module mit den größten Werten in den einzelnen Spalten aufgeführt. Dadurch sind Dialoge bzw. Module leicht identifizierbar, bei denen keine Objekte mehr angelegt werden können oder deren Platz für Bezeichner knapp wird.
- Die dritte Tabelle ist informativer Art und enthält Anzahl und Speichergröße der Regel-Frames, die vom Regelinterpreter genutzt werden, sowie Anzahl und Speichergröße von Puffern, die für das Speichern von Zwischenergebnissen benötigt werden.

```
              class  defaults    models instances     count     alloc
------------------ --------- --------- --------- --------- ---------
        thisevclass         0         0         1         1        84
          clipboard         0         0         1         1       100
```

```
         setupclass         0         0         1         1       140
        accelerator         1         1         2         4       336
             module         0         0         1         1       628
         pushbutton         1         0         1         2       752
             dialog         0         0         1         1       783
               text         1         1         7         9       828
           edittext         1         0         1         2       980
             window         1         1         1         3        2k
               rule         2         3         5        10       19k
            <total>         7         6        22        35       25k

   module     count     slots     alloc    labels labelsize name
--------- --------- --------- --------- --------- --------- -----------------
--
   dialog         1        27        3k        29        3k
   module         1         8        3k         1        3k
<maximum>                  27        3k        29        3k dialog D

   frames   f-alloc   scratch   s-alloc    values   v-alloc
--------- --------- --------- --------- --------- ---------
        2       22k         4        4k        69       17k
```

Die Spalten **values** und **v-alloc** der dritten Tabelle werden in den IDM-Versionen A.05.01.g3 und A.05.01.h nicht ausgegeben.

Kürzungen

Maximal 126 Klassen werden aufgelistet. Es werden keine Objektklassen-bezogenen Allokierungsgrößen ermittelt (Spalte enthält nur Nullen).

8.6 Memory

Dieser Abschnitt enthält die Größe des allokierten Heap-Speichers und gibt damit Anhaltspunkte über den Speicherverbrauch der gesamten Anwendung. Diese Funktionalität ist nur unter Windows oder mit eingeschaltetem IDM-Memory-Debugging verfügbar. Ermittelt wird die Summe der allokierten Speicherblöcke über **HeapWalk()** sowie zusätzlich über _heapwalk für die C-Runtime. Der vom IDM direkt genutzte Heap ist mit „*“ markiert, der Default-Heap des Prozesses mit „P“.

```
MEMORY:
        heap     alloc     other
- ---------- --------- ---------
*        crt      386k       56k
  0x02CC0000        4k       61k
  0x02BA0000        5k      167k
  0x029C0000      386k      613k
  0x02A10000        9k       56k
  0x006F0000        7k      999k
```

```
  0x00340000       12k       52k
  0x00630000       11k       53k
P 0x007B0000      137k      797k
     <total>      568k        3m
```

Kürzungen

Maximal die ersten 20 Heaps werden angezeigt.

Hinweis

Es ist zu beachten, dass die Heaps auch IDM-fremde Speicherallokierungen beinhalten, die z.B. von der Applikationsfunktion, Systemroutinen oder externe Funktionen (DLL, In-Process-OLE-Control) genutzt werden.

8.7 Slots

Die Ausgabe der Slots dient vornehmlich dem Erkennen von Fehlern in der IDM-Bibliothek, die mit Referenzierung sowie Freigabe und Zerstören von Objekten zusammenhängen. Es werden Objekt-Slots aufgelistet, die nicht vollständig entfernt werden konnten oder verdächtig hohe Referenzierungs- und Locking-Zähler aufweisen.

```
SLOTS:
        slot       class       refs locks drop hull object
------------ ----------- ---------- ----- ---- ---- --------------------
[0x00020028]      window          1     1    1    0 window D.WINDOW:[1]
```

Kürzungen

Maximal 20 Slots werden aufgelistet. Alle gesperrten Slots werden ungekürzt ausgegeben.

Wird für die Dumpstate-Ausgabe die Startoption, Builtin- oder Schnittstellenfunktion mit dem Parameter *dump_locked* (siehe Startoption **-IDMdumpstate <enum>** im Kapitel „Startoptionen“) verwendet, werden zusätzlich alle Attributewerte von gesperrten (locked) Objekten ausgegeben. Attributwerte von Ressourcen, Regeln und Funktionen können nicht ausgegeben werden.

