



# Benutzer Handbuch **Administration**

## **DBAnalyser**

DBAnalyser-Version 1.2.39



---

# Inhaltsverzeichnis

<b>DBAnalyser</b>	<b>1</b>
<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>1. Ziel des Programms</b>	<b>3</b>
<b>1.1 Erste Schritte</b>	<b>3</b>
1.1.1 Erstes Starten des DBAnalysers	4
<b>2. Bedienungsanleitung</b>	<b>5</b>
<b>2.1 Starten des Programms</b>	<b>5</b>
<b>3. Funktionsweise</b>	<b>7</b>
<b>3.1 Allgemein</b>	<b>7</b>
<b>3.2 Interaktionen mit der ORACLE - Datenbank.</b>	<b>9</b>
3.2.1 Wie kann man EasyViews generieren?	10
3.2.2 Wie kann man EasyViews löschen?	10
<b>3.3 Die Fehlertypen</b>	<b>11</b>
3.3.1 Erläuterung der Fehlertypen	11
3.3.2 Die ( <i>Generic</i> ) Prüfverfahren	12
3.3.3 Die ( <i>Specific</i> ) Prüfungen	17
<b>4. Konfiguration der <i>ini</i> – Datei</b>	<b>27</b>
<b>4.1 Allgemein</b>	<b>27</b>
4.1.1 Wie definiert man eine konkrete Aufgabe?	27
4.1.2 Wie definiert man eine fehlende Referenz?	29
4.1.3 Beispiele	30
<b>4.2 Zusätzliche Hinweise</b>	<b>38</b>
4.2.1 Starten des DBAnalysers im Batch-Modus	38
<b>Index</b>	<b>39</b>

# 1. Ziel des Programms

In einer Multiserverumgebung können z. B. durch Hardware - und Netzwerkausfälle inkonsistente Daten entstehen, die nicht sofort auf der Benutzeroberfläche zu erkennen sind.

Mit dem DBAnalyser können eventuell auftretende Datenbankfehler frühzeitig erkannt, sichtbar gemacht und behoben werden.

Die Aufgabe des DBAnalysers ist es also, die PE Datenbank nach inkonsistenten Daten zu durchsuchen. Werden solche Daten gefunden, können diese dokumentiert und die Inkonsistenz behoben werden.

Durch ein regelmäßiges Ausführen des Programms in einer Produktivumgebung kann gewährleistet werden, dass die Datenbank fehlerfrei ist.

## 1.1 Erste Schritte

Die Erfahrung hat gezeigt, dass nicht alle möglichen Fehlertypen in jeder Umgebung auftreten. Deshalb ist es vorteilhaft, nach einer Anlaufphase den DBAnalyser an die jeweilige Umgebung anzupassen.

Der DBAnalyser kann nur von einem Administrator, der Zugriff auf den Server hat, benutzt werden.



### Hinweis

*Für einige Konfigurationen benötigen Sie SQL-Kenntnisse.*

*SQL ist eine ANSI (American National Standards Institute) Standardcomputersprache für den Zugriff und die Manipulation von Datenbanksystemen. SQL Statements werden verwendet, um Daten aus einer Datenbank zurückzuholen und zu aktualisieren. SQL kann beispielsweise in ORACLE, MS SQL Server, DB2, Informix, MS Access, Sybase u.v.a. Datenbanken verwendet werden.*

*Wenn Sie über keine SQL-Kenntnisse verfügen, sollten Sie ‚SQL Queries‘ nicht bearbeiten.*

- Im Verzeichnis *PPRServer\program\bin* finden Sie die ausführbare Datei **DBAnalyser.exe**.
- Über eine *.ini*-Datei kann der DBAnalyser konfiguriert werden. Sie finden beim ersten Öffnen des DBAnalyser Dateien **md\_<funktionenname>.ini**. Diese Dateien können als Vorlage dienen und auch für erste Prüfungen benutzt werden, sollten aber danach durch eine, der jeweiligen Installation angepassten Datei ersetzt werden.
- Jede Aktion des DBAnalysers wird in der Datei **DBAnalyser.log** fortlaufend protokolliert. In der Protokolldatei werden nicht nur die Aktionen des DBAnalysers festgehalten, sondern auch alle erkannten Fehler gespeichert.

### 1.1.1 Erstes Starten des DBAnalysers

☒ Check☐ Handle

Vor **der ersten Anwendung** des DBAnalysers sollten Sie alle

- Prüfungen aktivieren und
- alle Korrekturen **deaktivieren**.

⇒ Damit werden Ihnen alle gefundenen Fehler nur angezeigt. Diese Fehlerliste wird in der *DBAnalyser.log* Datei gespeichert.

Wichtig ist nun die Prüfung der gefundenen Fehler. Einige Fehlertypen setzen genaue Kenntnisse der DELMIA Process Engineer Datenstruktur voraus und benötigen eine eingehendere Untersuchung.

Sind Sie bei einigen Fehlertypen der Korrektur wegen unsicher, schicken Sie die log-Datei an den DELMIA Support. Zusammen mit dem Supportteam von DELMIA kann das Fehlerprotokoll überprüft werden. Dies kann zu eventuellen Anpassungen der Konfigurations-Datei führen. Erst danach sollte der DBAnalyser auch die Korrektur der Fehler übernehmen. Lesen Sie dazu auch den Hinweis in der [Bedienungsanleitung](#).

Wenn die Konfigurationsdatei soweit angepasst wurde (mehrere Prüfungsdurchläufe), dass alle möglicherweise auftretenden Fehler und deren Korrekturen enthalten sind, kann der DBAnalyser im Batch-Modus betrieben werden. D. h. nach dem Start des DBAnalysers werden alle Prüfungen und Korrekturen ohne eine weitere Interaktion des Anwenders durchgeführt.

Nachfolgend werden die Bedienung und die Funktionsweise des DBAnalysers beschrieben.

## 2. Bedienungsanleitung



### Achtung

**Bevor Sie den DBAnalyser starten, sichern Sie Ihre Datenbank. Eine unsachgemäße Bedienung des DBAnalyzers kann zu Datenverlusten führen.**

### 2.1 Starten des Programms

Die nachfolgend aufgeführten Schritte sollten nur dann durchgeführt werden, wenn Sie bereits Erfahrung mit dem DBAnalyser und dem Datenmodell des DELMIA Process Engineers besitzen.

1. **Konfiguration der ini-Datei:** Konfigurieren Sie eine der Datei `md_<funktionsname>.ini` wie im Kapitel „[Funktionsweise](#)“ beschrieben. Sie finden die Datei im Verzeichnis `PPRServer\program\bin`. Für den ersten Start sollten alle

**Handle=false**

gesetzt werden.

Mit den Dateien `md_<funktionsname>.ini` können Sie die Prüfverfahren konfigurieren, aber auch die Anzeigewerte des Dialogs DBAnalyser beeinflussen. Damit ein Prüfverfahren ausgeführt wird, müssen zwei Bedingungen erfüllen werden:

- Es muss in der ini-Datei konfiguriert sein und
- das entsprechende Kontrollkästchen muss ebenfalls aktiviert sein.

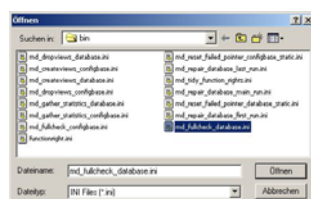
Inaktive Kontrollkästchen werden übersprungen, auch wenn sie konfiguriert wurden.

2. **Beenden aller PPR Server Prozesse:** Beenden Sie alle PPR Server Prozesse.



3. **Starten des DBAnalyzers:** Starten Sie das Programm DBAnalyser.exe im `PPRServer\program\bin` Verzeichnis.

- ⇒ Es öffnet sich ein Datei-Selektor, in dem Sie die ini-Datei selektieren. Es werden Ihnen mehrere **`md_<funktionsname>.ini`** Dateien zur Auswahl angeboten. Aus dem Dateinamen können Sie bereits die unterschiedlichen Funktionen der einzelnen Dateien ablesen. Um aber die einzelnen Funktionen zu bearbeiten, müssen Sie die Dateien mittels eines Editors öffnen. Im Abschnitt [Konfiguration der ini – Datei](#) wird die Bearbeitung der ini – Datei beschrieben.



**Abbildung 1:** Datei-Selektor für die ini-Datei

⇒ Nach der Auswahl öffnet sich der DBAnalyser - Dialog.

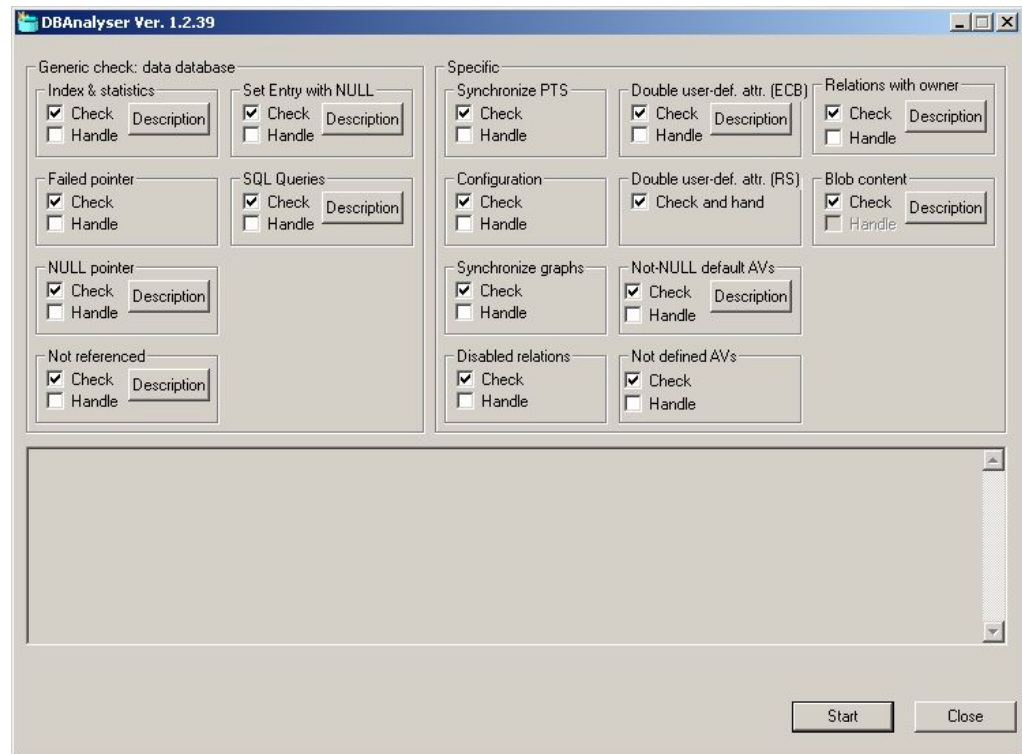


Abbildung 2: DBAnalyser - Dialog

4. Während der Analyse-Phase darf *kein* PPR Server gestartet werden.
5. **Starten der Prüfung:** Über den Button **Start** wird die Prüfung gestartet. Im unteren Teil des Dialogs wird Ihnen der Fortschritt der Prüfung angezeigt. Werden Fehler während einer Prüfung gefunden, wird für jeden Prüfungstyp (z. B. NULL pointer usw.) eine Auflistung aller Fehler angezeigt.
6. **Korrektur der Fehler:** Sie können die vorgeschlagenen Änderungen durchführen, wenn, wie bereits in der Einleitung erwähnt, eine sorgfältige Prüfung der Fehler stattgefunden hat.
7. **Neustart:** Wurden vom DBAnalyser Fehler gefunden und korrigiert, sollte der DBAnalyser nochmals gestartet werden (Schritt 3). Dafür müssen Sie den DBAnalyser zuerst schließen und wie unter Schritt 3 beschrieben nochmals starten.

