# ISA Dialog Manager

## DISTRIBUTED DIALOG MANAGER (DDM)

A.06.03.c

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.

**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


Darstellungskonventionen ... 3

Inhalt ... 5

1 Verteilung der Anwendung auf verschiedene Rechner ... 7

2 Allgemeine Architektur ... 8

2.1 Schutz und Sicherheit für Daten und Systeme ... 9

2.1.1 Secure Sockets Layer (SSL) bzw. Transport Layer Security (TLS) ... 9

2.1.2 Secure Shell (SSH) ... 9

2.2 Schlüssel und Zertifikate ... 9

3 Umstellung einer nicht-verteilten auf eine verteilte Anwendung ... 10

3.1 Definition der Anwendungsteile - Aufteilung ... 10

3.2 Definition einer Fallback-Strategie (Ausweichstrategie für Fehlerfall) ... 10

3.3 Änderungen in den Sourcen ... 11

3.4 Änderung an Skripts zur Erzeugung der Anwendung (makefiles) ... 12

4 Funktionalität und Syntax im Dialogskript ... 13

4.1 Objekt Application ... 13

4.1.1 TCP/IP ... 14

4.1.1.1 Eine ungesicherte Verbindung einrichten ... 14

4.1.1.2 Eine mit SSL gesicherte Verbindung einrichten ... 15

4.1.2 IPv6-Unterstützung ... 16

4.2 Zuordnung der Funktionen ... 16

4.3 Beispiel ... 16

5 Applikationsschnittstelle ... 18

5.1 C-Schnittstelle ... 18

5.2 COBOL-Schnittstelle ... 18

5.3 Records und Funktionsdeklarationen ... 20

6 Prinzipielles Arbeiten in verteilter Umgebung ... 21

7 Startoptionen und -modi ... 22

7.1 Allgemeine Startoptionen ... 22

7.2 Protokoll TCP/IP ..... 22
7.2.1 Option -IDMlisten ..... 22
7.2.2 Option -IDMserve ..... 23
8 Voraussetzungen ..... 25
9 Übersetzen und Linken ..... 26
10 Änderung ab Version A.05.01.d ..... 27
10.1 Konsequenzen für bisherige Dialoge ..... 27
10.2 Anwendungsbeispiel ..... 28
10.2.1 Client ..... 28
10.2.2 Server ..... 32
10.3 Sonstige Hinweise ..... 33
10.4 Änderungen am Applikations-Objekt ..... 34
10.4.1 start/finish-Ereignis ..... 34
10.4.2 Fehlerverhalten bei Applikationsfunktionen ..... 34
10.5 Netzwerk-Applikationsseite ..... 34
Index ..... 37

# 1 Verteilung der Anwendung auf verschiedene Rechner

Mit Hilfe des verteilten Dialog Managers ("DDM") können Sie Ihre Anwendung in unterschiedliche Prozesse aufteilen. Diese Aufteilung sollte in sinnvollen Einheiten erfolgen, um den Kommunikationsaufwand zwischen den Prozessen möglichst gering zu halten. Ansonsten kommt es zu einer zu hohen Netz- bzw. Rechnerbelastung, was nicht ohne Auswirkung auf die Performanz der Anwendung bleibt.

Einer dieser Prozesse verarbeitet die Benutzerschnittstelle und die Interaktionen des Benutzers, die anderen Prozesse verarbeiten den algorithmischen Teil der Anwendung. Diese Prozesse können auf verschiedenen Maschinen mit unterschiedlichen Betriebssystemen laufen, z.B. die Benutzerschnittstelle läuft unter Microsoft Windows und der Anwendungsteil läuft auf einer Unix-Maschine. Die Kommunikation zwischen den Prozessen wird vom DDM unter Verwendung grundlegender Kommunikationsdienste und vom Netzwerk zwischen den Maschinen erledigt.

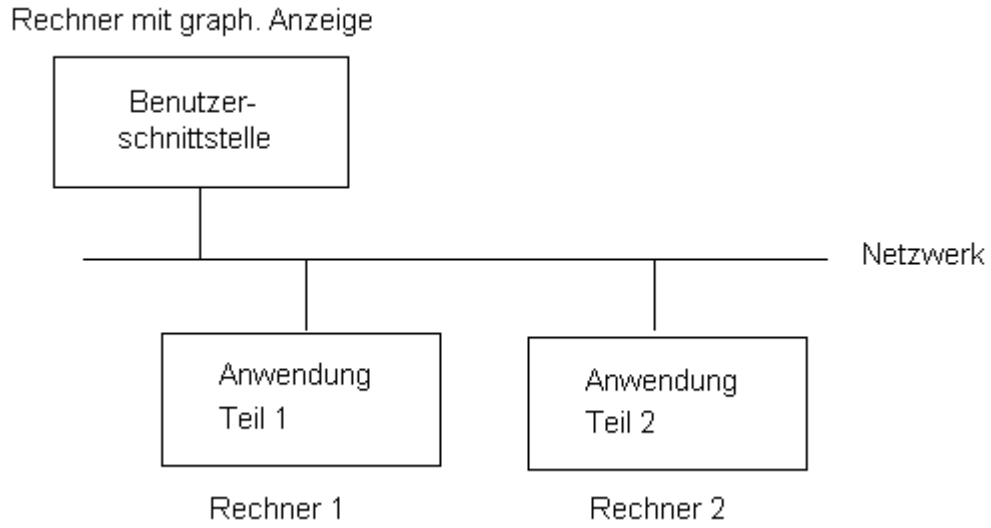


***Abbildung 1:*** *Verteilter Dialog Manager „DDM“*

Bei der Aufteilung der Anwendung kann natürlich auch ein Teil auf dem Display-Rechner verbleiben.

Dadurch ist es möglich, diesen Rechner besser auszulasten, den Server zu entlasten sowie die Ausgabe beliebiger Grafiken oder das Abfragen externer Schnittstellen zu realisieren.

# 2 Allgemeine Architektur

Der DDM baut auf die grundlegenden Kommunikationsdienste auf, die auf dem Rechner verfügbar sein müssen. Mit Bezug auf das ISO OSI-Referenzmodell; der Kommunikation hat der Dialog Manager die Schicht 7 ("application layer"), die Schicht 6 ("presentation layer") und - je nach Netzwerkkomponente - die Schicht 5 ("session layer") implementiert.

Die anderen Schichten 1 - 4 ("physical", "datalink", "network" und "transport layer") müssen verfügbar sein, ansonsten läuft der DDM nicht.

Wenn der Benutzerschnittstellenteil den Anwendungsteil aufrufen möchte, schickt der Dialog Manager eine Anfrage an den Anwendungsteil. Der Dialog Manager verpackt zwischen den Prozessen die Information und packt sie wieder aus. Der Kommunikationsmechanismus zwischen diesen Prozessen gleicht "rpc" ("remote procedure calls"), d.h. die einzelnen Prozesse enthalten Stubs, die für die Kommunikation gebraucht werden. Diese Stubs sind Teil vom Dialog Manager.