8.8 Visible Objects

In diesen Abschnitt werden alle sichtbaren Objekte herausgeschrieben inklusive der Werte ihrer vordefinierten Attribute und sichtbaren Kindobjekte. Der Abschnitt soll bei Darstellungsproblemen einen Abzug möglichst aller relevanten Objekte und Attribute liefern um dadurch beispielsweise Probleme schneller reproduzieren zu können.

```
VISIBLE OBJECTS:
window Wi {
```

```
  .repos_id "";
  .userdata nothing;
  .visible true;
  .sensitive true;
  .navigable true;
  .fgc null;
  .bgc null;
  .font null;
:
}
```

In den Modus *dump_full* und *dump_uservisible* (siehe Startoption **-IDMdumpstate <enum>** im Kapitel „Startoptionen“) werden alle sichtbaren Objekte ausgegeben, die direkt unter einem Dialog bzw. Modul hängen. Zu diesen Objekten werden alle vordefinierten und benutzerdefinierten Attribute inklusive aller sichtbaren oder unsichtbaren Kindobjekte und Kind-Records ausgegeben.

Grundsätzlich können keine Ressourcen, Regeln, Methoden und Funktionen ausgegeben werden.

Kürzungen

Nur die obersten sichtbaren Objekte werden ausgegeben, ohne Wert/Kindobjekte.

8.9 Beispiel

Konstellation

Eine IDM-Anwendung startet über C einen eigenen Thread, der nach etwa 10 Sekunden abstürzt, da er auf eine ungültige Speicheradresse zugreift. Im Thread werden etwa 5 x 10MB Speicher allokiert. Der IDM legt währenddessen laufend Fenster und Records an, wobei zwar die Fenster wieder zerstört werden, die Records aber nicht.

Das Ende der Logdatei sieht dann etwa wie folgt aus:

```
FATAL ERROR: Exiting due to exception 0xC0000005 (ACCESS VIOLATION)
*** DUMP STATE BEGIN ***

ATTENTION: not in IDM main thread!

PROCESS:
pid=3024, thread=5176, IDM-thread=4684, date=2009-11-16, time=16:58:52

STACK:
#0: create(window, dialog D, true)
#1: rule D.Test ("c-thread-access-violation"), this=dialog D,
file=bad.dlg:78+40%
  valstack=["c-thread-access-violation", window, true]
#2: rule on extevent 1 , this=dialog D, file=bad.dlg:112+71%
#3: DM_EventLoop(null)
```

```
THISEVENTS:
#0: source=external, object=dialog D, data=1

USAGE:
              class  defaults    models instances     count     alloc
------------------ --------- --------- --------- --------- ---------
             record         1         0     24704     24705         0
             window         1         0         1         2         0
          clipboard         0         0         1         1         0
             dialog         0         1         1         2         0
             module         0         0         1         1         0
         setupclass         0         0         1         1         0
        thisevclass         1         0         1         2         0
        accelerator         1         0         3         4         0
           function         0         0         6         6         0
               rule         1         1         5         7         0
               text         1         6        16        23         0
            <total>         6         8     24740     24754         0

   module     count     slots     alloc    labels labelsize name
--------- --------- --------- --------- --------- --------- -----------------
--
   dialog         1     24745       99k        28        3k
   module         1         9        3k         1        3k
<maximum>               24745       99k        28        3k dialog D

   frames   f-alloc   scratch   s-alloc
--------- --------- --------- ---------
        2       13k         5        5k

VISIBLE OBJECTS:
#0: window D.WiMain

MEMORY:
        heap     alloc     other
- ---------- --------- ---------
*        crt       11m        3m
  0x030D0000        4k       61k
  0x030E0000        5k      167k
  0x00300000       11k       54k
  0x02EF0000       11m        4m
  0x027A0000        6k       58k
  0x00030000       13k       52k
P 0x00A50000       48m      791k
     <total>       59m        5m

*** DUMP STATE END ***
```

Aus der Dumpstate-Ausgabe kann man folgendes ablesen:

» Die Meldung *„ATTENTION: not in IDM main Thread!“* besagt, dass der Absturz (**ACCESS VIOLATION**) nicht im IDM passiert. Daher ist der Callstack in dieser Situation unverdächtig. Als es zu dem Absturz im anderen Thread kam, befand sich der IDM im Aufruf der Builtinfunktion **create**.