Alle erkannten Fehler werden in der Datei **DBAnalyser.log** protokolliert. Die Protokolldatei benötigen Sie auch, wenn Sie sich an den DELMIA Support wenden.

## 3. Funktionsweise

### 3.1 Allgemein

Der DBAnalyser verfügt über zwei unterschiedliche Gruppen von Prüfverfahren:

#### Generic

und

#### Specific

**Abbildung 3:** Die unterschiedlichen Prüfverfahren

Um die unterschiedlichen Fehlertypen zu finden und (bei Bedarf) zu korrigieren, werden die Datenbanken (*database*) des DELMIA Manufacturing Hub nach vorgegebenen Kriterien durchsucht. Eine Korrektur führt nicht unbedingt zu einer Vervollständigung der Daten, dies hängt vielmehr von dem jeweiligen Fehlertyp ab.

- Die *Generic* Prüfverfahren können mit jeder Datenbank arbeiten und können mittels der **ini-Datei** konfiguriert werden.
- Die *Specific* Verfahren können nicht immer konfiguriert werden.

☒ Check

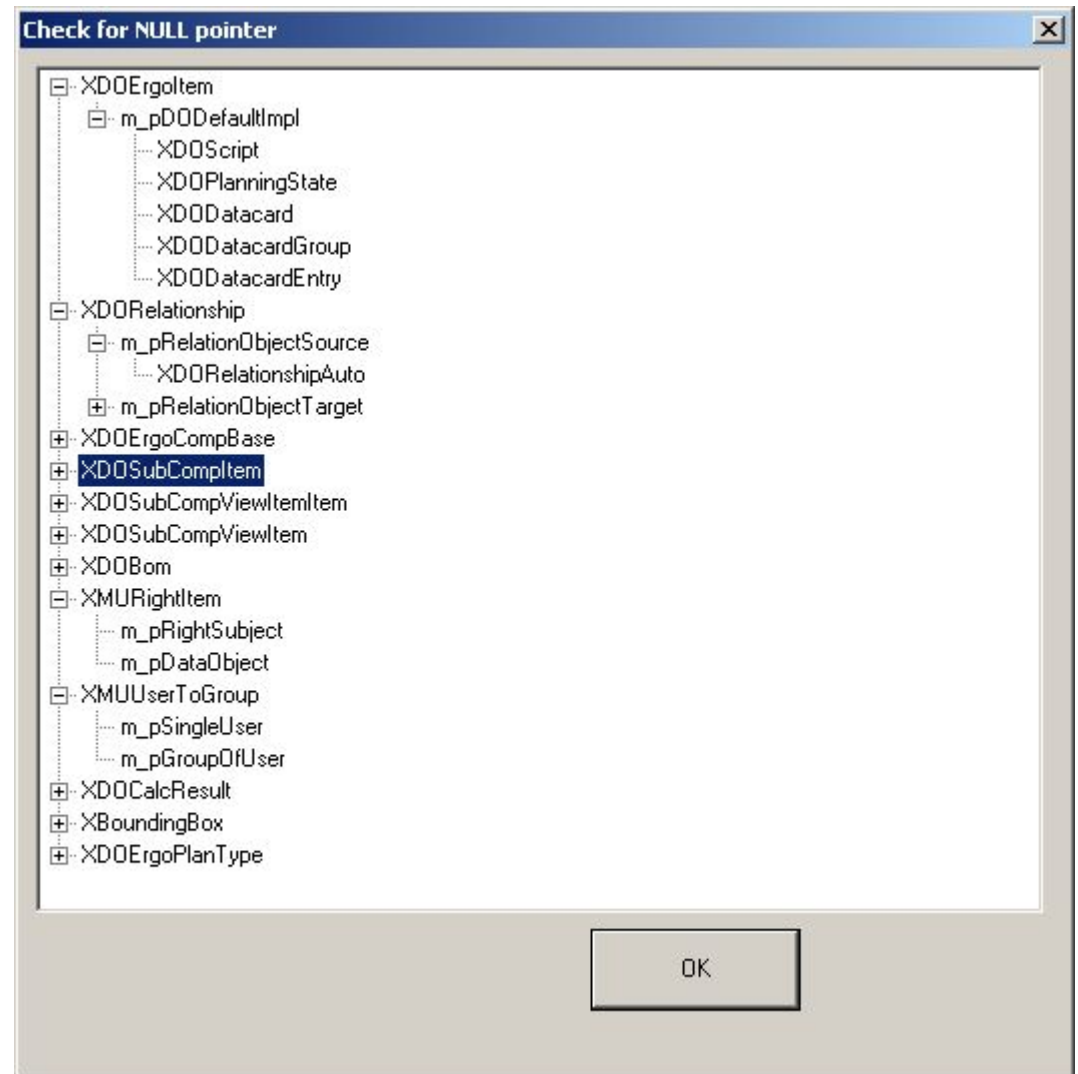
Mit dem Kontrollkästchen **Check** aktivieren Sie die Prüfung des jeweiligen Fehlertyps. Ohne Check wird auch kein *Handle* ausgeführt, auch wenn das Kontrollkästchen aktiviert ist.

☒ Handle

Mit dem Kontrollkästchen **Handle** aktivieren Sie, zusätzlich zur Prüfung, auch die Möglichkeit zur Korrektur des jeweiligen Fehlertyps. Um die Korrekturmöglichkeit zu erhalten, muss auch Check aktiviert werden.

Description

Über den Button **Description** können Sie die Einstellungen zur Fehlersuche zu diesem Fehlertyp aufrufen. Diese Einstellungen werden über die **ini-Datei** gesteuert. Siehe auch [Abbildung 4](#).



**Abbildung 4:** Dialog nach dem aktivieren des Buttons *Description* (NULL pointer)

Der DBAnalyser kann auch im Batch-Modus betrieben werden. Näheres dazu finden Sie unter [Starten des DBAnalyzers im Batch-Modus](#).



## 3.2 Interaktionen mit der ORACLE - Datenbank.

Der PPRServer (wird in anderem Zusammenhang auch IPDServer genannt) kommuniziert mit einer ORACLE - Datenbank, in der alle Daten dauerhaft gespeichert werden. Die Kommunikation erfolgt über eine POET – Schnittstelle.

Die Schnittstelle setzt feste, von der Version unabhängige Klassen - und Attributnamen in die versionsabhängigen ORACLE – Tabellen- und Spaltennamen um. Die ORACLE – Namen sind leider recht kryptisch und nicht leicht zu merken.

Manchmal ist es sehr hilfreich, direkt in ORACLE Abfragen auszuführen, um bestimmte Daten aus der Datenbank zu erhalten. SQL ist viel mächtiger als die vom POET verwendete OQL - Sprache. Außerdem gibt es auch andere Regeln, die man beachten muss, um z. B. alle XDOErgoCompBase – Objekte zu erhalten, bei denen ein m\_pErgoProject – Attribut einen bestimmten Wert hat.

Um diese Kommunikation mittels SQL zu vereinfachen, wurde der DBAnalyser um eine Funktionalität - den EasyViews - erweitert. Was verbirgt sich hinter den EasyViews?

Für jede POET – Klasse wird zusätzlich eine Ansicht (View) angelegt, die alle Instanzen der Klasse zurückliefert und genauso heißt wie die Klasse selbst. Die einzelnen Spalten, die diese View enthält, heißen genauso wie die entsprechenden POET – Attribute.

Zusätzlich enthält jedes Objekt zwei zusätzliche Attribute, die die POET - ID abbilden, und zwar:

- die **OID** (datenbankweite, eindeutige Nummer) und
- die **CID** (Klassen – ID).

Einmal angelegt, können EasyViews sowohl vom DBAnalyser, als auch von jedem ORACLE Client Programm (z.B. SQLPlus) benutzt werden.

Erst nachdem EasyViews angelegt wurden, sind Abfragen wie im nachfolgenden Beispiel möglich:

Als Ergebnis einer Abfrage sollen alle XDOErgoCompBase – Objekte, die zum Projekt „My first project“ gehören, aber nicht in einer Stückliste verwendet werden, zurückgegeben werden.

### Beispiel

```
SELECT ecb.OID, ecb.CID, ecb.m_name
FROM XDOErgoCompBase ecb, XDOErgoProject proj
WHERE
ecb.m_pErgoProject = proj.OID AND
proj.m_name = 'my first project' AND
ecb.OID NOT IN (SELECT m_pErgoCompBase FROM XDOSubCompItem);
```

### 3.2.1 Wie kann man EasyViews generieren?

Es stehen Ihnen zwei Möglichkeiten zur Verfügung.

8. Der [COMMON\_DATA] Abschnitt wird um eine Zeile erweitert (siehe auch: [Konfiguration der ini – Datei](#)):

```
CreateEasyViews=true
```

9. Oder Sie starten den DBAnalyser mit der Datei [md\\_createviews\\_database.ini](#) (bzw. [md\\_createviews\\_configbase.ini](#)). Beide Dateien werden vom Setup ins das *PPRServer/program/bin* Verzeichnis kopiert.  
Der DBAnalyser braucht zum Erzeugen der EasyViews ca. 2 – 4 Minuten (dies ist von der Hardware des Computers abhängig und nicht von der Größe der Datenbank).

### 3.2.2 Wie kann man EasyViews löschen?

Auch hier stehen Ihnen zwei Möglichkeiten zur Verfügung.