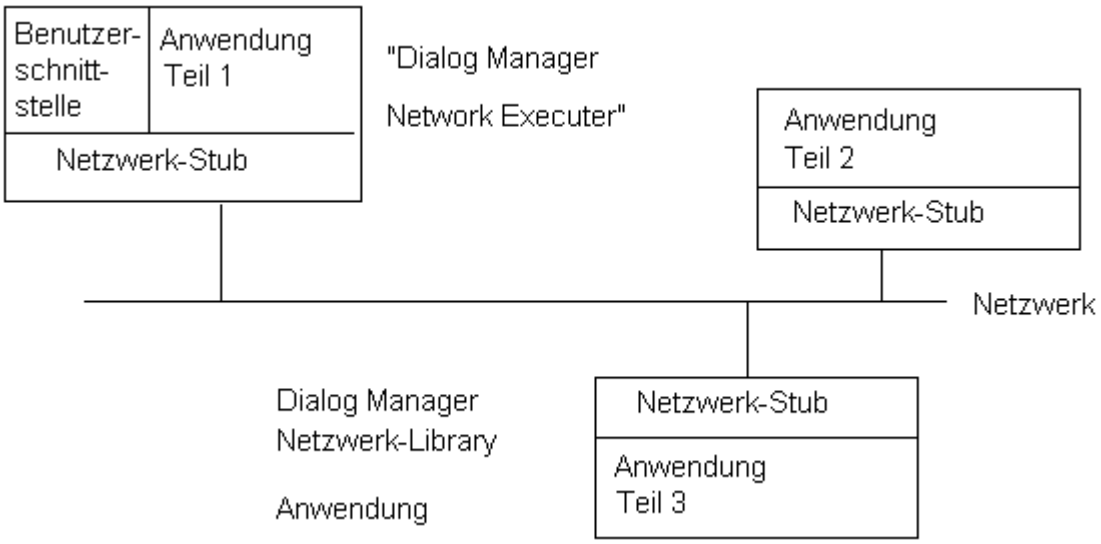


***Abbildung 2:*** *Allgemeine Architektur*

Um den Benutzerschnittstellenteil ablaufen lassen zu können, ist ein "DM Network Executer" verlangt. Dieses Programm übernimmt die Kommunikation zwischen den verschiedenen Anwendungsteilen. Dieser "DM Network Executer" ist entweder das zur Verfügung gestellte Simulationsprogramm **idmndx** oder kann durch Linken mit speziellen DM-Libraries selbst erzeugt werden.

Auf der anderen Seite werden netzwerkfähige Anwendungen gebraucht. Diese können durch das Linken der Anwendung mit der Netzwerk-Stub-Library des Dialog Managers (**libIDMnet.a**) und durch das Hinzufügen spezieller Funktionen zum Source-Code (**AppInit**, **AppFinish**) erzeugt werden.

Um eine zusätzliche Kommunikation zwischen den verschiedenen Anwendungsteilen realisieren zu können, stellt der Dialog Manager Mechanismen zur Verfügung, um Funktionen netzwerktransparent aufrufen zu können.

## 2.1 Schutz und Sicherheit für Daten und Systeme

Zum Schutz von Systemen und Daten können kryptografische Protokolle integriert werden.

### 2.1.1 Secure Sockets Layer (SSL) bzw. Transport Layer Security (TLS)

Um eine vertrauliche End-zu-End-Datenübertragung zwischen Darstellungsseite (Client) und Anwendungsseite (Server) zu gewährleisten, können die transportierten Daten mit Hilfe von **SSL** bzw. **TLS** verschlüsselt werden. Bei **TLS** handelt es sich um eine Weiterentwicklung von **SSL**. Da der Begriff **SSL** geläufiger ist, wird er in der Folge verwendet.

Voraussetzung ist, dass auf dem System **OpenSSL** installiert ist. Wenn die dynamische Bibliothek **libssl** (Version 1) im Suchpfad liegt, lädt der IDM diese zur Laufzeit und bietet die SSL-Unterstützung an.

### 2.1.2 Secure Shell (SSH)

Das SSH-Protokoll ist eine Methode, um Befehle sicher über ein unsicheres Netzwerk an einen Computer zu senden. SSH verwendet Kryptographie zur Authentifizierung und Verschlüsselung von Verbindungen zwischen Geräten. Für einen sicheren Start der Anwendungsseite unterstützt die Netzwerkstelle SSH.

Voraussetzung ist, dass auf dem System **libssh** installiert ist (WINDOWS: Version 1, UNIX: ab Version 0.9). Wenn die dynamische Bibliothek **libssh** im Suchpfad liegt, lädt der IDM diese zur Laufzeit und bietet die **SSH**-Unterstützung an.

## 2.2 Schlüssel und Zertifikate

Um **SSH** bzw. **SSL** zu verwenden, werden Schlüssel und Zertifikate benötigt. Eine Anleitung zum richtigen Aufsetzen der Umgebung würde weit über den Rahmen dieser Dokumentation hinausgehen. Unter **https://docs.openssl.org/master** wird auf entsprechende Literatur verwiesen.

Für einfache Tests kann folgendes Kommando verwendet werden:

```
openssl req -x509 -newkey rsa:4096 -keyout private.pem
  -out cert.pem -days 365 -nodes
```

Hierdurch werden die Zertifikatsdateien **cert.pem** und **private.pem** ohne „Pass Phrase“ erzeugt. Wenn beide Dateien im Verzeichnis der IDM-Anwendungsseite liegen, werden keine weiteren Startoptionen benötigt.

# 3 Umstellung einer nicht-verteilten auf eine verteilte Anwendung

Die Umstellung einer herkömmlichen, d.h. nicht-verteilten Anwendung auf eine verteilte Anwendung erfordert Änderungen im Dialogskript und im Source-Code.

Verfolgen Sie dies bitte anhand des mitgelieferten exemplarischen Beispiels **list**.

## 3.1 Definition der Anwendungsteile - Aufteilung

Um Anwendungen verteilen zu können, muss die bestehende Dialogdatei in sinnvolle Untereinheiten sog. Anwendungen (***application***) aufgeteilt werden. Diese Änderungen sind dabei ausschließlich auf die Funktionsdefinition beschränkt und sehen wie folgt aus:

- Funktionen, die auf dem Display-Rechner (Benutzerschnittstelle), d.h. lokal, liegen sollen, müssen (wie bisher) global zum Dialog gehörend definiert werden.
- Funktionen, die auf anderen Rechnern bzw. Anwendungsteilen liegen sollen, müssen innerhalb einer Anwendung definiert werden.

Dabei gilt, dass Funktionen nur einmal definiert werden dürfen, d.h. ein Funktionsname darf nur einmal verwendet werden.

Die Definition der Anwendung umfasst neben der Deklaration der in ihr enthaltenen Funktionen auch noch Angaben über den Programmnamen, den Rechnernamen, das zu verwendende Protokoll und den Verbindungsaufbau zu dieser Anwendung. Diese können anhand von Attributen zur Laufzeit geändert werden (Einschränkungen vgl. "Objekt Application").

## 3.2 Definition einer Fallback-Strategie (Ausweichstrategie für Fehlerfall)

Im Gegensatz zu lokalen Anwendungen können durch den Einsatz eines Netzwerkes systembedingte Fehler zur Programmlaufzeit auftreten. Dadurch kann es z.B. nicht möglich sein, alle eigentlich für die Anwendung benötigten Funktionen zu benutzen, da andere Rechner oder das Netzwerk ausgefallen sind. Aus diesem Grund können in verteilten Anwendungen solche Fehler erkannt und teilweise umgangen bzw. behoben werden. Um nun auf solche Fehler zu reagieren, stehen verschiedene Möglichkeiten zur Verfügung:

- Dynamisches Ändern des Namens eines Rechners oder Programms in den Namen eines anderen Rechners/anderen Programms, auf den/das der Zugriff während des Programmlaufs erlaubt ist.
  Danach erneutes Starten der Anwendungsteile.