» Die Anzahl der ***record***-Instanzen im USAGE-Abschnitt ist verdächtig hoch. Dies ist in der **record**-Zeile der **class**-Tabelle sowie in der **<maximum>**-Zeile der **module**-Untertabelle erkennbar. Es spiegelt sich außerdem als hoher Speicherbedarf in der **crt**-Zeile der MEMORY-Tabelle wieder.

» Die MEMORY-Tabelle zeigt, dass sich im Default-Heap des Prozesses (durch „P“ gekennzeichnet) ein erheblicher Speicherverbrauch von über 50MB angesammelt hat.

8.10 Besonderheiten Windows/Threads

Der IDM verwaltet einen eigenen Callstack für Regelaufrufe, Methoden, eingebaute Funktionen, DM-Schnittstellenfunktionen und DM-Applikationsfunktion (siehe auch Kapitel „Callstack“). Für DM-Schnittstellenfunktionen (außer **DM_SendEvent**, **DM_QueueExtEvent**, die **nicht** auf dem Callstack erscheinen) findet eine Überprüfung statt, ob sie aus demselben Thread aufgerufen werden in dem der IDM initialisiert wurde. Ist dies nicht der Fall erfolgt die Ausgabe einer Fehlermeldung.

Bis auf wenige Ausnahmen sind der IDM und seine Schnittstellen nicht für die Nutzung in Multi-Threaded-Situationen ausgelegt. Auch der Callstack dient nur zur Verwaltung IDM-spezifischer Funktionalität und nicht für beliebige Applikationsfunktionen.

9 Tracing (Ablaufverfolgung)

Die Abarbeitung im Dialog Manager kann mit Hilfe verschiedener Kommandozeilen-Switches automatisch mitprotokolliert und Fehler erkannt werden.

9.1 Beschreibung des Tracing

Mit Hilfe der Tracing-Einrichtung können Dialog Manager-Anwendungen ausgetestet und Fehler beseitigt werden. Die Trace-Datei beinhaltet alle ausgeführten Regeln, alle aufgerufenen Anwendungsfunktionen und alle aufgerufenen Dialog Manager-Funktionen mit ihren gegenwärtigen Parametern (bitte vergleichen Sie hierzu auch das Handbuch „C-Schnittstelle - Grundlagen“). Des weiteren werden protokolliert:

- Eingebaute Funktionen mit Seiteneffekten (z.B. **create**)
- Setvalues (immer - nicht nur, wenn ein *changed*-Ereignis erzeugt worden ist)
- Aufrufe an Format- und Canvas-Funktionen
- DM-Aufrufe in Netzwerk-Anwendungen auf der Server-Seite

Die Datei sieht allgemein wie folgt aus:

```
[<Abkürzung>]Aktion
```

In <Aktion> steht die Erklärung dazu, was gerade gemacht wurde.

Die <Abkürzung> steht dafür, was der Dialog Manager gemacht hat. Folgende Codes können hier vorkommen:

[AC]	AppMain call. Aufruf an Main-Programm **AppMain** des Benutzers.
[AR]	AppMain return. Rückgabewert des **AppMain**-Programms an den Dialog Manager.
[BA]	Builder action: Start einer Aktion (z.B. **-writeexport**) im **Builder-Prozess-Modus**.
[BC]	Builtin function call. Aufruf einer eingebauten Funktion .
[BD]	Builder return: Ende einer Aktion (z.B. **-writeexport**) im **Builder-Prozess-Modus**.
[BM]	Builder message: Nachricht des IDM im **Builder-Prozess-Modus**.
[BR]	Builtin function return. Rückgabewert einer eingebauten Funktion.
[BX]	Builder command: Kommando für den IDM im **Builder-Prozess-Modus**, z.B. **-buildertimeout**.
[DC]	Dump Call. Quelltext-Ausgabe des aktiven Fensters nach Drücken der mit der Startoption **-IDMobjdump_fkey** definierten Funktionstaste.

[DE] Dialog event. Benutzerereignis, das vom Benutzer durch Interaktion mit dem System erzeugt wurde.

[DR] Dump Return. Ende der Quelltext-Ausgabe des aktiven Fensters nach Drücken der mit der Startoption **-IDMobjdump_fkey** definierten Funktionstaste.

[DS] Dialog start. Ausführung der Dialogstart-Regel.