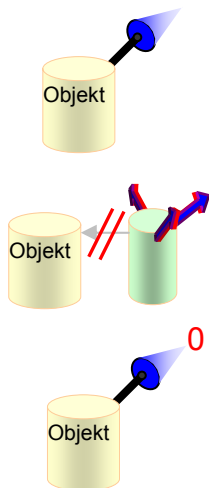
1. Ebenso wie beim Erzeugen erweitern Sie den [COMMON\_DATA] Abschnitt um eine Zeile (siehe auch: [Konfiguration der ini – Datei](#)):

```
CreateEasyViews=false
```

2. Oder Sie starten den DBAnalyser mit der Datei [md\\_dropviews\\_database.ini](#) (oder [md\\_dropviews\\_configbase.ini](#)). Diese Dateien finden Sie ebenfalls im Verzeichnis *PPRServer/program/bin*. Der DBAnalyser benötigt zum Löschen der EasyViews weniger als eine Minute. Auch dies ist von der verwendeten Hardware abhängig.

## 3.3 Die Fehlertypen

### 3.3.1 Erläuterung der Fehlertypen



#### Unterschied zwischen 'Nicht Referenziert' und 'NULL Pointer' und 'korrupte Referenz'

- Zeigt ein Objekt auf ein anderes, nicht existentes Objekt, ist dieser Zeiger eine korrupte Referenz. Zeigt ein XDOErgoCompBase Objekt also auf einen nicht existierenden Planungstypen, besitzt es eine korrupte Referenz.
- Wird ein Objekt nicht von anderen Objekten referenziert, zeigt also kein Zeiger *auf* das Objekt, ist es NOT\_REFERENCED.
- Zeigt ein Objekt auf eine Zahl 0 in einer Tabelle (nicht "NULL"), besitzt es einen 'NULL Pointer'.

#### Korrupte Referenzen:

Wenn ein Datenbank-Objekt gelöscht wird, werden alle Referenzen, die auf das Objekt zeigen, auf NULL gesetzt. Zusätzlich wird das Objekt aus allen Sets entfernt. Bei auftretenden Fehlern entstehen korrupte Referenzen.

#### Fehlende Referenzen:

Einige Objekte sind nur dann vollständig und verwendbar, wenn sie mit bestimmten anderen Objekten verknüpft sind.

#### Beispiele fehlender Referenzen:

- XDOErgoCompBase braucht unbedingt *einen* Zeiger auf das Projekt und *einen* Zeiger auf den Planungstyp.
- XDORelationship braucht immer *zwei* Zeiger: auf das Source-Objekt und auf das Target-Objekt.
- XDOErgoItem braucht (fast immer; Ausnahmen müssen in der ini – Datei spezifiziert werden) *einen* Zeiger auf das Objekt, an dem das XDOErgoItem hängt.

Weitere Fälle sind eher Release- und kundenspezifisch.

Nachfolgend werden alle Fehlertypen beschrieben. Die Schlüsselgruppe benötigen Sie bei der Konfiguration.

### 3.3.2 Die (Generic) Prüfverfahren

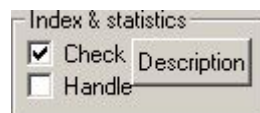
Die *Generic* Prüfverfahren können mit jeder Datenbank arbeiten und werden mittels der **ini-Datei** konfiguriert. Hiermit passen Sie die ini-Datei sukzessive an Ihre DPE - Installation an. Es gibt sechs *Generic* Prüfverfahren. Bei den *Generic* Prüfverfahren können Sie die zu durchsuchende Datenbank (DataBaseType=**CONFIGBASE** oder DataBaseType=**DATABASE**) festlegen.

#### 3.3.2.1 Index & statistics

(Schlüsselgruppe [INDEX\_STATISTICS]);

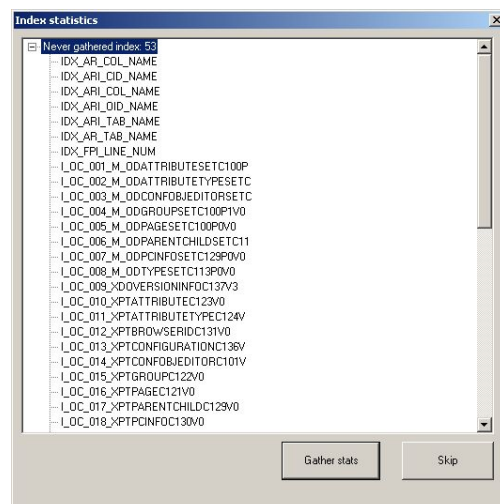
Überprüft:

- ob alle notwendigen Indexe auch tatsächlich da sind;
- den Zeitpunkt, an dem Statistiken zu den vorhandenen Indexen berechnet wurden.



Statistiken sollten von einem Datenbank-Administrator regelmäßig ermittelt und fehlende Indexe dann neu generiert werden. Diese Arbeiten kann nur ein Datenbank-Administrator ausführen.

⇒ **Reparatur:** Die Erstellung von Statistiken kann auch der DBAnalyser übernehmen – wenn der Button „**Gather stats**“ im Ergebnis-Fenster betätigt wird.



⇒ Dabei wird für jeden Index folgender Befehl ausgeführt:  
**DBMS\_STATS.GATHER\_SCHEMA\_STATS(database\_name, estimate\_percent => Prozentsatz, cascade=> TRUE);**  
 Der **Prozentsatz** wird in der INI-datei definiert und muss zwischen 1 und 99 Prozent liegen.

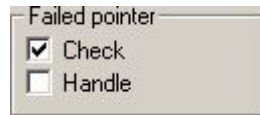
**Beispiel:**

```
[INDEX_STATISTICS_OPTIONS]
estimate_percent=8
```

### 3.3.2.2 Failed pointer

(Schlüsselgruppe [FAILED\_POINTER]);

- findet alle Datenobjekte mit ungültigen (korrupten) Referenzen.

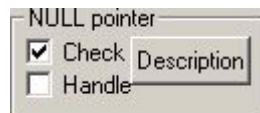


⇒ **Reparatur:** Alle ungültigen (korrupten) Referenzen werden auf Null gestellt.

### 3.3.2.3 NULL pointer

(Schlüsselgruppe [NULL\_POINTER]);

- findet alle Datenobjekte einer gegebenen Klasse mit einem NULL Zeiger

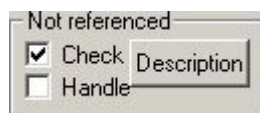


⇒ **Reparatur:** Alle gefundenen Objekte werden gelöscht.

### 3.3.2.4 Not referenced

(Schlüsselgruppe [IS\_NOT\_REFERENCED]);

- findet alle Datenobjekte einer gegebenen Klasse, die nicht mindestens von einem Objekt einer gegebenen Klasse referenziert werden

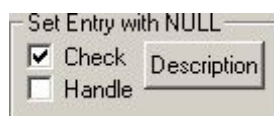


⇒ **Reparatur:** Alle gefundenen Objekte werden gelöscht.

### 3.3.2.5 Set Entry with NULL

(Schlüsselgruppe [SET\_ENTRY\_WITH\_NULL]);

- findet alle Datenobjekte mit Sets, die korrupte Referenzen besitzen



⇒ **Reparatur:** Alle gefundenen Objekte werden, je nach Konfiguration der .ini – Datei, gelöscht oder verändert.

**Anmerkungen zu Set Entry with NULL**

Einige Klassen des Datenmodells besitzen Sets, die transiente Objekte enthalten können.

Dazu zählen z. B.:

- XDOTimeAnalysisLineContainer/m\_timeAnalysisLineSet
- XDOTimeAnalysisCoordinateContainer/m\_timeAnalysisCoordinateSet
- m\_setExposedLinks/XExposedLinksSet

Solche Sets können Objekte enthalten, die korrupte Referenzen besitzen. Weil diese Objekte nicht dauerhaft sind, werden sie weder von FAILED\_POINTER oder NULL\_POINTER, noch von SQL\_QUERY gefunden.

SET\_ENTRY\_WITH\_NULL muss aber immer zuerst konfiguriert werden.

**Wie wird SET\_ENTRY\_WITH\_NULL konfiguriert?**

Anhand zweier Beispiele kann die Konfiguration erklärt werden:

**Beispiel 1:**

ClassName=*XExposedLinksSet*

SetName=*m\_setExposedLinks*

PointerToTest=*m\_pDODefaultImpl*

NullAllowed=*false*

Action=*DeleteSetEntry*

- ⇒ Bei allen Objekten der Klasse *XExposedLinksSet* werden die Einträge im Set *m\_setExposedLinks* analysiert.
- ⇒ Mit *PointerToTest=m\_pDODefaultImpl* wird jeder Eintrag als fehlerhaft angesehen, der mit dem Zeiger *m\_pDODefaultImpl* auf ein nicht mehr existierendes Objekt zeigt oder eine Null enthält (*NullAllowed=false*).
- ⇒ Alle fehlerhafte Einträge werden gelöscht (*Action=DeleteSetEntry*)

**Beispiel 2:**

ClassName=*XDOTimeAnalysisLineContainer*

SetName=*m\_timeAnalysisLineSet*

PointerToTest=*m\_pSubCompltem*

NullAllowed=*true*

Action=*SetToNull*

- ⇒ Bei allen Objekten der Klasse *XDOTimeAnalysisLineContainer* werden Einträge im Set *m\_timeAnalysisLineSet* analysiert.
- ⇒ Als fehlerhaft gilt jeder Eintrag, der mit dem Zeiger *m\_pSubCompltem* auf ein nicht mehr existierendes Objekt zeigt. (Null-Werte sind erlaubt, weil *NullAllowed=true*).

- ⇒ Bei allen fehlerhaften Einträgen soll der o. g. Zeiger auf Null gesetzt werden (*Action=SetToNull*)

### 3.3.2.6 SQL Queries

(*Schlüsselgruppe [SQL\_QUERY]*);

- Führt SQL Abfragen aus. Nur für SQL- und DELMIA Datenmodell - Kennern zu empfehlen. Beispiele finden Sie in den *ini*-Dateien.



- ⇒ **Reparatur:** Alle gefundenen Objekte werden gelöscht.

#### Zusätzliche Anmerkungen zu SQL Queries

Wenn Sie Abfragen in SQL formulieren, werden Ihnen POET – Objekte zurückgeliefert. Die gefundenen Objekte werden anschließend angezeigt (es ist konfigurierbar, ob nur die Objekt-ID oder auch zusätzliche Eigenschaften, wie z. B. der Name, angezeigt werden). Alle gefundenen Objekte können (bei Bedarf) in einer Aktion gelöscht werden.

Eine SQL-Abfrage kann sowohl auf die POET - Views als auch auf die Easy-Views zuzugreifen.