» Anwendungsteile lokal starten bzw. anbinden. Dies erfolgt durch das Setzen des Attributs .local auf *true*. Diese Methode ist nur dann sinnvoll, wenn die lokale Anwendung die benötigten Funktionsaufrufe beinhaltet, im Normalfall aber aus Performanz-Gründen nicht nutzt. Wird eine Anwendung lokal gestartet, wird die zugehörige Regel *on <application> start* ausgeführt. In dieser Start-Regel kann dann die Funktion aufgerufen werden, die die lokal zur Verfügung gestellten Funktionen an den DM übergibt.
Danach kann wie bisher weitergearbeitet werden.

### Beispiel

Im Fall des Beispiels **list** soll die Variante b, d.h. die lokale Anbindung verwendet werden. Dazu wird die Start-Regel so geändert, dass - falls der Start der remote-Anwendung nicht gelingt - die lokale Anwendung auf lokal umgesetzt wird.

Durch die zusätzliche Funktion `InitTestAppl` werden die Namen der nun lokal verfügbaren Funktionen an den DM übergeben. Dies erfolgt durch den Funktionsaufruf in der Start-Regel der Anwendung:

```
on TestAppl start
```

Mit diesen Änderungen sind alle notwendigen Änderungen im Dialogskript abgeschlossen.

Zusätzliche Änderungen sind noch in den Anwendungssourcen notwendig.

## 3.3 Änderungen in den Sourcen

Für eine komplett verteilte Anwendung müssen mindestens zwei Anwendungen zusammengebunden werden:

» der lokale Teil, der zum Display (Benutzerschnittstelle) gebunden wird,
» der Teil, der als "remote"-Anwendung dienen soll.

Im lokalen Teil sind prinzipiell keine Änderungen durchzuführen. Der Einstieg erfolgt wie gewohnt über das Programm **AppMain**. Hier müssen die üblichen Initialisierungsschritte wie beim nicht-verteilten DM durchgeführt werden.

Im "remote"-Teil hingegen darf kein **AppMain** enthalten sein. Dafür müssen die Funktionen

» **AppInit**
» **AppFinish**

zur Initialisierung und Beendigung der Anwendung enthalten sein.

Die Aufgabe der **AppInit**-Funktion ist es, die eigentliche Anwendung zu initialisieren und die darin enthaltenen Funktionen an den DM zu übergeben.

Die Aufgabe der **AppFinish**-Funktion ist es, den entsprechenden Anwendungsteil korrekt zu beenden.

**Beispiel**

Um einen gemeinsamen Source-Code für das Beispiel **list** zu haben ("remote"- und lokaler Teil), werden Bedingungen ("ifdef") im Source-Code eingefügt.

Für den Fall, dass XXX_NETWORK definiert ist, enthält die Source die vier Funktionen sowie ein **AppInit** und ein **AppFinish**.

Für den Fall, dass XXX_NETWORK nicht definiert ist, wird wie bisher ein **AppMain** und die Funktion `InitTestAppl` definiert.

Für dieses Beispiel wurde die lokale Anbindung der Funktionen als Fallback gewählt, d.h. die lokale Source enthält zusätzlich die vier vom Beispiel verlangten Funktionen. Diese werden aber nur dann verwendet, wenn die "remote"-Anwendung nicht gestartet werden kann.

## 3.4 Änderung an Skripts zur Erzeugung der Anwendung (makefiles)

Statt ein Programm zu erzeugen, sollen zwei Programme erzeugt werden.

Dazu muss die "remote"-Anwendung nur mit der Netzwerk-Library des DM gebunden werden (**libIDM-net.a**).

Die lokale Anwendung muss zusätzlich neben der bisher verwendeten

- **libIDM.a** (Motif)
- **dm.lib** (MS Windows)

mit der Library für den DM Network Executer

- **libIDMndx** (Motif)
- **dmndx.lib** (MS Windows)

gebunden werden, wobei zuerst die Library für den DM Network Executer angegeben werden muss.

Danach erhält man zwei Anwendungsteile, die miteinander kommunizieren können.

Zur Anpassung an Ihre Rechnerumgebung muss noch der Rechnername im Dialogskript durch einen bei Ihnen verfügbaren Namen ausgetauscht werden.

# 4 Funktionalität und Syntax im Dialogskript

Um verteilte Anwendungen verarbeiten zu können, müssen neue Konstrukte in der Dialog-Beschreibungssprache eingesetzt werden. Dazu steht das Objekt "Application" zur Verfügung, das Informationen über die Anwendung, deren Funktionen und die Art der Kommunikation enthält. Die so erhaltenen Anwendungen können dann einzeln gestartet und wieder beendet werden.

Durch die Untergliederung in "Applikationen" wird dem Dialog Manager mitgeteilt, welche Funktionen zusammengehören und zusammen eine Anwendung aus Sicht des Benutzers darstellen.

## 4.1 Objekt Application

Das Objekt Application ist im Prinzip wie jedes andere Objekt zu behandeln, außer dass es keinerlei Display-Information enthält. Es besitzt folgende Attribute:

| Attribut | Zugriff | Bedeutung |
|---|---|---|
| .active | set, get | Mit Hilfe dieses Attributs kann definiert und abgefragt werden, ob die Anwendung gerade läuft. Eine Änderung dieses Attributes von *false* auf *true* bewirkt, dass die Anwendung gestartet wird. Wird dieses Attribut von *true* auf *false* geändert, so wird die Anwendung wieder beendet. |
| .connect | set, get | Dieses Attribut definiert, dass sich die Anwendung mit einem laufenden Server Prozess in Verbindung setzen soll, der auf dem Host gestartet worden ist (definiert durch Name oder IP Adresse und der Portnummer). |
| .exec | set, get | Dieses Attribut definiert, dass die Anwendung einen Prozess auf dem Host starten soll, der durch den Pfad gegeben ist (definiert durch Name oder IP Adresse). Der Wert des Attributs beinhaltet den Programmnamen, Pfad, Rechnernamen und sonstige Informationen. |
| .label | set, get | Interner Name der Anwendung, Identifikator. |
| .local | set, get | Mit Hilfe dieses Attributs kann definiert werden, ob die Anwendung lokal oder über ein Netzwerk laufen soll. |
| .transport | set, get | Mit Hilfe dieses Attributes wird definiert, welchen intern vorhandenen Transportmechanismus die Kommunikationsschicht verwenden soll. Dabei gibt es folgende Möglichkeit: *tcpip*. |

Die Attribute *.transport*, *.connect*, *.local* und *.exec* können nur geändert werden, wenn *.active = false* ist.

Die Attribute *.connect* und *.exec* sind vom verwendeten Transport abhängig, d.h., zukünftige neue Arten einer Transportschicht können andere Arten des Verbindungsaufbaus enthalten.

#### Hinweis bei Verwendung von .exec zum Starten des Servers

Bei Verwendung von ".exec" wird auf dem Anwendungsserver mittels "rexec" Protokoll eine IDM Anwendung gestartet, und der Darstellungsrechner wartet darauf (entsprechend -IDMlisten), dass sich diese Anwendung verbindet. Die Anwendung auf dem Anwendungsserver öffnet dann eine neue Verbindung (entsprechend .connect) um sich mit dem Darstellungsrechner zu verbinden. Dies wird im Normalfall eine "normale" Verbindug sein, unabhängig davon, wie die Verbindung zum "rexecd" Service aufgebaut wurde.