[EC] ErrorHandler Call. Fehlerhandler wird aufgerufen

[EE] Exit. Die Anwendung wird nicht normal beendet. Der Status des Abbruchs wird in die Trace-Datei geschrieben.

[EF] Evaluation failure. Evaluationsfehler während des Interpretierens von Regelcode. Ausgabe erfolgt beispielsweise auch innerhalb einer fail()-Klammerung.

[ER] ErrorHandler Return. Rückkehr aus dem Fehlerhandler

[EQ] Return executing SQL statement. Rückgabewert nach der Ausführung eines SQL-Statements.

[EX] External event. Externes Ereignis.

[FA] Function assignment phase (Phase der Funktionszuweisung). Die übergebenen Parameter mit Schreibzugriff (als Ausgabeparameter deklariert) werden nun ihren Objektattributen zugewiesen.

[FC] Function call. Eine Funktion der Anwendung wurde vom Dialog Manager aufgerufen.

[FE] Finish dialog. Dialog beenden-Ereignis, das durch den Aufruf der Funktion **exit** in den Regeln aufgerufen wurde.

[FR] Function call return. Rückgabewert einer Anwendungsfunktion.

[IA] Interface argument. Argument, das von der Anwendung an eine Dialog Manager-Funktion übergeben wurde.

[IC] Interface call. Eine Dialog Manager-Funktion wurde von der Anwendung aufgerufen.

[IE] Interface error. Dialog Manager-Funktion gibt Fehler zurück.

[IR] Interface call return. Rückgabewert einer Dialog Manager-Funktion.

[IV] Interface value. Zusätzliche Werte, übergeben zu oder zurückgegeben von Schnittstellenfunktionen.

[MC] Method call. Aufruf einer Methode.

[ME] Module loading finished. Das Laden eines.Moduls ist beendet

[ML] Module loading. Ein Modul wird geladen

[MR] "Method return". Rückgabewert einer Methode.

[NI] Network information. Dies beschreibt, welches Netzwerkprotokoll ausprobiert wurde und welches gestartet wurde.

[PD] Finish of perform rule. Ausführung von Sub-Regel wurde beendet.

[PR] Perform rule. Aufruf einer Sub-Regel durch eine andere Regel.

[RC] Rule call. Aufruf einer benannten Regel.

[RR] Rule return. Rückgabewert einer benannten Regel.

[SC] Simulated function call. Simulierter Funktionsaufruf.

[SE] Setval event. Internes Ereignis, das durch das Setzen eines Objektattributes erzeugt wurde.

[SQ] Execute SQL statement. Ausführen eines SQL-Statements.

[SR] Simulated function return. Rückgabewert einer simulierten Funktion.

[SV] SetValue. SetValue zu Attribut von Objekt.

[TD] Tracing disabled. Abschalten des Tracing.

[TE] Tracing enabled. Aktivierung des Tracing.

[UA] Userclass argument. Informationen über ein Argument einer USW Callback-Funktion.

[UC] Userclass call. Eine USW Callback-Funktion wird aufgerufen.

[UM] User message. Diese Meldung wurde von der Anwendung mit Hilfe des Aufrufs von **DM_TraceMessage** geschrieben.

[UR] Userclass return. Rückkehr aus einer USW Callback-Funktion.

[UV] Userclass value information. Informationen über einen Rückgabewert einer USW Callback-Funktion.

[VS] Versionstring. Versionsstring der verwendeten Dialog Manager-Version (5 Zeilen).

[WE] Window system event. Fenstersystem-Ereignis.

[XD] Finish execution rule. Ausführung einer Regel wurde beendet.

[XR] Execution rule. Ausführung einer Regel wurde gestartet.

9.2 Konfigurierung des Tracing

Bei der Bearbeitung sehr großer Dialoge werden auch die Tracedateien entsprechend groß. Aus diesem Grund kann das Tracing konfiguriert werden. Die Möglichkeit das Tracing an- und auszuschalten

(*setup.tracing := true/false*) ist dabei zusätzlich verfügbar.

Die Konfigurierbarkeit des Tracing ermöglicht es, das Tracing für spezielle Trace-Ereignisse abzustellen. Dies kann z.B. nützlich sein für Schleifen oder andere trace-intensive Regeln bzw. Programmcodes. Sie können die Trace-Ereignisse am Setup-Objekt abstellen. Das Attribut .tracing wird mit einem String indiziert:

```
setup.tracing["<Abkürzung>"] := false;
```

Beispiel

```
setup.tracing["RR"] := false;
```

Diese Definition stellt alle Rückgabewerte von benannten Regeln ab.