- ⇒ Es wird empfohlen, immer auf EasyViews zuzugreifen, weil:
- einerseits die Fehlersuche einfacher ist (keine kryptischen Namen),
  - andererseits kann die gleichen Abfrage, ohne den Abfragentext anzupassen, in unterschiedlichen Versionen verwendet werden (z. B. PE12, PE13 und PE14).



#### Hinweis

Zum Erzeugen von EasyViews stehen Ihnen die beiden *ini* - Dateien *md\_createviews\_database.ini* und die *md\_createviews\_configbase.ini* zur Verfügung.

Diese Views können mit den beiden *ini* - Dateien *md\_dropviews\_configbase.ini* und *md\_dropviews\_database.ini* wieder gelöscht werden.

#### Anforderungen

Die SQL Abfrage kann beliebig komplex sein und alle Möglichkeiten von ORACLE SQL ausschöpfen. Dabei muss nur **eine** Anforderung berücksichtigt werden:

Es müssen immer **zwei Spalten** zurückgeliefert werden. Diese Spalten müssen die OID und die CID der gesuchten Objekte enthalten.

**Wie wird eine SQL Abfrage konfiguriert?**

<b>QueryName</b>	Kurze Bezeichnung. Die Bezeichnung wird im Ergebnisfenster und in der Log-Datei angezeigt. Sollte möglichst kurz verfasst werden.
<b>QueryDescription</b>	Ist für die Notation da und enthält die ausführliche Beschreibung, was die Abfrage eigentlich machen soll.
<b>QueryBody</b>	Enthält die eigentliche Abfrage. Darf nur eine Zeile belegen (max 2048 Zeichen). Kann mit einem Semikolon abgeschlossen werden.
<b>LoadFullDescription</b>	<b>true</b> oder <b>false</b> . Gibt an, ob für die gefundenen Objekte die Beschreibung angezeigt wird oder nicht.  Nicht für alle Objekte ist es sinnvoll eine Beschreibung zu erzeugen – sie kostet zusätzlich Zeit und Speicher.

**Beispiel:**

[SQL\_QUERY\_1]

**QueryName**=SCI between ECBs from two different projects (except WSCs)**QueryDescription**=SCI between ECBs from two different projects (except WSCs)

**QueryBody**=SELECT sci.OID, sci.CID FROM XDOSubCompltem sci, XDOBom bom, XDOErgoCompBase ecb1, XDOErgoCompBase ecb2 WHERE sci.m\_pBom = bom.OID AND sci.m\_pErgoCompBase = ecb2.OID AND bom.m\_pParentDO = ecb1.OID AND ecb1.m\_pErgoProject <> ecb2.m\_pErgoProject AND ecb2.m\_pErgoProject > 0

**LoadFullDescription**=true

[SQL\_QUERY\_2]

**QueryName**=Old XChangeEntry**QueryDescription**=Old XChangeEntry's. You can change or state more precisely date (e.g. with time)

**QueryBody**=SELECT oid, cid FROM XChangeEntry WHERE m\_dtdate < to\_date('2000-08-10', 'yyyy-mm-dd')

**LoadFullDescription**=false



### 3.3.3 Die (*Specific*) Prüfungen

Einige *Specific* Prüfverfahren können ebenfalls konfiguriert werden. Bei den *Specific* Prüfverfahren können Sie jedoch die Datenbank ([CONFIGBASE](#) oder [DATABASE](#)) nicht festlegen. Sie müssen nicht bei jedem Start des DBAnalyzers ausgeführt werden. Beispielsweise muss die Synchronisation der Planungstypensätze nur dann überprüft werden, wenn ein Import (Projekt, PTS oder Konfiguration) ausgeführt wurde oder während der Bearbeitung eines Planungstypensatzes Betriebsstörungen auftraten.

#### 3.3.3.1 Synchronize PTS:

*Schlüsselgruppe [SYNCHRONIZE\_PTS]);*

- Überprüft alle Planungstypensätze (PTS).
- Alle in Projekten verwendeten Planungstypensätze (Slave - Planungstypensätze) müssen die gleiche Anzahl von Planungstypen wie der Planungstypensatz in der Systembibliothek (Master - Planungstypensatz) haben.



⇒ **Reparatur:** Alle fehlenden Slave Planungstypen werden erzeugt.

- Zu jedem Master - Planungstyp (XDOErgoPlanType) Objekt existiert genau ein XPtType Objekt.

⇒ **Reparatur:** Alle fehlenden XPtType Objekte werden erzeugt.

#### 3.3.3.2 Configuration

*Schlüsselgruppe [CONFIGURATION]);*

Es werden alle Objekte in der *CONFIGBASE* und *DATABASE* überprüft, die Typen und Eltern–Kind–Beziehungen definieren. Dabei wird geprüft:

- **a)** ob es zu jedem Master – XDOErgoPlanType - Objekt ein XPtType – Objekt existiert;
- **b)** ob es zu jedem XPtType - Objekt ein XDOErgoPlanType – Objekt oder eine POET - Klasse existiert;
- **c)** ob jedes XPtType – Objekt entweder mit dem „*m\_odSuperType*“ oder mit dem „*m\_strBaseClassName*“ auf ein gültiges XPtType – Objekt (oder 0) zeigt;
- **d)** ob alle XPtType – Objekte eindeutige Namen haben;
- **e)** ob alle XPtParentChild – Objekte eindeutig sind (Kombination aus *m\_strChildListName*, *m\_parent* und *m\_child* soll eindeutig sein).

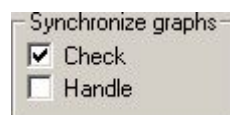


- ⇒ **Reparatur:** Alle Objekte, die nach dem Kriterium b), d) und e) als fehlerhaft identifiziert werden, werden im Ordner „obsolet“ angezeigt und können vom DBAnalyser automatisch gelöscht werden (Button „Delete (Obsolet only)“).
- ⇒ **Reparatur:** alle anderen kommen in den Ordner „redundant“ und müssen bei Bedarf mit dem Konfigurationswerkzeug oder anderen DPE – Mitteln repariert werden.

### 3.3.3.3 Synchronize graphs:

*Schlüsselgruppe [TOO\_MANY\_GRAPH\_BOMS]);*

- Überprüft, ob zu jedem XDODGraph Objekt genau ein *XDOSubCompViewItemListPro* (GraphBOM) Objekt existiert.

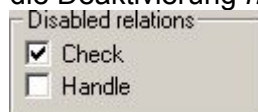


- ⇒ **Reparatur:**
  - Alle fehlenden *XDOSubCompViewItemListPro* Objekte werden erzeugt;
  - Alle überflüssigen *XDOSubCompViewItemListPro* Objekte werden gelöscht;
  - Alle fehlenden XDODGraph Objekte werden erzeugt;

### 3.3.3.4 Disabled relations:

*(Schlüsselgruppe [DISABLED\_RELATIONS]);*

- Überprüft in der Konfiguration, ob jedes XDORelationship Objekt eine aktivierte Eltern-Kind-Beziehung besitzt. Hintergrund: Sie können Eltern-Kind-Beziehungen deaktivieren. Damit ist diese Beziehung nicht mehr sichtbar aber nach wie vor in der Datenbank vorhanden, sie wird durch die Deaktivierung *nicht* gelöscht.



- ⇒ **Reparatur:** Löscht die XDORelationships, deren Eltern-Kind-Beziehungen deaktiviert sind;

### 3.3.3.5 Double user-def. Attr. (ECB):

(Schlüsselgruppe [USER\_DEF\_ATTR\_ECB]);



#### Hinweis

Diese Suche kann nur unter der Voraussetzung gestartet werden, wenn zuvor so genannte EasyViews erzeugt wurden. Hierfür stehen Ihnen die beiden ini - Dateien [md\\_createviews\\_database.ini](#) und die [md\\_createviews\\_configbase.ini](#) zur Verfügung.

Diese Views können mit den beiden ini - Dateien [md\\_dropviews\\_configbase.ini](#) und [md\\_dropviews\\_database.ini](#) wieder gelöscht werden.

- Überprüft, ob ein XDOAttributeValue mit dem gleichen Pendant in XDOErgoCompBase existiert. Dies ist der Fall, wenn benutzerdefinierte Attribute und vorkonfigurierte Attribute für den gleichen Sachverhalt verwendet werden (zum Beispiel: 'name' und 'm\_name').



⇒ **Reparatur:** Löscht alle gefundenen XDOAttributeValues.

Sie können gezielt einzelne Attribute von der Prüfung ausschließen (siehe auch [Struktur einer INI-Datei](#)).

### 3.3.3.6 Double user-def. Attr. (RS):

(Schlüsselgruppe [OTHER\_PROBLEMS]);

- Überprüft, ob die Attribute 'name', 'nameshort', 'modificationdate' oder 'pprloaderhistoricalid' als „benutzerdefiniertes Attribut“ gekennzeichnet sind (CONFIGBASE).

⇒ **Reparatur:** Markiert dieses Attribut als 'nicht benutzerdefiniert';

- Überprüft, ob die XDOAttributeValue Objekte, die den oben erwähnten Attributen entsprechen, auf jedem XDORelationship Objekt existieren.

⇒ **Reparatur:** Überträgt die Werte auf das XDORelationship Objekt und löscht die entsprechenden XDOAttributeValue Objekte.

### 3.3.3.7 Not-NULL default AV's:

(Schlüsselgruppe [DEFAULT\_AVS]);

Ab der Version PE 5.13 SP5 bis einschließlich der Version PE 5.14 wurden in der Datenbank für spezielle Attribute der Ergokomponenten keine Werte (AV's) angelegt. Dies betraf alle Attribute vom Typ

Integer oder Double deren Wert 0 war und

Stringattribute deren Wert leer war.

Wird in diesen Versionen der Standardwert dieser Attribute geändert (ungleich 0 oder nicht leer) und neue Komponenten die diese Attribute benutzen erzeugt, stimmen die Werte der Datenbank und die der Konfiguration nicht mehr überein.

#### Beispiel:

- ❑ In Attribut\_22 war der Standardwert ein Leerstring, also kein Eintrag.
- ⇒ Alle Objekte die dieses Attribut verwenden und in denen nicht explizit der Wert des Attributes geändert wurde, haben keinen Eintrag in Attribut\_22.
- ❑ Der Standardwert für Attribut\_22 wird auf „Abteilung x“ verändert, also kein Leerstring mehr.
- ⇒ Alle Objekte die nach der Änderung erzeugt werden und dieses Attribut verwenden, haben als Wert den Eintrag „Abteilung x“ in Attribut\_22.

Obwohl in der Konfiguration ein Standardwert für Attribut\_22 steht, existieren nun Objekte in der Datenbank die unterschiedliche Werte besitzen.