## 4.1.1 TCP/IP

### 4.1.1.1 Eine ungesicherte Verbindung einrichten

**Attribut *.connect***

Im Falle des Protokolls "TCP/IP" sieht die Syntax von *.connect*; wie folgt aus:

```
"<host>:<portnumber>"
```

Im Parameter "host" wird der Rechnername, im Parameter "portnumber" die zu verwendende Portnummer angegeben.

**Attribut *.exec***

Im Falle des Protokolls "TCP/IP" sieht die Syntax von *.exec* wie folgt aus:

```
"<host>[%<username>[%<password>]]:<path>"
```

Im Parameter "host" wird der Rechnername angegeben, auf dem das Programm gestartet werden soll.

Im Parameter "username" kann der Name des Users angegeben werden, unter dem sich der Benutzer auf dem Host-Rechner einloggen soll.

Im Parameter "password" kann das dort gültige Password des Benutzers angegeben werden. Diese beiden Parameter sind optional.

Optional bedeutet, dass die Angaben abhängig von der Art der Installation des Netzwerks zwingend sind oder nicht.

Im Parameter "path" wird der Pfad des zu startenden Programms angegeben.

### Hinweis

Um einen Doppelpunkt vor dem syntaktisch vorgeschriebenen Doppelpunkt zu verwenden (im Name oder Passwort) muss dieser verdoppelt werden. Im Kommandoteil des .exec Attributes (nach syntaktisch vorgeschriebenen Doppelpunkt) werden Doppelpunkte direkt übernommen.

## 4.1.1.2 Eine mit SSL gesicherte Verbindung einrichten

#### Attribut *.connect*

Um die Verbindung über **SSL** durchzuführen kann bei der Spezifikation der Verbindung über *.connect* das Schema *"ssl://"* vorangestellt werden (Beispiel `.connect "ssl://myserver:0815";`).

### Anmerkung

Auch beim Attribut *.transport* kann das Schema *"ssl://"* angegeben werden. Wenn bei beiden Attributen ein Schema angegeben ist, muss es identisch sein. Ein einmal angegebenes Schema *"ssl://"* lässt sich nicht mehr abstellen.
Beispiel

| Ohne SSL | Mit SSL |
|---|---|
| `application Appl {`<br>`  .connect "localhost:4711";`<br>`}` | `application Appl {`<br>`  .connect "ssl://localhost:4711";`<br>`}` |

#### Attribut *.exec*

Um die Anwendungsseite über **SSH** an Stelle von RSH zu starten, kann bei der Spezifikation des Kommandos das Schema *"ssh://"* vorangestellt werden (Beispiel `.exec "ssh://myserver:list";`).

Standardmäßig wird das RSH-Protokoll verwendet, wenn kein Sicherheitsschema beim *.transport*-Attribut angegeben wurde. Andernfalls wird das SSH-Protokoll verwendet. Soll von diesem Standard abgewichen werden, muss das gewünschte Schema beim *.exec*-Attribut angegeben werden. Hierbei wird mit *"ssh://"* das SSH-Protokoll und mit *"rsh://"* das RSH-Protokoll ausgewählt.

### Anmerkungen

- Die für SSL notwendigen zusätzlichen Startoptionen müssen beim Kommando angegeben werden. Nur das Sicherheitsschema „ssl“ wird – falls verwendet – dem Kommando als zusätzliche Startoption `-IDMtransport ssl` hinzugefügt.
- Die Kommandozeile erhält automatisch die Startoption `-IDMtellport`. Wird als Kommando ein Skript oder ähnliches an Stelle einer IDM-Serveranwendung angegeben, dann muss von der Ausgabe der IDM-Serveranwendung zumindest die Zeile, die mit `-IDMport` beginnt, komplett (inklusive führender und folgender Zeilenumbrüche „\n“) weitergeleitet werden.

**Beispiel**

| Ohne SSH | Mit SSH |
|---|---|
| `application Appl1 {`<br>`  .exec"host%account%passwd:list";`<br>`}` | `application Appl1 {`<br>`  .exec "ssh://host%account%passwd:list";`<br>`}` |

| Ohne SSH | Mit SSH und SSL |
|---|---|
| `application Appl2 {`<br><br>`  .exec "host%account%passwd:list";`<br>`}` | `application Appl2 {`<br>`  .transport "ssl";`<br>`  .exec "host%account%passwd:list";`<br>`}` |

### 4.1.2 IPv6-Unterstützung

Der DISTRIBUTED DIALOG MANAGER (DDM) unterstützt das IPv6-Protokoll auf allen Architekturen, die IPv6 **nativ** unterstützen.

Wenn im Dialogskript eine IPv6-Adresse angegeben werden soll, muss sie – wie bei URLs üblich – in eckige Klammern ('[' und ']') eingeschlossen werden (z. B. "[::1]").

## 4.2 Zuordnung der Funktionen

Bei einer Verteilung der Anwendung müssen die in dem Dialog enthaltenen Funktionen verschiedenen Anwendungen zugewiesen werden. Dabei ist es nicht möglich, dass mehrere Anwendungen dieselbe Funktion beinhalten.

Diese Zuordnung erfolgt, indem die Funktionen bei ihrer Deklaration den Anwendungen zugeordnet werden. Diese Zuordnung erfolgt dabei so, dass die zu einer Anwendung gehörenden Funktionen innerhalb der Deklaration dieser Anwendung deklariert werden, also genauso wie die Kinder eines Fensters definiert werden. Werden Funktionen außerhalb einer Anwendung im Dialog definiert, so gehören sie zum Dialog und müssen zur Dialogseite dazugebunden werden.

## 4.3 Beispiel

Das folgende Beispiel zeigt, wie die Anwendung in zwei Teile, einem Teil direkt beim Dialog und einem Remote-Teil, aufgespalten wird:

```
dialog Example

application Remote
{
  .exec   "boole:/usr/bin/example1";
  .active false;
```

```
  function c integer LoadDB;   /* remote function */
}

function callback CheckValue;  /* local function */

on dialog start
{
  Remote.active := true;
  if (not Remote.active) then
    exit();
  endif
}
```

# 5 Applikationsschnittstelle

Um eine netzwerkfähige Anwendung realisieren zu können, müssen spezielle Funktionen in der Anwendung zum Starten und Beenden der Anwendung bereitgestellt werden. Zusätzlich kann zur Kommunikation zwischen den Anwendungsteilen eine Dialog Manager-Utility-Funktion herangezogen werden. Diese Funktion erlaubt es, Funktionen aufzurufen, unabhängig davon, ob die Funktion lokal oder "remote" realisiert ist.

## 5.1 C-Schnittstelle

Die Main-Funktion **AppMain** muss durch die Funktionen

- **AppInit**
- **AppFinish**

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

Mit Hilfe der Funktion DM_CallFunction können Funktionen in beliebigen Anwendungsteilen aufgerufen werden.

### Beispiel

```
int DML_c AppInit __4(
(DM_ID, appl),
(DM_ID, dialog),
(int, argc),
(char **, argv))
{
    if (!DM_BindCallBacks(ApplFuncMap, ApplFuncCount, appl,
        DMF_Silent))
            DM_TraceMessage("There are functions missing", 0);
    return 0;
}

int DML_c AppFinish __2(
(DM_ID, appl),
(DM_ID, dialog))
{
    return 0;
}
```

## 5.2 COBOL-Schnittstelle

Das COBOL-Hauptprogramm **COBOLMAIN** muss durch die Funktionen

» **COBOLAPPINIT**

» **COBOLAPPFINISH**

ersetzt werden.