Beim Abstellen bestimmter Trace-Ereignisse – z.B. ["RC"] (Aufruf einer benannten Regel) – können Sie auch einen Gruppierungseffekt erreichen: in dem eben genannten Beispiel werden neben den Aufrufen benannter Regeln auch die Rückgabewerte benannter Regeln unterdrückt. (s. unten Tabelle Spalte „Gruppierung“.)

Anmerkung

Bitte beachten Sie, dass folgende Trace-Ereignisse nicht abzustellen sind:

- Wichtige Trace-Ereignisse (siehe Tabelle unten, Spalte „Abstellbar durch Benutzer“.)
- Trace-Ereignisse via **DM_TraceMessage**

Warnung

Falls Sie die Unterstützung des IDM-Supports benötigen sollten, empfehlen wir Ihnen, nicht zu viele Trace-Messages abzustellen. Ansonsten kann Ihnen eventuell nicht effizient geholfen werden, da aufgrund des Fehlens entscheidender Trace-Messages keine ausreichende Analyse möglich ist.

Im Folgenden finden Sie entsprechende Programmiertipps.

Richtig

```
variable boolean I_NEED_HELP_FROM_ISA_SUPPORT := true;
...
rule MyCode()
{
  TraceEvent("RC", false);
  ...
}

rule TraceEvent(string TraceTag, boolean Value)
{
  !! Check the global variable for support.
  if ( not I_NEED_HELP_FROM_ISA_SUPPORT ) then
```

```
    setup.tracing[TraceTag] := Value;
  endif
}
```

Falsch

```
rule MyCode()
{
  !! Do not look for help in your rules without a trace.
  !! Nobody knows which rule was called!
  setup.tracing["RC"] := false;
  ...
}
```

In der folgenden Tabelle ist zusammengefasst, wie Trace-Ereignisse in der Tracedatei gruppiert und eingerückt werden. Außerdem ist angegeben, ob die Trace-Meldungen vom Benutzer abgestellt werden können. In der Tabelle bedeuten:

- „Abkürzung“ des jeweiligen Trace-Ereignisses – bitte vergleichen Sie dazu das Kapitel „Beschreibung des Tracing“.
- „Gruppierung“: Trace-Ereignis, welches das in der ersten Spalte stehende Trace-Ereignis mit an- bzw. ausstellt.
- „Einrückung“:

  = keine Einrückung

  > nach rechts

  < nach links

Abkürzung	Gruppierung	Einrückung	An-/Abstellbar durch Benutzer
IC	IC	>	Ja
IR	IC	<	Ja
IE	--	=	Nein
IA	IC	=	Ja
IV	IC	=	Ja
FC	FC	>	Ja
FR	FC	<	Ja
FA	FC	=	Ja

Abkürzung	Gruppierung	Einrückung	An-/Abstellbar durch Benutzer
EE	--	=	Nein
VS	--	=	Nein
UM	--	=	Nein
UC	--	=	Ja
UR	--	=	Ja
UA	--	=	Ja
UV	--	=	Ja
AC	AC	>	Ja
AR	AC	<	Ja
NI	--	=	Nein
TE	--	=	Nein
TD	--	=	Nein
SV	--	=	Ja
PR	PR	>	Ja
PD	PR	<	Ja
SQ	SQ	>	Ja
EQ	SQ	<	Ja
BC	BC	>	Ja
BR	BC	<	Ja
RC	RC	>	Ja
RR	RC	<	Ja
MC	MC	>	Ja
ME	ME	=	Nein
ML	ME	=	Nein
MR	MC	<	Ja

Abkürzung	Gruppierung	Einrückung	An-/Abstellbar durch Benutzer
DS	--	=	Ja
FE	--	=	Ja
SE	--	=	Ja
EX	--	=	Ja
DE	--	=	Ja
XR	XR	>	Ja
XD	XR	<	Ja
SC	SC	>	Ja
SR	SC	<	Ja
WE	--	=	Ja
EC	--	>	Ja
ER	EC	<	Ja

9.3 Zeitstempel bei der Ablaufverfolgung

Mit Hilfe der Option **-IDMtracetime <Nr>** kann während des Programmablaufes der absolute oder relative Zeitbedarf von Funktionen und Regeln mitprotokolliert werden. Dadurch ist es möglich, Stellen zu identifizieren, die sehr Zeit intensiv sind und daher sich für diese Funktionen oder Regeln entsprechende Tuningmaßnahmen lohnen können.