Mit Hilfe der *Schlüsselgruppe [Not-Null default AVs]* ist es möglich, die Werte in der Datenbank, denen der Konfiguration anzupassen. In der ini- Datei wird angegeben, welche Attribute für eine Korrektur zu berücksichtigen sind. Vorgesehen ist eine solche Korrektur insbesondere für neu hinzugekommene Attribute.



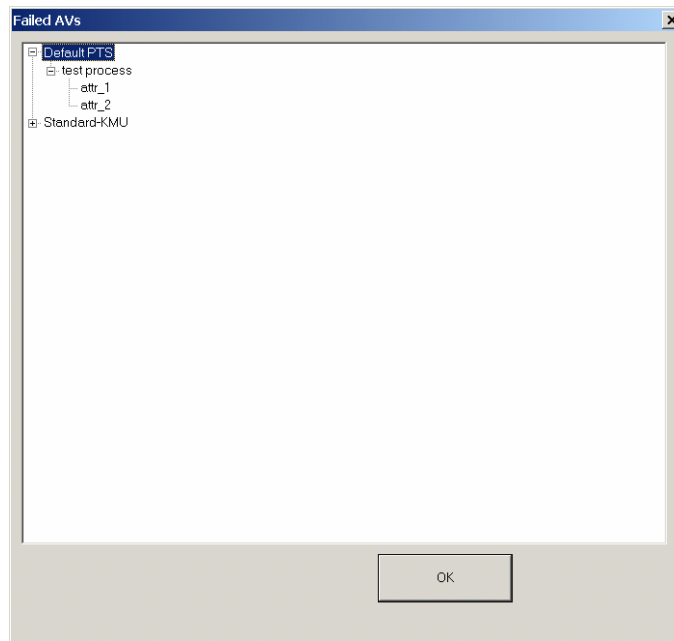
- ⇒ **Reparatur:** Alle Einträge, der in der ini-Datei angegebenen Attribute, in denen kein Wert in der Datenbank vorhanden ist, werden mit dem Standardwert der Konfiguration überschrieben.

#### Beispiel:

```
[DEFAULT_AVS_AV_DATA_1]
Plantypeset=Default PTS
Plantype=test process
NumberOfAttributes=2
AttributeName_1=attr_1
AttributeName_2=attr_2
```

In diesem Fall werden die Attribute **attr\_1** und **attr\_2** für Komponenten überprüft, die auf dem Planungstyp ‚test process‘ im Planungstypensatz ‚Default PTS‘ aufbauen.

- Über das Kontrollkästchen ‚Check‘ erhält man darüber Informationen, welche AV's der verschiedenen Ergokomponenten betroffen sind.
- Ist ‚Handle‘ aktiviert, werden die AV's auf den Standardwert der Konfiguration gesetzt.



### 3.3.3.8 Not defined AVs

(Schlüsselgruppe [NOT\_DEFINED\_AVs]);

Durchsucht die Datenbank nach benutzerdefinierten Attributen, die nicht mehr verwendet werden.

In DELMIA Process Engineer® ist es möglich benutzerdefinierte Attribute für bestimmte Ergokomponententypen zu definieren. Solche Attribute machen den DELMIA Process Engineer® einerseits flexibler, andererseits langsamer und speicherhungriger. Nichtbenötigte benutzerdefinierte Attribute können gelöscht werden und sind deswegen nicht mehr im Konfigurationsmanager sichtbar. Aber das entsprechende Datenbankobjekt existiert weiterhin, auch wenn ein Attribut wegkonfiguriert (gelöscht) wurde, somit wird kein Speicherplatz gewonnen. Weitaus verwirrender ist die Tatsache, dass bei einer Neuanlage eines weiteren Attributes mit demselben Name wie das gelöschte Attribut, die alten Werte wieder sichtbar werden.



**Reparatur:** löscht, oder besser gesagt bereinigt die Datenbank von allen benutzerdefinierten Attributen, die nicht mehr in der Konfiguration oder dem Planungstypensatz erscheinen.

### 3.3.3.9 Relations with owner

(Schlüsselgruppe [RELATIONSHIP\_OWNER\_OPTIONS]);

Ab PE5.16 haben alle Relationen ein Attribut *m\_pOwner*. Im Konfigurationsmanager ist an dem entsprechenden XPtParentChild Objekt definiert, auf welches Objekt dieses Attribut zeigen soll. Dabei kann man zwischen 5 Varianten auswählen:

- **No Owner** (*m\_pOwner* = 0)
- **Source Owner** (*m\_pOwner* = *m\_pRelationObjectSource*);
- **Target Owner** (*m\_pOwner* = *m\_pRelationObjectTarget*);
- **Explicit Owner** (*m\_pOwner* zeigt auf jedes beliebige dodefaultimpl Objekt);
- **Common Parent Owner** (*m\_pOwner* zeigt auf die Ergokomponente, zu deren Stücklisteneinträgen sowohl das Source- als auch das Target - Objekt gehören -> gemeinsamer Vater).

Durch Änderungen der Konfiguration oder durch Server-Fehler kann es vorkommen, dass *m\_pOwner* nicht auf das Objekt zeigt, auf das es zeigen sollte. *Relations with owner* durchsucht die Konfiguration nach Objekten, deren Attribut *m\_pOwner* auf falsche Objekte zeigt.



- ⇒ **Reparatur:** Für NoOwner relationship: der Zeiger wird auf Null gesetzt.
- ⇒ **Reparatur:** Für SourceOwner relationship: der Zeiger wird auf Source-Objekt gesetzt.
- ⇒ **Reparatur:** Für TargetOwner relationship: der Zeiger wird auf Target-Objekt gesetzt.
- ⇒ **Reparatur:** Für ExplicitOwner relationship: in der ini – Datei wird definiert ob der Zeiger ignoriert, auf das Projekt zeigt oder gelöscht wird.
- ⇒ **Reparatur:** Für die CommonParentOwner relationship werden zwei Fälle unterschieden:
  1. *Owner ist Null:* in der ini – Datei wird definiert ob der Zeiger ignoriert, auf das Projekt zeigt oder gelöscht wird.
  2. *Owner ist nicht Null und ist nicht common parent:* in der ini – Datei wird definiert ob der Zeiger ignoriert, auf das Projekt oder den einzigen Vater zeigt, oder gelöscht wird.



#### Hinweis

Wenn am Attribut *m\_pOwner* Änderungen in der Konfiguration gemacht wurden, sollte der DBAnalyser immer gestartet und die Relationen geprüft werden.

**Wie wird SET\_ENTRY\_WITH\_NULL konfiguriert?**

3. Zuerst legen Sie fest, ob die Funktionsgruppe SET\_ENTRY\_WITH\_NULL gestartet werden soll.

[RELATIONSHIP\_OWNER]

Check=true

Handle=false

4. Danach definieren Sie, ob alle oder nur ausgewählte Relationen geprüft werden sollen.

5. Im letzten Schritt müssen Sie festlegen was mit fehlerhaften Relationen geschehen soll, wenn eine Reparatur nicht möglich ist.

Dazu ein Beispiel aus der INI - Datei:

[RELATIONSHIP\_OWNER\_OPTIONS]

RelationTypes=\*

ActionByExplicitOwner=SetToProject

ActionByCommonParentWithoutOwner=SetToUniqueCommonParent

ActionByCommonParentWithWrongOwner=SetToUniqueCommonParent

Die einzelnen Positionen:

1. **RelationTypes**: entweder \* (d. h. alle Relationen werden geprüft) oder einzelne Relationstypen die durch Semikolon getrennt werden, z. B.:  
RelationTypes=process\_runsbefore\_process;nodes
2. **ActionByExplicitOwner**: Definiert, was mit jeder ExplicitOwner Relation passieren soll, die ein leeres m\_pOwner Attribut hat.  
Folgende Optionen stehen zur Verfügung:
  - **Ignore**: Relation wird nicht modifiziert
  - **SetToProject**: m\_pOwner wird auf das Projekt zeigen, zu dem auch das Source - Objekt (oder dessen Vater, falls Source selbst keine Ergokomponente ist) gehört
  - **Delete**: Relation wird gelöscht
3. **ActionByCommonParentWithoutOwner**: Definiert, was mit jeder CommonParentOwner Relation bei der m\_pOwner = 0 ist, geschehen soll
4. **ActionByCommonParentWithWrongOwner**: Definiert, was mit jeder CommonParentOwner Relation geschehen soll, die mit m\_pOwner auf ein Objekt zeigt, das nicht der gemeinsame Vater von Source und Target ist (d. h. in der Owner - Stückliste fehlt entweder das Source- oder Target-Objekt oder beide). Folgende Optionen stehen zur Verfügung:
  - **SetToUniqueCommonParent**: m\_pOwner wird auf die Ergokomponente zeigen, in derer Stückliste sowohl Source- als auch Target- Objekte auftauchen. Damit der Vater als "UNIQUE" anerkannt wurde, darf es nur ein solches Objekt geben. Es ist durchaus möglich, dass es den UniqueCommonParent gar nicht gibt.
  - **Ignore**: Relation wird nicht modifiziert

- ▶ **SetToProject:** m\_pOwner wird auf das Projekt zeigen, zu dem auch das Source - Objekt (oder dessen Vater, falls Source selbst keine Er-gokomponente ist) gehört.
- ▶ **Delete:** Relation wird gelöscht.

Alle mit Problemen behafteten Relationen, werden nach Typen sortiert. Pro Typ wird ein Ordner angelegt und die entsprechenden Relationen angezeigt.

Es sind insgesamt 8 Ordner möglich:

- NoOwner: relations with owner (action: set to null)
- SourceOwner: relations without owner (action: set to source)
- TargetOwner: relations without owner (action: set to target)
- ExplicitOwner: relations without owner (action: s. ActionByExplicitOwner)
- CommonParentOwner: relations without owner (action: siehe ActionBy-CommonParentWithoutOwner)
- CommonParentOwner: relations with wrong owner (action: siehe Action-ByCommonParentWithWrongOwner)
- CommonParentOwner: relations without owner (set to common parent impossible)
- CommonParentOwner: relations with wrong owner (set to common parent impossible)

Die letzten zwei Ordner können nur dann existieren, wenn

**ActionByCommonParentWithoutOwner = SetToUniqueCommonParent**

und

**ActionByCommonParentWithWrongOwner = SetToUniqueCommonParent** ist.

⇒ In allen anderen Fällen ist eine Reparatur immer möglich.

Die Unterscheide zwischen **CommonParentWithoutOwner** und **CommonParentWithWrongOwner** erfolgt aus folgendem Grund:

Ein **CommonParentWithoutOwner** Fehler tritt auf, wenn ein NoOwner Relationstyp zu einem CommonParentOwner umkonfiguriert wird. In diesem Fall gehen Informationen verloren, wenn die Relation einfach gelöscht wird. Deshalb ist es sinnvoll diese Relationen einzeln zu betrachten und dem richtigen Zeiger manuell (oder mit Hilfe eines Skriptes) zu setzen.