### Beispiel

```
* SET OSVS

IDENTIFICATION DIVISION.
PROGRAM-ID. COBOLMAIN.

DATA DIVISION.
WORKING STORAGE SECTION.
COPY "IDMcobws.cob".

77 NULL-OBJECT   PIC 9(4) BINARY VALUE 0.
77 FUNC_NAME   PIC X(32) VALUE SPACES.
77 BUFFER   PIC X(80) VALUE SPACES.

LINKAGE SECTION.
01 EXIT-STATUS PIC 9(4) BINARY.
01 APPL-ID PIC 9(4) BINARY.
01 DIALOG-ID PIC 9(4) BINARY.

PROCEDURE DIVISION USING EXIT-STATUS.
MAIN SECTION.

WAKING-UP.
    DISPLAY "COBOLMAIN CALLED".
    GOBACK.

    ENTRY "COBOLAPPINIT" USING EXIT-STATUS APPL-ID DIALOG-ID.
INIT SECTION.

WELCOME.
    DISPLAY "COBOLAPPINIT CALLED".

INITIALIZE -IDM.
    MOVE "@" TO DM-SETSEP.
    MOVE LOW-VALUE TO DM-GETSEP.
    MOVE "DMcob_Initialize" TO FUNC-NAME.
    CALL "DMcob_Initialize" USING DM-STDARGS DM-COMMON-DATA.

BIND-FUNCTIONS.
    CALL "BindFuncs" USING DM-STDARGS APPLIC-ID.
    PERFORM ERROR CHECK.
```

```
    GOBACK.

    ENTRY "COBOLAPPFINISH" USING EXIT-STATUS APPLIC-ID
        DIALOG-ID.
FINISH SECTION.

GOOD-BYE.
    DISPLAY "COBOLAPPFINISH CALLED".
GOBACK.
```

## 5.3 Records und Funktionsdeklarationen

Wenn mit Records in einer Applikation gearbeitet werden soll, muss die dazu notwendige Datei über den Simulator mit der Option

**-application**

und

**+writetrampolin**

erzeugt werden.

### Aufruf

```
idm -application <applicationname>
        + writetrampolin <filename> <dialogskript>
```

Diese so erzeugte C-Datei muss dann wie üblich übersetzt werden.

Sollen für eine Applikation C-Funktionsprototypen erstellt werden, muss der Simulator ebenfalls mit der Option **-application** und **+writeproto** gestartet werden.

### Aufruf

```
idm -application <applicationname>
        +writeproto  <filename>  <dialogskript>
```

# 6 Prinzipielles Arbeiten in verteilter Umgebung

Bei der Programmierung von verteilten Anwendungen sollten folgende Gegebenheiten beachtet werden.

- In der Regel ist der Funktionsaufruf mit einem sehr hohen Overhead versehen, d.h. Funktionsaufrufe über das Netzwerk sind relativ teuer.
- Die bei einem Funktionsaufruf angegebene Datenmenge hat für die Gesamtbelastung der Rechner eine untergeordnete Bedeutung.

Aus diesem Sachverhalt lassen sich folgende grundlegende Regeln für die Dialog Manager-Programmierung ableiten:

- Die Verteilung der Funktionen sollte so erfolgen, dass sehr häufig benötigte Funktionen möglichst auf dem Anzeigerechner realisiert sind.
- Beim Aufruf der Anwendungsfunktionen sollte diesen möglichst alle Information mitgegeben werden, die sie benötigen. Diese Informationen können in beliebig strukturierte Records verpackt und an die Anwendungsfunktion übergeben werden. Dieses ist wesentlich effizienter als das nachträgliche Abfragen von Attributen über die Schnittstellenfunktion **DM_GetValue**.
- Objekte sollten prinzipiell mit der mächtigsten, für sie zur Verfügung stehenden Funktion bearbeitet werden. Daher sollten für die Bearbeitung von Listboxes und Tablefields wenn möglich die Funktionen **DM_GetContent** bzw. **DM_SetContent** benutzt werden.

# 7 Startoptionen und -modi

Der Dialog Manager unterstützt verschiedene Protokollarten, welche für die grundlegenden Kommunikationsdienste verwendet werden können. Abhängig von dem verwendeten Protokoll gibt es unterschiedliche Kommandozeilenoptionen.

## 7.1 Allgemeine Startoptionen

Der Anwendungsteil, der den Displayteil darstellt, versteht alle Optionen, die auch das nicht verteilte Programm verstehen würde (siehe Kapitel „Startoptionen“ im Handbuch „Entwicklungsumgebung“).

Im Serverteil werden nur folgende Optionen verstanden:

- **-IDMversion**
- **-IDMtracefile**
- **-IDMerrfile**

D.h. alle Optionen, die eigentlich irgendwelche Anzeigeinformationen darstellen, werden ignoriert.

## 7.2 Protokoll TCP/IP

Auf den meisten Systemen ist der Kommunikationsdienst TCP/IP verfügbar. Um diesen Protokolltyp zu verwenden, müssen Sie den Dialog Manager mit der Option **-IDMtransport tcpip** starten.

Die gesamte Anwendung besteht aus verschiedenen ausführbaren Programmen. Eine Anwendung kann der „DM Network Executer“ (**idmndx**) sein, wenn kein Anwendungsteil lokal ist. Ist irgendein Anwendungsteil lokal, ist es ein mit dem Dialog Manager zusammengebundenes Programm. Die anderen Anwendungen sind Ihre Anwendungsteile, die mit der Netzwerk-Library des Dialog Managers (**libIDMnet.a**) und den verlangten System-Libraries für die Netzwerk-Funktionalität gebunden sind.

Im folgenden finden Sie die Startoptionen, die beim Start von Anwendungen im verteilten Dialog Manager benötigt werden. Diese sind in der Regel nur in folgenden Fällen notwendig:

- Wenn ein oder mehrere Anwendungsteile als Server gestartet werden sollen.
- Wenn die Angabe bei der Definition der jeweiligen Anwendung im Dialogskript nicht vollständig erfolgt ist oder überschrieben werden soll.

Eigentlich stehen für den Verbindungsaufbau Attribute im Dialogskript zur Verfügung, z.B. *.transport*, *.connect*, *.exec* (siehe Objekt Application).

### 7.2.1 Option -IDMlisten

Mit dieser Option wird die Anwendung im **Listen-Modus** gestartet, d.h. die Anwendung wartet darauf, eine Verbindung zu dem anderen Teil der Anwendung auf einer anderen Maschine zu akzeptieren.

Wenn der andere Teil Verbindung verlangt, wird die Verbindung akzeptiert und die Initialisierung ist erfolgt.

### Syntax

**-IDMlisten <portnumber>**

Die *<portnumber>* muss auf Ihrer Maschine frei und verfügbar sein. Normalerweise muss die Portnummer größer als 1024 sein, da die meisten Portnummern kleiner als 1024 von den Systemservices verwendet werden. Die Portnummern zwischen 5800 und 7000 werden vom Fenstersystem X Windows verwendet. Um daher sicher zu gehen, dass die Portnummer, die Sie verwenden wollen, verfügbar ist, fragen Sie bitte Ihren Systemadministrator oder nehmen Sie eine Portnummer über 10000.

