



HOME

User Manual

DELMIA Process Engineer[®]

DBAnalyser



Foreword

This manual provides an introduction to the basic operations and functions of the DBAnalyser.

While developing these functions we have made every effort to create a clearly organized, easy-to-understand program structure.

A user-friendly interface as well as a clear menu guide will enable you to quickly learn how to operate the program and to get familiar with its functions so that you can carry out your planning tasks in a quick and reliable way.

No Liability or Guarantee

Our programs and manuals have been compiled with great care and to the best of our knowledge. They have also been tested in a production setting. However, we assume no liability and provide no guarantee that the software and related descriptions are free of error or are suitable for special purposes.

DELMIA assumes no liability for any damage that may arise from the use of this software. By using this software, the user acknowledges this exclusion from liability and shall hold DELMIA exempt from all claims.

Copyright

The information in our documents may be copied and distributed for internal purposes provided it is done free of charge and the contents are not altered or distorted.

Any other form of usage, especially the sale on CD-ROM or in any other publication in whole or in part is only permitted after prior written consent by DELMIA.

Some parts of this software are owned by Unigraphics Solutions Inc. and are copyrighted © 2010. All rights reserved.

Some parts of this software are owned by combit® GmbH and are copyrighted. Report-/Print module List and Label® Version 8.0: Copyright combit® GmbH 1991-2010.

Modifications

Moreover, DELMIA retains the right to make modifications and improvements to the product described in this manual at any time without prior notification.

DELMIA and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries, in the United States or other countries.

© 2001-2010 Dassault Systèmes - All rights reserved

Table of Contents

1. Introduction	1
1.1 How to Use this Manual	1
1.2 Documentation Conventions and Symbols	1
1.3 New Functions in DBAnalyser	2
2. Overview	3
2.1 Getting Started	3
2.1.1 Initial Start of DBAnalyser	3
3. Starting the Program	5
4. Operating Mode	7
4.1 General Information	7
4.2 Interactions with the ORACLE Database	8
4.3 Error Types	9
4.3.1 Explanation of the Error Types	9
4.3.2 Generic Test Procedures	10
4.3.3 Specific Tests	16
5. Configuration of the <i>ini</i> File	28
5.1 Concrete Task	28
5.1.1 Structure of an INI file	28
5.1.2 Defining Missing Reference	29
5.1.3 Examples	31
5.2 Additional Notes	37
5.2.1 Start the DBAnalyser in Batch Mode	37
List of Figures	39
List of Tables	40
Index	41

1. Introduction

This manual explains how to use the Process Engineer DBAnalyser for your planning purposes.

1.1 How to Use this Manual

This manual enables you to get familiar with the operation and functions of the Process Engineer. This manual briefly describes:

- DBAnalyser Functions



Note

When handling the DBAnalyser functions, please also refer to the general introduction to Process Engineer in the General Introduction Manual.



Click [General Introduction](#) to access the manual.

1.2 Documentation Conventions and Symbols

The symbols used in this manual are intended to provide you with keys to the contents in an immediately understandable manner.



This symbol is used to introduce key concepts that are covered in the sections immediately following this symbol. As a result, this symbol most frequently appears at the beginning of chapters or sections.



Note

*This symbol is used to mark notes, which provide you with additional information you need to have for further work. You will either find the Note sign at the beginning of a chapter or in a particular text passage in the chapter. Texts bearing this sign are additionally marked with **Note**. The text is always in italics.*




Caution

*This symbol indicates that the text that follows describes particular circumstances that you must avoid to avoid potential errors with the operation of the program or harm to data. You will either find the Caution sign at the beginning of a chapter or near a particular text passage in the chapter. Texts that are introduced by this sign are additionally marked with **Caution**. The text is always in italics.*

Example

This symbol marks examples which serve to illustrate a certain situation.

- 1) This symbol marks the individual operational steps involved in a particular operating instruction. Operating instructions describe operational steps, for example, how to open a menu or execute a function.
- This symbol marks listed subjects. The symbol for listed subjects can be either used to structure a continuous text or to list main subject keywords.
- This symbol marks list inside a bulleted or numbered list.
-  This symbol marks cross reference information that is available in another manual.

1.3 New Functions in DBAnalyser

No new functionality has been added for this release.

2. Overview

In a multiserver environment it is possible for hardware and network failures, for example, to lead to inconsistent data which is not immediately recognizable as such on the user interface.

Such database errors can be detected in advance, displayed, and rectified with the DBAnalyser.

The DBAnalyser searches the PE database for inconsistent data. If such data are found, they can be documented and the inconsistency can be rectified.