Ein **CommonParentWithWrongOwner** Fehler tritt auf, wenn eine oder mehrere ungültige Relationen gefunden werden. Entsprechende Relationen sollten dann einfach gelöscht werden.



### 3.3.3.10 Blob content:

(Schlüsselgruppe [BLOB\_CONTENT]);

Durchsucht Blob's in denen Skripte abgelegt wurden.

Wird bei einem Versions-Wechsel eine Skript-Funktion durch eine andere ersetzt, ist es sehr aufwendig, alle Skripte zu finden, die davon betroffen sind. Mit Hilfe der Schlüsselgruppe [BLOB\_CONTENT] haben Sie die Möglichkeit, alle betroffenen Einträge zu finden.



⇒ **Reparatur:** Überschreibt oder löscht KEINE Einträge. Sie erhalten lediglich die Information in welchem Skript und an welcher Stelle der gesuchte Eintrag auftritt. Skripte sind häufig vom verwendeten Planungstypensatz und der Konfiguration abhängig. Deshalb sollte jeder Eintrag vor dem Austausch individuell überprüft werden.

Für jedes Paar ClassName – AttributeName wird ein Ordner angelegt, in dem alle gefundenen Einträge (nach ItemName sortiert) abgelegt werden. Zu jedem Vorkommen werden Information über die Position im Blob mitgespeichert (Zeile + Spalte).

-> Sie finden alle gefundenen Einträge im *Ergebnisfenster* und in der *Log-Datei*.

#### Wie muss die .ini-Datei aufgebaut sein?

<b>ClassName</b>	definiert, welche POET– Klasse analysiert wird (XDOScript).
<b>AttributeName</b>	definiert, welches Attribut analysiert wird.
<b>NumberOfItems</b>	Anzahl der gesuchten Begriffe.
<b>Item_ /</b>	definiert den Name des I-ten Begriffs;
<b>Comment_ /</b>	Ein Kommentartext. Wird zu jedem gefundenen Begriff ausgegeben.
<b>CaseSensitive_ /</b>	<b>true</b> oder <b>false</b> , Definiert, ob bei der Suche Groß-Kleinschreibung berücksichtigt werden soll.
<b>WholeWordOnly_ /</b>	<b>true</b> oder <b>false</b> . Definiert, ob bei der Suche nur komplette Wörter berücksichtigt werden sollen

#### Beispiel allgemein:

```
[BLOB_CONTENT_POET_CLASS_1]
ClassName=XDOScript
AttributeName= m_script
NumberOfItems=2
Item_1=OldFunction_1
Comment_1=not exist (replace with NewFunction_2)
CaseSensitive_1=true
WholeWordOnly_1=true
Item_2= DummyFunction_3
Comment_2=performance problem?
CaseSensitive_2=false
WholeWordOnly_2=false
```

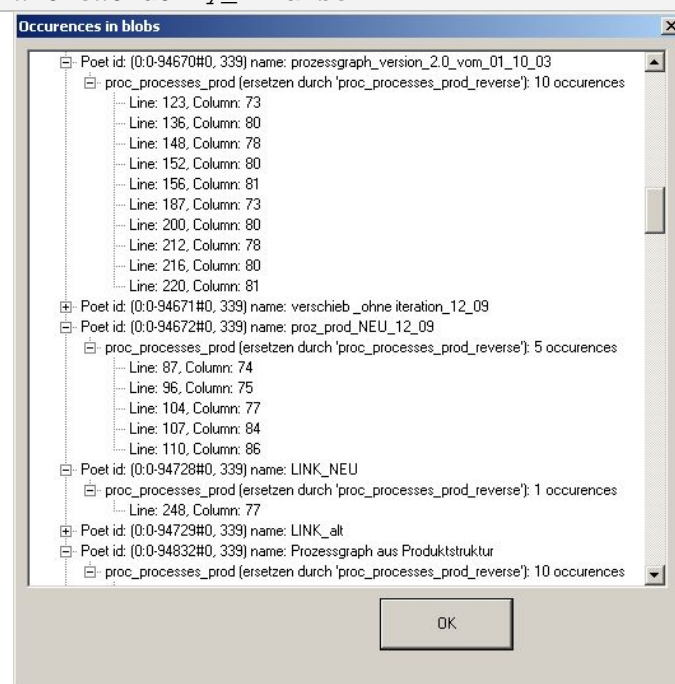
```
[BLOB_CONTENT_POET_CLASS_2]
ClassName=XDOScriptVariable
AttributeName=m_blobScript
NumberOfItems=2
Item_1=ModifiedFunction_4
Comment_1=One attribute more
CaseSensitive_1=true
WholeWordOnly_1=true
Item_2=RemovedFunction_5
Comment_2=not exist (use another algorithm)
CaseSensitive_2=false
WholeWordOnly_2=false
```

**Beispiel:**

```
[COMMON_DATA]
DataBaseType=DATABASE
Interactive=true
CreateEasyViews=false
DropEasyViews=false

[BLOB_CONTENT]
Check=true
Handle=false

[BLOB_CONTENT_POET_CLASS_1]
ClassName=XDOScript
AttributeName=m_script
NumberOfItems=1
Item_1=proc_processes_prod
Comment_1=ersetzen durch 'proc_processes_prod_reverse'
CaseSensitive_1=true
WholeWordOnly_1=false
Item_2=String 2
Comment_2=performance problem?
CaseSensitive_2=false
WholeWordOnly_2=false
```



## 4. Konfiguration der *ini* – Datei

### 4.1 Allgemein

Der DBAnalyser kann über eine *.ini*-Datei konfiguriert werden.

Je nachdem was Sie mit dem DBAnalyser prüfen wollen, können Sie unterschiedliche ini - Dateien erstellen.

Grundsätzlich kann man zwei unterschiedliche Typen von ini - Dateien unterscheiden:

- Ini - Dateien die die **Konfiguration** überprüfen und
- Ini - Dateien die die **Datenbank**überprüfen

Der Aufbau ist in beiden ini - Dateien der gleiche. Sie unterscheiden sich nur im ersten Abschnitt, der [COMMON\_DATA].

#### 4.1.1 Wie definiert man eine konkrete Aufgabe?

Die konkrete Aufgabe wird durch Anlegen einer INI-Datei definiert. Diese INI-Datei muss alle Einzelheiten enthalten.

##### 4.1.1.1 Struktur einer INI-Datei

###### [COMMON\_DATA]

Der erste Abschnitt enthält die [COMMON\_DATA].

⇒ Hier geben Sie den Typ der zu prüfenden Datenbank an.

Der Schlüssel ist **DataBaseType**, mit den zwei möglichen Werten

- **DATABASE** und
- **CONFIGBASE**

In der dritten Zeile wird mittels **Interactive=true** oder **false** festgelegt, ob Benutzereingaben erfolgen sollen (**=true**) oder nicht (**=false**).

Mit (**=false**) wird nur der Dialog angezeigt, es können aber keine Änderungen vorgenommen werden. Diese Option ist nur dann sinnvoll, wenn der DBAnalyser im Batch-Modus oder über Nacht betrieben wird, da alle Fehler sofort, ohne Nachfrage, nach der Prüfung behoben - oder fehlerhafte Objekte gelöscht werden (wenn **Handle=true**).

###### Anlegen der Schlüsselgruppen

In den nächsten Abschnitten werden die Schlüsselgruppen angelegt.

Als Fehlertyp können derzeit folgende Schlüsselgruppen eingegeben werden:


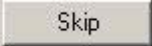
[INDEX\_STATISTICS]

[FAILED\_POINTER]

[NULL\_POINTER]  
 [IS\_NOT\_REFERENCED]  
 [SQL\_QUERY]  
 [SYNCHRONIZE\_PTS]  
 [TOO\_MANY\_GRAPH\_BOMS]  
 [OTHER\_PROBLEMS]  
 [CONFIGURATION]  
 [DISABLED\_RELATIONS]  
 [USER\_DEF\_ATTR\_ECB]  
 [BLOB\_CONTENT]

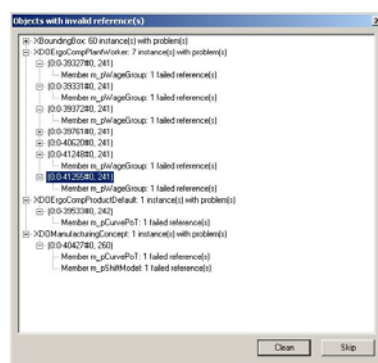
In den nächsten beiden Zeilen wird festgelegt, wie die Schlüsselgruppe behandelt werden soll.

#### Normaler Modus

- ☐ Mit **Check** = true wird nach dem angegebenen Fehlertyp gesucht. Wenn ‚false‘ eingetragen wird, wird diese Prüfung übersprungen.
- ☐ Mit **Handle** = true wird bei dem angegebenen Fehlertyp zusätzlich zur Prüfung auch die Möglichkeit zur Korrektur des jeweiligen Fehlertyps angeboten. Beispielsweise können im Falle des Fehlertyps *Failed pointer* alle fehlerhaften Referenzen auf NULL gesetzt, im Falle des Fehlertyps *NULL pointer* alle Objekte mit fehlenden Referenzen gelöscht oder im Falle des Fehlertyps *Synchronize PTS* alle Planungstypensätze synchronisiert werden. Wenn ‚false‘ eingetragen wird, können die auftretenden Fehler nicht korrigiert werden (der Button **Delete** ist inaktiv  ).

Um die Korrekturmöglichkeit zu erhalten, muss auch Check aktiviert werden.

- ➔ Mit dem Button **Skip** kann zur nächsten Prüfung übergangen werden, ohne dass Korrekturen vorgenommen wurden. Siehe auch [Abbildung 5](#).



**Abbildung 5:** Dialog, wenn Fehler gefunden wurden

#### Batch Modus

Im **Batch-Modus** werden mit *Handle* = true alle gefundenen Fehler sofort korrigiert. Mit *Handle* = false wird keine Korrektur vorgenommen. Es werden lediglich Informationen über die gefundenen Probleme in die *DBAnalyser.log* Datei eingetragen.