Nachdem die Ausführung der Anwendung beendet worden ist, wird auch die im **Listen-Modus** gestartete Anwendung beendet. Wenn der Benutzer seine Anwendung wieder starten möchte, muss er die Anwendung im **Listen-Modus** erneut starten.

Daher wird diese Option normalerweise während der Entwicklung einer verteilten Anwendung verwendet oder wenn die Anwendung nur gelegentlich verwendet wird.

Diese Option ist unter UNIX verfügbar.

## 7.2.2 Option -IDMserve

Mit dieser Option wird die Anwendung im **Server-Modus** gestartet, d.h. die Anwendung erwartet, eine Verbindung zu einem oder mehreren Clients zu akzeptieren. Wenn ein Client eine Verbindung verlangt, akzeptiert die Anwendung die Verbindung, sucht nach einer neuen freien Portnummer, schaltet den Verbindungsteil zu dem gefundenen freien Port und dupliziert sich selbst. Daher läuft die Anwendung zweimal, nachdem die Anwendung eine Verbindung akzeptiert hat. Eine Anwendung wartet immer noch auf eine Verbindung und die andere Anwendung läuft normal.

### Syntax

**-IDMserve <portnumber>**

Die *<portnumber>* muss auf Ihrer Maschine frei und verfügbar sein (vgl. oben -IDMlisten).

Nach oder während der Ausführung einer Anwendung kann der Server neue Verbindungen akzeptieren. Wenn der Benutzer seine Anwendung wieder starten möchte, muss er nur den anderen Teil der Anwendung erneut starten. Der Server selbst kann nur durch Unterbrechen oder Abbrechen des Prozesses gestoppt werden.

Daher wird diese Option normalerweise verwendet, wenn die Anwendung oft oder von mehr als einem Benutzer gleichzeitig verwendet wird.

Diese Option ist auf Systemen verfügbar, die einen Fork-Mechanismus implementiert haben.

### Tracing

Die Optionen **-IDMtracefile**, **-IDMtracetime**, **-IDMerrfile**, **-IDMerrwinfile** werden an die Kind-Prozesse weitergeben. Sinnvoll ist hier die Benutzung des %P-Platzhalters, da ansonsten Tracefile bzw.

Logfile überschrieben werden.

# 8 Voraussetzungen

Für den Einsatz des verteilten Dialog Managers auf unterschiedlichen Hardware-Plattformen müssen verschiedene Anforderungen erfüllt sein.

Generell gilt für alle Plattformen, dass die Rechner Bestandteil eines funktionierenden Netzwerks sind und ein entsprechender Kommunikationsmechanismus vorhanden ist.

Daneben müssen die Systemvoraussetzungen für den Dialog Manager erfüllt sein.

Auf UNIX-Rechnern müssen „BSD-Sockets“ in Form von Libraries auf dem entsprechenden Rechner vorhanden sein.

### Sonstige Installationsvoraussetzungen

Motif:

- UNIX
  C-Compiler, Motif 2.0 bzw. CDE (basiert auf Motif 1.2) mit Entwicklungsumgebung (Libraries, Include Files)

mit COBOL-Schnittstelle für

- MICRO FOCUS COBOL

mit Netzwerkfähigkeit

- Netzwerk benötigt Entwicklungsumgebung (mit Libraries, Include Files)

# 9 Übersetzen und Linken

Das Programm muss wie üblich mit den richtigen Optionen für den Dialog Manager übersetzt werden.

Danach müssen die Anwendungsteile wie folgt zusammengebunden werden:

» Lokaler Teil:

1. lokale Anwendungsteile
2. dann die DM-Network Executer-Library (**libIDMndx.a** bzw. **dmndx.lib**)
3. dann die DM-Library (**libIDM.a** oder **libIDMaw.a** bzw. **dm.lib**)
4. die jeweiligen Fenstersystem-Libraries sowie Libraries für den Kommunikationsdienst

» Remote-Teil:

1. die Remote-Anwendungsteile
2. die DM-Netzwerk-Library (**libIDMnet.a**)

### Siehe auch

Kapitel „Übersetzen und Linken von DM-Programmen“ im Handbuch „C-Schnittstelle - Grundlagen“

Kapitel „Übersetzen und Linken“ im Handbuch „COBOL-Schnittstelle“

# 10 Änderung ab Version A.05.01.d

Das bisherige Verhalten des verteilten Dialog Managers (DDM) auf Netzwerkfehler mit einer sofortigen Beendigung von Client bzw. Server zu reagieren ist mit Version A.05.01.d abgeschafft.

Das Applikationsobjekt als Bindeglied und Abstraktionskonzept, um Anwendungsteile (die z.B. lokal, C/COBOL, über eine dynamische Bibliothek oder über Netzwerk verfügbar sind) für die Regelsprache zu definieren, erhält die Fähigkeit Fehler zu erkennen und angemessen darauf zu reagieren.

Folgendes kurzes Regelwerk soll das zu erwartende Verhalten des IDM's hierzu beschreiben:

- Ein Fehler ist gegeben, wenn entweder ein technischer Netzwerkfehler auftritt (Fehler die von den Netzwerksystemfunktionen gemeldet werden, z.B. Hardware-Defekte, Verbindungsverlust durch „abgebrochenen" Server-Prozess), die IDM-Versionen zwischen Client- und Server-Seite inkompatibel sind oder das IDM-Netzwerkprotokoll verletzt wird. Der letztere Fall sollte dann unbedingt dem IDM Support gemeldet werden.
  Handelt es sich bei der Applikation um eine dynamisch angebundene Bibliothek kann es sich genauso gut um eine nicht gefundene Datei handeln.
  Ebenso als Fehler zu werten ist die nicht vorhandene Anbindung für die Applikationsart (z.B. idmndx-Bibliothek wurde nicht dazugelinkt).
  Ein Abbruch des Programms z.B. durch ein „KILL"-Signal oder durch Beenden eines Prozesses über den „Task-Manager" kann nicht als Applikationsfehler erkannt werden. Die andere Seite wird dies lediglich als Netzwerkfehler gemeldet bekommen.
  Das .active-Attribut am Applikations-Objekt spiegelt, wie bisher auch, den Aktivierungszustand wieder.
- Sobald ein Fehler erkannt wird, wird die zugehörige Applikation deaktiviert. Zusätzlich wird ein Fehlercode auf den Fehlerstack gelegt.
- Für eine erfolgreich aktivierte Applikation (initial, als auch bei Umsetzen des .active-Attributes) wird ein start-Ereignis ausgelöst. Für eine Applikation, die deaktiviert wird (z.B. durch exit(), DM_StopDialog, wie auch durch Umsetzen des .active-Attributes) wird ein finish-Ereignis ausgelöst. start/finish-Ereignisse kommen initial bzw. beim Anwendungsende vor bzw. nach einem dialog start/finish, ansonsten aber asynchron!
- In den Attributen .errorcode sowie .systemerror stehen nähere Hinweise zum aufgetretenen Fehler.
- Passiert ein Fehler während der Abarbeitung einer Applikationsfunktion, so wird die Applikation deaktiviert und entweder die Simulationsregel aufgerufen oder ein Fail erzeugt.

Erkennt der DDM-Server einen Fehler, so wird nach Abarbeitung der momentanen Server-Applikationsfunktion die Netzwerk-Service-Schleife verlassen und AppFinish() aufgerufen, allerdings mit 0-IDs.

## 10.1 Konsequenzen für bisherige Dialoge

Es gibt einige Auswirkungen, die als IDM Applikationsprogrammierer berücksichtigt werden sollten.