Es gibt drei Arten der Zeitmessung, die über den Parameter <*Nr*> eingestellt werden können:

0
: Im Tracefile werden keine Zeiten aufgeführt

1
: Dieser Wert kennzeichnet den Starttime-Modus. Bei diesem Modus werden alle Start- und Endezeiten mitprotokolliert, der Zeitverbrauch für einen einzelnen Aufbau kann dann aus der Differenz berechnet werden. Dabei wird in diesem Modus ausschließlich die System- und die Benutzerzeit betrachtet.
In diesem Modus erscheinen im Tracefile am Zeilenanfang die Zeiten im Format [hh:mm:ss:uuu]:

- hh = Stunden
- mm = Minuten

- ss = Sekunden
- uuu = Millisekunden

2

Dieser Wert kennzeichnet den Tracetime-Modus. In diesem Modus wird der Zeitunterschied zum letzten mitprotokollierten Aufruf gegeben. In diesem Modus kann also relativ einfach erkannt werden, wie viel Zeit für einzelne Aktionen verbraucht wird.
In diesem Modus erscheinen im Tracefile die Zeitdifferenz zur letzten Trace-Ausgabe im Format [ss:uuu] am Zeilenanfang:

- ss = Sekunden
- uuu = Millisekunden

3

Dieser Wert kennzeichnet den Realtime-Modus. In diesem Fall wird für jede zu protokollierende Aktion die reale Zeit (Uhrzeit) im Tracefile abgelegt.
In diesem Modus erscheinen im Tracefile die Realzeit im Format [hh:mm:ss] am Zeilenanfang:

- hh = Stunden
- mm = Minuten
- ss = Sekunden

9.4 Safety-Tracing

Verfügbarkeit

Der Safety-Modus des Tracing ist in den DM Versionen A.05.01.g3 und A.05.01.h sowie ab A.05.02.e verfügbar.

Das Safety-Tracing ist ein besonderer Modus des Tracefiles. Um das Mitlaufen des Tracefiles auch bei längeren Anwendungssitzungen mit möglichst geringen Performanzeinbußen und Ressourcenverbrauch zu ermöglichen, wird in diesem Modus ein Tracing in einen begrenzten Ringpuffer, der im Speicher gehalten wird, durchgeführt. Hierzu muss statt der Option **-IDMtracefile <filepath>** entweder **-IDMstracefile <filepath>** verwendet werden oder zusätzlich zu **-IDMtracefile <filepath>** der Safety-Modus über **-IDMstrace** eingeschaltet werden.

Bezogen auf den Ringpuffer heißt begrenzt, dass die Zeilenzahl und die Anzahl der Zeichen pro Zeile (ohne Einrückung) beschränkt sind. Außerdem wird eine Kürzung bei der Ausgabe von Stringwerten (typischerweise in "…" gesetzt) vorgenommen und die Stringlänge in [ ] angehängt. Die Begrenzungen können über die Option **-IDMstraceopts** beeinflusst werden. Für die hierarchische Einrückung im Tracefile werden zwei Leerzeichen verwendet. Dies kann **nicht** durch die Option **-IDMindent** beeinflusst werden. Sind die ausgegebenen Zeilen nicht zusammenhängend, so ist dies durch eine Zwischenzeile mit einem Doppelpunkt kenntlich gemacht. Ein Überschreiten der maximalen Zeilenlänge ist durch eine Tilde (~) am Zeilenende gekennzeichnet.

Der Inhalt des Ringpuffers wird erst beim Beenden der Anwendung in die Trace-Datei geschrieben. Um das Speichern auch bei Abstürzen zu gewährleisten, ist ein aktiver Exception-Catcher (siehe Kapitel „Exception-Catcher“) nötig.

Ein Abschalten des Tracing oder der Ausgabe bestimmter Tracecodes ist beim Safety-Tracing **nicht** möglich.

Beim Safety-Tracing gelten folgende Begrenzungen:

	Minimal	Maximal	Standard
Bytes pro Zeile	20	65.536	200
Zeilen	20	100.000	1.000
Stringlänge			40
Einrückungstiefe		50 (100 Leerzeichen)	

Index

F

G

H

I

K

L

M

N

O

P