## 4.1.2 Wie definiert man eine fehlende Referenz?

### 4.1.2.1 Beispiel für eine einzelne Definition

- ☉ Es sollen alle Instanzen der Klasse "**ClassToTest**" und den davon abgeleiteten Klassen gesucht werden, bei denen in der Membervariable "**m\_Member1**" ein NULL – Wert steht.
- ☉ Dabei sollen alle Instanzen der Klassen "**ClassException1\_1**", "**ClassException1\_2**", ..., "**ClassException1\_L**" ignoriert werden.
- ☉
  - Die Suche soll für die Membervariable "**m\_Member2**" wiederholt werden.
  - Überspringe dabei alle Instanzen der Klassen "**ClassException2\_1**", "**ClassException2\_2**", ..., "**ClassException2\_M**".
- ☉
  - Die Suche soll für die Membervariable "**m\_MemberK**" wiederholt werden.
  - Überspringe dabei alle Instanzen der Klassen "**ClassExceptionK\_1**", "**ClassExceptionK\_2**", ..., "**ClassExceptionK\_N**".

⇒ Die Definition wird so abgebildet:

#### [NULL\_POINTER\_POET\_CLASS\_1]

ClassName= **ClassToTest**

NumberOfPointerToTest=K

PointerToTest\_1= **m\_Member1**

NumberOfExceptionClassesForPointer\_1=L

ExceptionClass\_1\_ForPointer\_1= **ClassException1\_1**

ExceptionClass\_2\_ForPointer\_1= **ClassException1\_2**

...

ExceptionClass\_**L**\_ForPointer\_1= **ClassException1\_L**

PointerToTest\_2= **m\_Member2**

NumberOfExceptionClassesForPointer\_2=**M**

ExceptionClass\_1\_ForPointer\_2= **ClassException2\_1**

ExceptionClass\_2\_ForPointer\_2= **ClassException2\_2**

...

ExceptionClass\_**M**\_ForPointer\_2= **ClassException2\_M**

...

PointerToTest\_**K**= **m\_MemberK**

NumberOfExceptionClassesForPointer\_**K**=N

ExceptionClass\_1\_ForPointer\_**K**= **ClassExceptionK\_1**

ExceptionClass\_2\_ForPointer\_**K**= **ClassExceptionK\_2**

...

ExceptionClass\_**N**\_ForPointer\_**K**= **ClassExceptionK\_N**

Für die nächste Klasse wird dann ein Abschnitt mit der Bezeichnung [NULL\_POINTER\_POET\_CLASS\_2] angelegt usw.

Wird die Reihenfolge unterbrochen, also auf...POET\_CLASS\_1 folgt ...POET\_CLASS\_3, ignoriert der DBAnalyser diesen Eintrag und sucht nach der nächsten Schlüsselgruppe, z. B. [IS\_NOT\_REFERENCED\_POET\_CLASS\_1].

### Unterschiede in den ini – Dateien

Die Bezeichnungen sind in beiden ini – Dateien, die Konfiguration oder die Datenbank betreffend, die gleichen. Sie unterscheiden sich nur in der zweiten Zeile: DataBaseType=CONFIGBASE oder DataBaseType=DATABASE (und bei den Klassen- und Attributnamen).

## 4.1.3 Beispiele

### 4.1.3.1 Beispiel für NULL Pointer

Von der Klasse XDOErgoltem sind viele weitere Klassen abgeleitet. Es sollen nun alle Instanzen der Klasse "**XDOErgoltem**" und den davon abgeleiteten Klassen gesucht werden, bei denen der Pointer (Zeiger) "**m\_pDODefaultImpl**" ein NULL – Wert ist.

Welche Klasse soll untersucht werden?

⇒ **XDOErgoltem**

Wie viele Pointer (Zeiger) werden gesucht?

⇒ 1

Welcher Pointer ist dies?

⇒ **m\_pDODefaultImpl**

Wie viele Klassen sollen bei der Prüfung unberücksichtigt bleiben? (Die Anzahl ist immer anzugeben, auch wenn keine Klasse ausgeschlossen wird. Die Anzahl muss dann Null sein; *NumberOfExceptionClassesForPointer\_1=0*)

⇒ 4

Wenn die Anzahl der unberücksichtigten Klassen größer als Null ist, dann: welche Klasse(n) soll(en) nicht berücksichtigt werden?

⇒ **XDOScript**  
**XDOPlanningState**  
**XDODatacard**  
**XDODatacardGroup**

In der ini-Datei sieht dies so aus:

```
[NULL_POINTER_POET_CLASS_1]
ClassName=XDOErgoltem
NumberOfPointerToTest=1
PointerToTest_1=m_pDODefaultImpl
NumberOfExceptionClassesForPointer_1=4
ExceptionClass_1_ForPointer_1=XDOScript
ExceptionClass_2_ForPointer_1=XDOPlanningState
ExceptionClass_3_ForPointer_1=XDODatacard
ExceptionClass_4_ForPointer_1=XDODatacardGroup
```

Wenn weitere Pointer betrachtet werden sollen, müssen die danach aufgelistet werden, z. B. PointerToTest\_2=m\_XY.

#### 4.1.3.2 Beispiele für NOT\_REFERENCED

1

XDOBlob braucht *eine* Referenz auf das Objekt **XBlob**. Ohne XDOBlob und den Zeiger kann ein **XBlob** Objekt nicht gelesen werden und belegt unnötigen Speicherplatz.

Welche Klasse soll untersucht werden?

⇒ **XBlob**

Wie viele Klassen referenzieren diese Klasse?

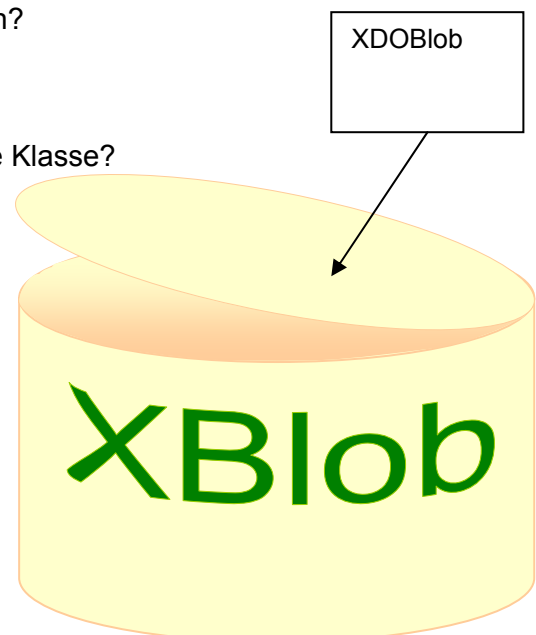
⇒ **1**

Welche Klasse(n) ist (sind) dies?

⇒ **XDOBlob**

Welche Referenz ist es?

⇒ **m\_realBlob**



In der ini-Datei sieht dies so aus:

```
[IS_NOT_REFERENCED_POET_CLASS_1]
ClassName=XBlob
NumberOfReferencingClasses=1
ReferencingClass_1=XDOBlob
ReferencingPointer_1=m_realBlob
```

2

Die Klasse *XDOAttributeValueSet* (die Klasse der benutzerdefinierten Attribute) wird von mehreren Klassen referenziert. Ein *XDOAttributeValueSet* muss von einer anderen Klasse referenziert werden. Also wenn z. B. ein *XDOErgoCompBase* - Objekt gelöscht wird, wird auch das zugehörige *XDOAttributeValueSet* gelöscht.

Welche Klasse soll untersucht werden?

⇒ *XDOAttributeValueSet*

Wie viele Klassen referenzieren diese Klasse?

⇒ 3

Welche Klasse(n) ist (sind) dies?

⇒ *XDOErgoCompBase*  
*XDORelationship*  
*XDOCalcResult*

Welche Referenzen werden verwendet?

⇒ *m\_pAttributeValueSet*  
*m\_pAttributeValueSet*  
*m\_pAttributeValueSet*



### Achtung

*In diesem Beispiel kann man neben dem reinen Editieren der ini-Datei noch weitere Besonderheiten sehen. Wird eine referenzierende Klasse weggelassen oder vergessen, interpretiert der DBAnalyser alle Objekte, die diese Klasse referenzieren als Fehler und löscht die Objekte der Klasse *XDOAttributeValueSet*.*

In der ini-Datei sieht dies so aus:

```
[IS_NOT_REFERENCED_POET_CLASS_2]
ClassName=XDOAttributeValueSet
NumberOfReferencingClasses=3
ReferencingClass_1=XDOErgoCompBase
ReferencingPointer_1=m_pAttributeValueSet
ReferencingClass_2=XDORelationship
ReferencingPointer_2=m_pAttributeValueSet
ReferencingClass_3=XDOCalcResult
ReferencingPointer_3=m_pAttributeValueSet
```