Die IDM-Anwendung (Client) bzw. Server-Teil beendet sich **nicht** mehr, bzw. nicht mehr sofort!

Das start/finish-Ereignis wird nicht nur für eine lokale Applikation, sondern auch für Netzwerk-Applikationen und für lokale Applikationen mit dynlib-Transport ausgelöst.

Von Seiten der Regelsprache kann ein Fehler bei einer bisher vorhandenen aktiven Applikation dadurch erkannt werden, dass ein finish-Ereignis kommt und damit angezeigt wird, dass die Applikation deaktiviert wurde; außerdem durch plötzliches Fehlschlagen einer Applikations-Funktion bzw. durch Aufrufen der Simulationsregel.

Auf der Serverseite wird bei erkanntem Fehler ein AppFinish() aufgerufen, um so dem Anwendungsprogrammierer die Möglichkeit zum „Aufräumen“ zu geben. Fehler während der Abarbeitung einer Applikationsfunktion sollten korrekt behandelt werden, da die Serverfunktion noch abgearbeitet wird.

## 10.2 Anwendungsbeispiel

Folgendes Client/Server-Beispiel zeigt die Nutzung von Simulationsregeln sowie das Reagieren auf start/finish-Ereignisse.

### 10.2.1 Client

```
!! sample.dlg
dialog D

default edittext
{ .borderwidth 0; }

default statictext
{ .sensitive false; }

default window
{ on close { exit(); } }

application Appl
{
  .active false;

  integer SimCount := 0;  // Zaehler für Aufrufe von
                          // Simulationsregeln
  integer FailCount := 0;  // Zaehler für aufgetretene Fails

  function string FuncSimple(integer I)
  {
    !! Simulationsregel fuer den Fehlerfall
    this.SimCount := this.SimCount + 1;
    return "SIM-" + this.SimCount;
```

```
  }

  !! Server-Funktion, die Regel auf der Client-Seite aufruft.
  !! Nun ohne Simulationsregel, Fehler wird durch fail()
  !! abgefangen
  function string FuncComplex(object Rule, integer I);

  on start
  {
    StStatus.text := "CONNECTED";
    PbSimple.sensitive := this.active;
    PbComplex.sensitive := this.active;
    CbConnect.active := this.active;
  }
  on finish
  {
    if this.errorcode <> error_none then
      StStatus.text := "FAIL: " + this.errorcode + " - " +
                       this.systemerror;
    else
      StStatus.text := "";
    endif

    PbSimple.sensitive := this.active;
    PbComplex.sensitive := this.active;
    CbConnect.active := this.active;
  }
}

rule string Convert(integer I)
{
  return "CONV-" + I;
}

window Wi
{
  .title "DDM NetErrors";
  .width 400;
  .height 200;

  integer SimCount := 0;

  checkbox CbConnect
  {
    .width 80;
    .active false;
```

```
  .text "Connect";
  on activate
  {
    Appl.connect := EtConnect.content;
    Appl.active := true;
    EtConnect.sensitive := false;
  }
  on deactivate
  {
    EtConnect.sensitive := true;
    Appl.active := false;
  }
}

edittext EtConnect
{
  .xauto 0;
  .xleft 80;
  .content "localhost:4711";
}

statictext StStatus
{
  .ytop 30;
  .xauto 0;
}

pushbutton PbSimple
{
  .ytop 60;
  .text "Call Simple-Func";
  .sensitive false;

  on select
  {
    variable integer I;

    Appl.SimCount := 0;
    Appl.FailCount := 0;

    if fail(I := atoi(EtCount.content)) then
      I := 0;
    endif
    while(I>0) do
      if fail(StCount.text := FuncSimple(I)) then
       !! Fail tritt nur auf wenn Funktion nicht von Server
```

```
        !! angebunden wurde.
       !! Normalerweise wird Simulationsregel im Fehlerfall
        !! gerufen
       Appl.FailCount := Appl.FailCount + 1;
       StCount.text := "FAILED-" + Appl.FailCount;
      endif
      updatescreen();
      I := I-1;
    endwhile
  }
}

edittext EtCount
{
  .ytop 60;
  .xleft 200;
  .content "2000";
}

pushbutton PbComplex
{
  .ytop 90;
  .text "Call Complex-Func";
  .sensitive false;

  on select
  {
    variable integer I, FailCount:=0;
    this.window.SimCount := 0;
    Appl.SimCount := 0;
    Appl.FailCount := 0;
    if fail(I := atoi(EtCount.content)) then
      I := 0;
    endif
    !! Schleife ueber extevent ausloesen sodas
    !! Abarbeitung von finish moeglich ist
    sendevent(this,1,I);
  }

  on extevent 1(integer I)
  {
    !! Aufruf der Netzwerkfunktion die wiederum
    !! Convert-Regel aufruft.
    if (I>0) then
      if fail(StCount.text := FuncComplex(Convert,I)) then
       !! Netzwerkfehler werden ueber fail() abgefangen
```

```
          Appl.FailCount := Appl.FailCount + 1;
          StCount.text := "FAILED-" + Appl.FailCount;
        else
          I := I-1;
          sendevent(this,1,I);
        endif
        updatescreen();
      endif
    }
  }

  statictext StCount
  {
    .ytop 90;
    .xleft 200;
    .xauto 0;
  }
}
```

### 10.2.2 Server

```
// sample.c
#include <stdio.h>
#include <IDMuser.h>
#include <samplefm.h>

static trace_errors(char *txt)
{
    DM_ErrorCode errbuf[100];
    uint nerrors, i;

    nerrors = DM_QueryError(errbuf, sizeof(errbuf), 0);
    for (i=0; i<nerrors; i++)
    DM_TraceMessage("%s - error #%d", DMF_LogFile|DMF_Printf,
                    txt, errbuf[i]);
}

DM_String DML_default DM_ENTRY FuncSimple __1((DM_Integer, I))
{
    static char buf[100];

    sprintf(buf, "FUNC-%d", I);

    return buf;
}
```

```
DM_String DML_default DM_ENTRY FuncComplex __2((DM_ID, ID),
                                               (DM_Integer, I))
{
  DM_Value args[1], retval;
  DM_String str = NULL;

  args[0].type = DT_integer;
  args[0].value.integer = I;

  if (DM_CallRule(ID, ID, 1, args, &retval, 0)
      && retval.type == DT_string)
  {
    str = retval.value.string;
  }
  else
    trace_errors("callrule");
  return str;
}

int DML_c AppInit __4((DM_ID, appl), (DM_ID, dialog),
                   (int, argc), (char **, argv))
{
    DM_TraceMessage("AppInit called", DMF_LogFile);

    BindFunctions_Appl (appl, dialog, 0);

    return 0;
}

int DML_c AppFinish __2((DM_ID, appl), (DM_ID, dialog))
{
    DM_ErrorCode errbuf[100];
    uint nerrors, i;

    DM_TraceMessage("AppFinish (%d,%d) called",
                    DMF_LogFile|DMF_Printf, appl, dialog);
    trace_errors("appfinish");

    return 0;
}
```

## 10.3 Sonstige Hinweise

Grundsätzlich führt der IDM kein automatisches/periodisches Überprüfen der Netzwerkverbindung durch. Ist dies von der Anwendung gewünscht, kann dies z.B. über das Timer-Objekt realisiert

werden.

## 10.4 Änderungen am Applikations-Objekt

Es wurden die Attribute .errorcode und .systemerror neu eingeführt. Näheres siehe Objekt Application.

### 10.4.1 start/finish-Ereignis

Das start-Ereignis wird am Applikationsobjekt ausgelöst, wenn die Applikation erfolgreich (ohne Fehler) aktiviert wurde.

Das finish-Ereignis wird ausgelöst, wenn die Applikation vorher aktiv war und deaktiviert wurde, dies kann aufgrund eines Fehlers passieren, initial wenn die Aktivierung scheitert wie auch durch Beendigung der Anwendung oder Umsetzen des .active-Attributes.

Passiert ein Fehler während der Ereignis-Schleife erfolgt die Abarbeitung der start/finish-Ereignisse der Applikation entsprechend der Ereignisreihenfolge. Dies bedeutet aber auch, dass der dann aktuelle .active-Zustand nicht zwangsläufig zum Ereignis passt.

Die Aktivierung eines Applikationsobjektes, welches initial .active auf true gesetzt hat, geschieht beim Starten des Dialoges über DM_StartDialog bzw. start() aus der Regelsprache. Die start/finish-Regel wird dementsprechend vor der Abarbeitung der dialog-Start-Regel ausgeführt und so wie bisher sichergestellt, das lokale Applikationen ihre Funktionen anbinden können.

Beendet man einen Dialog, z.B. über exit(), mit einer noch aktiven Applikation wird die finish-Regel erst nach der finish-Regel des Dialoges ausgeführt.

### 10.4.2 Fehlerverhalten bei Applikationsfunktionen

Kommt es während der Abarbeitung einer Applikationsfunktion zu einem Fehler wird die Applikation deaktiviert und eine vorhandene Simulationsregel aufgerufen. Ist keine Simulationsregel vorhanden wird ein Fail an den Regelinterpreter zurückgeliefert.

## 10.5 Netzwerk-Applikationsseite

Wird während dem Aufruf einer DM-Funktion auf der Netzwerkseite ein Fehler (Netzwerkfehler oder Protokollfehler) erkannt, so kehrt die Funktion mit einem Status zurück der einen Fehler ausdrückt.

Der Netzwerk-Stub wird nach Beendigung der Server-Funktion bei einem Fehler verlassen. Die AppFinish()-Funktion wird auch aufgerufen wenn dies Aufgrund eines Netzwerk- oder Protokollfehlers passiert solang vorher ein AppInit() aufgerufen wurde. Allerdings wird keine Applikation-ID oder Module-ID mitgegeben. Auch in diesem Fall liegt ein entsprechender Fehler auf dem Fehlerstack.

Der Systemfehler-Text kann über DM_ControlEx mit der Aktion DMF_GetSystemError erfragt werden. Als data ist ein (DM_String *) zu übergeben. Wird ein TRUE zurückgeliefert ist ein Fehlertext vorhanden und der String in der Applikationcodepage in der data-Variablen abgelegt.

# Index


**A**

active 13-14
Anforderungen 25
Anwendung 10
  Aufteilung 7
Anwendungsfunktion 21
Anwendungssourcen 11
Anwendungsteil 7-8, 10-11, 22
  lokal 11
  remote 11
Anzeigeinformation 22
AppFinish 8, 11, 18
AppInit 8, 11, 18
application 10, 20
Application 13
Applikationsschnittstelle 18
AppMain 11, 18
Architektur 8
Aufteilung der Anwendung 7

**B**

Beendigung
  Anwendung 11
Benutzer 14
Benutzerschnittstelle 7, 10-11
Benutzerschnittstellenteil 8
BSD-Sockets 25

**C**

C-Funktionsprototyp 20
C-Schnittstelle 18
Client 23
COBOL-Hauptprogramm 18
COBOL-Schnittstelle 18, 25
COBOLAPPFINISH 19
COBOLAPPINIT 19
COBOLMAIN 18
connect 13-14, 22

**D**

DDM 7-8
Deklaration
  Funktionen 16
Dialog-Beschreibungssprache 13
Dialog Manager-Utility-Funktion 18
Dialogskript 10-11
Display 11
Display-Rechner 7, 10
DM-Library 26
DM-Network Executer-Library 26
DM-Netzwerk-Library 26
DM Network Executer 8, 12, 22
dm.lib 12, 26
DM_CallFunction 18
DM_GetContent 21
DM_GetValue 21
dmndx.lib 12, 26

## E

exec 13-14, 22

## F

Fallback 12
  Strategie 10
Fehler 10
Fehlerfall
  Ausweichstrategie 10
Fork-Mechanismus 23
Funktion 16
Funktionsdefinition 10
Funktionsdeklaration 20

## H

Hardware-Plattformen 25
host 14
Host 13
Host-Rechner 14

## I

Identifikator 3, 13
IDMerrfile 22
IDMlisten 23
idmndx 8, 22
IDMserve 23
IDMtracefile 22
IDMtransport 22
IDMversion 22
Initialisierung
  Anwendung 11
InitTestAppl 12
IP Adresse 13
IPv6 16
IPv6-Adresse 16
ISO OSI 8

## K

Kommandozeilenoptionen 22
Kommunikation 7, 18
Kommunikationsdienst 8, 22
Kommunikationsmechanismus 8, 25

## L

label 13
libIDM.a 12, 26
libIDMaw.a 26
libIDMndx 12
libIDMndx.a 26
libIDMnet.a 8, 12, 22, 26
Linken 26
  Anwendung 9
  DM-Programm 26
list 10-11
Listbox 21
Listen-Modus 22
local 11, 13-14
lokal 18, 22
  Anbindung 11
  Teil 26
lokaler Teil 11

**M**

Makefile 12

Micro Focus COBOL 25

Motif 12, 25

MS-Windows 12

**N**

Netzwerk 7, 10, 13, 21, 25

- Library 12, 22
- Stub-Library 9

Netzwerk-Funktionalität 22

netzwerkfähige Anwendung 8, 18

**O**

Objekt

- application 10
- Application 13

OpenSSL 9

Option -IDMlisten 22

Option -IDMserve 23

Overhead 21

**P**

Password 14

path 14

Pfad 13-14

Plattformen 25

portnumber 14, 23

Portnummer 13-14, 23

Programmname 10, 13

Protokoll 10

- Arten 22
- TCP/IP 22

**R**

Rechnerbelastung 7

Rechnername 10, 12-14

Records 20-21

remote 18

- procedure calls 8

remote-Anwendung 11

remote-Teil 11

Remote-Teil 26

**S**

Schlüssel 9

Server 7, 13, 22-23

Server-Modus 23

Simulationsprogramm idmndx 8

Source-Code 10

SSH

- Schlüssel 9
- Zertifikat 9

SSL

- Schlüssel 9
- Zertifikat 9

Start-Regel 11

Startmodi,Kommunikationsdienst 22

Startoption 22

Startoptionen 22

Stubs 8

System-Libraries 22

Systemvoraussetzungen 25

## T

Tablefield 21
TCP/IP 14
transport 13-14, 22
Transportmechanismus 13
Transportschicht 14

## U

Übersetzen 26
    DM-Programm 26
Unix 23, 25
username 14

## V

Verbindungsaufbau 10, 14, 22
Verteilter Dialog Manager 7
Voraussetzungen 25

## W

writeproto 20
+/-writetrampolin 20

## X

XXX_NETWORK 12

## Z

Zertifikat 9
Zuordnung
    Funktionen 16
zusammenlinken 26