**Beispiel für eine INI-Datei:**

```
[COMMON_DATA]
DataBaseType=DATABASE
DataBaseType=CONFIGBASE
Interactive=true
CreateEasyViews=true
DropEasyViews=false

[INDEX_STATISTICS]
Check=false
Handle=true

[FAILED_POINTER]
Check=false
Handle=true

[NULL_POINTER]
Check=true
Handle=true

[IS_NOT_REFERENCED]
Check=true
Handle=true

[SQL_QUERY]
Check=true
Handle=false

[IS_NOT_IN_A_SET]
Check=false
Handle=true

[SYNCHRONIZE_PTS]
Check=true
Handle=false

[TOO_MANY_GRAPH_BOMS]
Check=true
Handle=true

[CONFIGURATION]
Check=true
Handle=true

[DISABLED_RELATIONS]
Check=true
Handle=false

[USER_DEF_ATTR_ECB]
Check=true
Handle=false

[DEFAULT_AVS]
Check=true
Handle=false

[NOT_DEFINED_AVS]
Check=true
Handle=false
```

```
[OTHER_PROBLEMS]
Transfer_User_Defined_Attributes=true
[INDEX_STATISTICS_OPTIONS]
estimate_percent=8

[IS_NOT_REFERENCED_POET_CLASS_1]
ClassName=XBlob
NumberOfReferencingClasses=1
ReferencingClass_1=XDOBlob
ReferencingPointer_1=m_realBlob

[IS_NOT_REFERENCED_POET_CLASS_2]
ClassName=XDOAttributeValueSet
NumberOfReferencingClasses=3
ReferencingClass_1=XDOErgoCompBase
ReferencingPointer_1=m_pAttributeValueSet
ReferencingClass_2=XDORelationship
ReferencingPointer_2=m_pAttributeValueSet
ReferencingClass_3=XDOCalcResult
ReferencingPointer_3=m_pAttributeValueSet

[NULL_POINTER_POET_CLASS_1]
ClassName=XDOErgoItem
LoadFullDescription=true
NumberOfPointerToTest=1
PointerToTest_1=m_pDODefaultImpl
NumberOfExceptionClassesForPointer_1=2
ExceptionClass_1_ForPointer_1=XDOScript
ExceptionClass_2_ForPointer_1=XDOPlanningState

[NULL_POINTER_POET_CLASS_2]
ClassName=XDORelationship
LoadFullDescription=true
NumberOfPointerToTest=2
PointerToTest_1=m_pRelationObjectSource
NumberOfExceptionClassesForPointer_1=1
ExceptionClass_1_ForPointer_1=XDORelationshipAuto
PointerToTest_2=m_pRelationObjectTarget
NumberOfExceptionClassesForPointer_2=1
ExceptionClass_1_ForPointer_2=XDORelationshipAuto

[NULL_POINTER_POET_CLASS_3]
ClassName=XDOErgoCompBase
LoadFullDescription=true
NumberOfPointerToTest=2
PointerToTest_1=m_pErgoProject
NumberOfExceptionClassesForPointer_1=1
ExceptionClass_1_ForPointer_1=XDOErgoSysElement
PointerToTest_2=m_pPlanType
NumberOfExceptionClassesForPointer_2=0

[NULL_POINTER_POET_CLASS_4]
ClassName=XDOSubCompltem
LoadFullDescription=false
NumberOfPointerToTest=2
PointerToTest_1=m_pErgoCompBase
NumberOfExceptionClassesForPointer_1=0
PointerToTest_2=m_pBom
```

```
NumberOfExceptionClassesForPointer_2=0
[NULL_POINTER_POET_CLASS_5]
ClassName=XDOSubCompViewItemItem
LoadFullDescription=false
NumberOfPointerToTest=1
PointerToTest_1=m_pSubCompItem
NumberOfExceptionClassesForPointer_1=0

[NULL_POINTER_POET_CLASS_6]
ClassName=XDOSubCompViewItem
LoadFullDescription=false
NumberOfPointerToTest=1
PointerToTest_1=m_pDODefaultImpl
NumberOfExceptionClassesForPointer_1=1
ExceptionClass_1_ForPointer_1=XDOSubCompView

[NULL_POINTER_POET_CLASS_7]
ClassName=XDOBom
LoadFullDescription=false
NumberOfPointerToTest=1
PointerToTest_1=m_pParentDO
NumberOfExceptionClassesForPointer_1=0

[SQL_QUERY_1]
QueryName=Lost AV sets
QueryDescription=AV Sets, die von keinem Objekt (Ergokomponente, Relation oder CalcResult) referenziert werden
QueryBody=SELECT OID, CID FROM XDOAttributeValueSet WHERE OID IN (SELECT OID FROM XDOAttributeValueSet MINUS( SELECT m_pAttributeValueSet FROM XDOErgoCompBase UNION SELECT m_pAttributeValueSet FROM XDORelationShip UNION SELECT m_pAttributeValueSet FROM XDOPCalcResult));
LoadFullDescription=false

[SQL_QUERY_2]
QueryName=Lost blobs
QueryDescription=XBlob, die von keinem XDOBlob referenziert werden
QueryBody=SELECT OID, CID FROM XBlob WHERE OID IN (SELECT OID FROM XBlob MINUS SELECT m_realBlob FROM XDOBlob);
LoadFullDescription=false

[CHECK_USER_DEF_ATTR_ECB_CLASS_1]
ClassName=XDOErgoCompProcessDefault
NumberOfIgnoredAttributes=2
IgnoredAttr_1=dummy_1
IgnoredAttr_2=dummy_2

[BLOB_CONTENT_POET_CLASS_1]
ClassName=XDOScript
AttributeName=m_source
NumberOfItems=2
Item_1=string 1
Comment_1=not exist
CaseSensitive_1=true
WholeWordOnly_1=true
Item_2=String 2
Comment_2=performance problem?
CaseSensitive_2=false
WholeWordOnly_2=false
```

**[BLOB\_CONTENT\_POET\_CLASS\_2]**

ClassName=XDOScriptVariable  
AttributeName=m\_blobScript  
NumberOfItems=2  
Item\_1=dummy\_1  
Comment\_1=Dummy comment 1  
CaseSensitive\_1=true  
WholeWordOnly\_1=true  
Item\_2=dummy\_2  
Comment\_2=Dummy comment 2  
CaseSensitive\_2=true  
WholeWordOnly\_2=true

**[DEFAULT\_AVS\_AV\_DATA\_1]**

Plantypeset=Default PTS  
Plantype=st  
NumberOfAttributes=2  
AttributeName\_1=attr\_1  
AttributeName\_2=attr\_2

**[DEFAULT\_AVS\_AV\_DATA\_2]**

Plantypeset=Default PTS  
Plantype=test process  
NumberOfAttributes=2  
AttributeName\_1=attr\_1  
AttributeName\_2=attr\_2

**Beispiel für DataBaseType = CONFIGBASE****[COMMON\_DATA]**

DataBaseType=CONFIGBASE  
Interactive=true  
CreateEasyViews=false  
DropEasyViews=false

**[FAILED\_POINTER]**

Check=true  
Handle=false

**[NULL\_POINTER]**

Check=true  
Handle=false

**[IS\_NOT\_REFERENCED]**

Check=false  
Handle=false

```
[SQL_QUERY]
Check=false
Handle=false

[IS_NOT_IN_A_SET]
Check=false
Handle=false

[SYNCHRONIZE_PTS]
Check=false
Handle=false

[TOO_MANY_GRAPH_BOMS]
Check=false
Handle=false

[PROJECT_OF_ECB_AND_PT]
Check=false
Handle=false

[DISABLED_RELATIONS]
Check=false
Handle=false

[OTHER_PROBLEMS]
Transfer_User_Defined_Attributes=false

[NULL_POINTER_POET_CLASS_1]
ClassName=XPtParentChild
NumberOfPointerToTest=2
PointerToTest_1=m_parent
PointerToTest_2=m_child
NumberOfExceptionClassesForPointer_1=0
```

## 4.2 Zusätzliche Hinweise

### 4.2.1 Starten des DBAnalysers im Batch-Modus

Wenn Sie zur Bearbeitung der ini-Datei eine Vorlage benötigen, öffnen Sie die Eingabeaufforderung (Die MS-DOS-Eingabeaufforderung wurde ab Windows 2000 in Eingabeaufforderung umbenannt und befindet sich im Menü *Zubehör*).

- Wie in [Abbildung 6](#) dargestellt, können Sie im Verzeichnis des DBAnalyser über den Schalter **/h** oder **/?** erfahren welche Parameter Ihnen zur Verfügung stehen.

```
DELMIA_R14\PPRServer\program\bin>dbanalyser /h
```

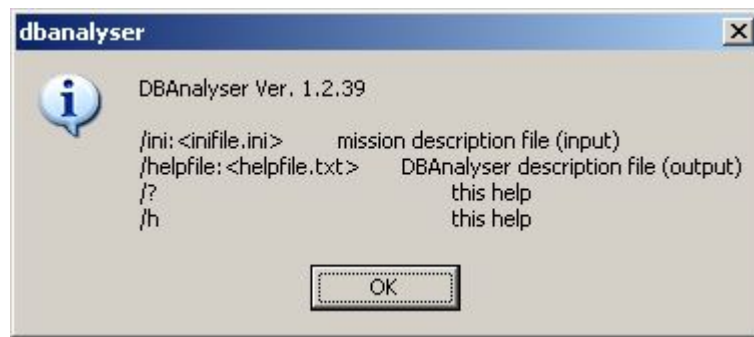


Abbildung 6: Eingabeaufforderung

- Mit **ini: <dateiname.ini>** können Sie den DBAnalyser mit der angegebenen ini-Datei (dateiname.ini) starten.
- Mit **helpfile: <dateiname.txt>** können Sie im Verzeichnis *...program/bin*, mit dem vorhin eingetragenen Namen (*dateiname*) eine Textdatei erzeugen, die Ihnen als Vorlage dienen kann und zusätzliche Beschreibungen zu den jeweiligen Fehlertypen enthält.

```
DELMIA_R13\PPRServer\program\bin>dbanalyser /helpfile:dbanalyser_help.txt
```

# Index

## B

Batch-Modus .....	4, 8, 28, 38
Button Delete.....	28
Button Description .....	7
Button Gather stats .....	12
Button Skip .....	28

## C

Check .....	7, 28
COMMON_DATA .....	27
CONFIGBASE .....	27

## D

DATABASE_POET_CLASS_1 .....	29
DBAnalyser.log.....	3, 4, 6
Delete .....	28

## E

EasyViews .....	10
EasyViews. generieren.....	10
EasyViews. löschen .....	10
Eingabeaufforderung .....	38

## F

fehlende Referenz .....	29, 30
Fehlertypen .....	11
0 (Null) Zeiger.....	13
BLOB_CONTENT.....	25
CONFIGURATION .....	17
Disabled relations .....	18
Double user-def. Attr. (ECB).....	19
Fehlende Referenzen .....	13
Fehlende Referenzen .....	11
Index & statistics.....	12
Keine Referenzen.....	13
Korrumpte Referenzen .....	11
Not defined AVs.....	21
Not-NULL default AV's.....	20
'NULL Pointer'.....	11
Relations with owner .....	22
SET_ENTRY_WITH_NULL.....	13
SQL Queries.....	15
Anforderungen .....	15
Aufbau von.....	16
Synchronize graphs.....	18
SYNCHRONIZE_PTS .....	17

## G

Generic .....	7, 13
Generic Prüfverfahren.....	12

## H

Handle.....	7, 28
helpfile.....	38

## I

ini – Datei .....	27
auswählen .....	5
Konfigurieren .....	5

## K

Konfiguration der ini – Datei.....	27
Konfiguration der ini-Datei	
Aufgaben definieren .....	27
Struktur der ini-Datei .....	27

## M

md_createviews_configbase.ini .....	15, 19
md_createviews_database.ini.....	15, 19
md_dropviews_configbase.ini.....	15, 19
md_dropviews_database.ini .....	15, 19
MS-DOS-Eingabeaufforderung .....	38
Schalter /? .....	38
Schalter /h .....	38
Schalter /helpfile.....	38
Schalter /ini.....	38

## N

Neustart .....	6
Not defined AVs .....	21
NOT_REFERENCED	
Beispiel.....	31
NULL Pointer	
Beispiel.....	30

## P

Prüfverfahren	
Generic.....	7, 13
Not defined AVs .....	21
Specific	
Blob content.....	25

---

Configuration .....	17
Disabled relations .....	18
Double user-def. Attr. (ECB) .....	19
Double user-def. Attr. (RS) .....	19
Not-NULL default AV's .....	20
Synchronize Graph .....	18
Synchronize PTS .....	17
<i>Specific</i> .....	17

---

## **R**

Referenzierte Klasse .....	32
Referenzierte Klasse. weglassen .....	32

---

## **S**

Schlüsselgruppe .....	27
<i>Specific</i> Prüfverfahren .....	17
SQL .....	3
Starten des DBAnalysers .....	4