Running this program on a regular basis in a productive environment ensures an error-free database.

2.1 Getting Started

Experience has shown that not all possible error types occur in every environment. It is therefore recommendable to adjust the DBAnalyser to the environment in which it is used after a start-up phase.

The DBAnalyser can be used only by an administrator who has access to the server.



Note

For some configurations knowledge of SQL command language is required.

SQL is an ANSI (American National Standards Institute) standard computer language for access to and manipulation of database systems. SQL statements are used to retrieve data from a database and to update it. SQL may be used for instance in the following and many other databases: ORACLE, MS SQL Server, DB2, Informix, MS Access, Sybase.

If you do not have knowledge of SQL, you should not edit "SQL Queries".

2.1.1 Initial Start of DBAnalyser

The executable file **dbanalyser.exe** and the **.ini** configuration files can be found in the directory **PPRServer\program\bin**.

All actions of the DBAnalyser are logged continuously in the file **DBAnalyser.log**. Not only the actions of the DBAnalyser, but also, all detected errors are logged in the log file.

Before using the DBAnalyser for the first time you should configure it to:

☒ Check

- Activate all tests

☐ Handle

- Deactivate all corrections

The DBAnalyser can be configured using the **md_fullcheck_database.ini** file.

1. Open the file **md_fullcheck_database.ini** using a text editor. It contains the following :

```
[COMMON_DATA]
DataBaseType=DATABASE
```

```
Interactive=true
CreateEasyViews=false
DropEasyViews=false

[INDEX_STATISTICS]
Check=true
Handle=false

[FAILED_POINTER]
Check=true
Handle=false

[NULL_POINTER]
Check=true
Handle=false
.
.
.
```

2. Set **Check=true** and **Handle=false** for all entries.

By configuring the DBAnalyser in this way, all detected errors are displayed. This error list is saved in the DBAnalyser.log file.

It is important to test the detected errors. Some error types require precise knowledge of the DELMIA Process Engineer data structure as well as a thorough examination.

If you are uncertain how to correct particular error types, send the log file to DELMIA Support. The log file can be examined together with the DELMIA support team. This can involve adjustments to the configuration file. Only then should the DBAnalyser apply the corrections to the errors.

When the configuration file has been adjusted (several test runs) so that all possible errors and their corrections are included, the DBAnalyser can be run in batch mode.

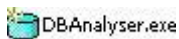
That is, upon start of DBAnalysers all tests and corrections are performed without further interaction with the operator.

3. Starting the Program

The following steps should be taken only if you are experienced in using the DBAnalyser and the data model of the DELMIA Process Engineer.

Configuration of the ini file: Configure the file **md_fullcheck_database.ini** as described in the chapter [Initial Start of DBAnalyser](#). You can find the file in the directory **PPRServer\program\bin**. When starting for the first time, all Handle should be set to false.

Terminate all PPR Server Processes: Terminate all PPR server processes.



Start the DBAnalyser: Start the program DBAnalyser.exe in the directory PPRServer\program\bin.

- A file selector opens in which you should select the **.ini** file. Several **md_<functionsname>.ini** files are offered for selection. These files can be used as a template for the first tests, but then should be replaced by a file adapted to the respective installation. From the file name you can extract the function of the individual files. But to edit the individual functions, the files must be opened using an editor. In section [Configuration of the ini File](#) editing of the *ini* file is described.

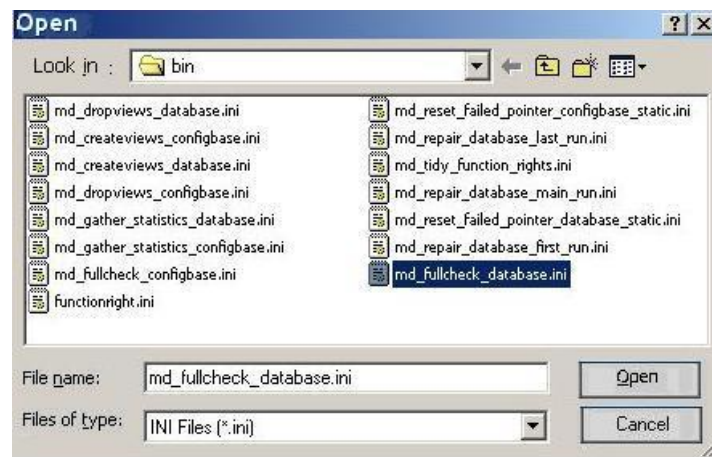


Figure 1: File Selection Window for the ini File

- After the selection is made, the **DBAnalyser** dialog opens.

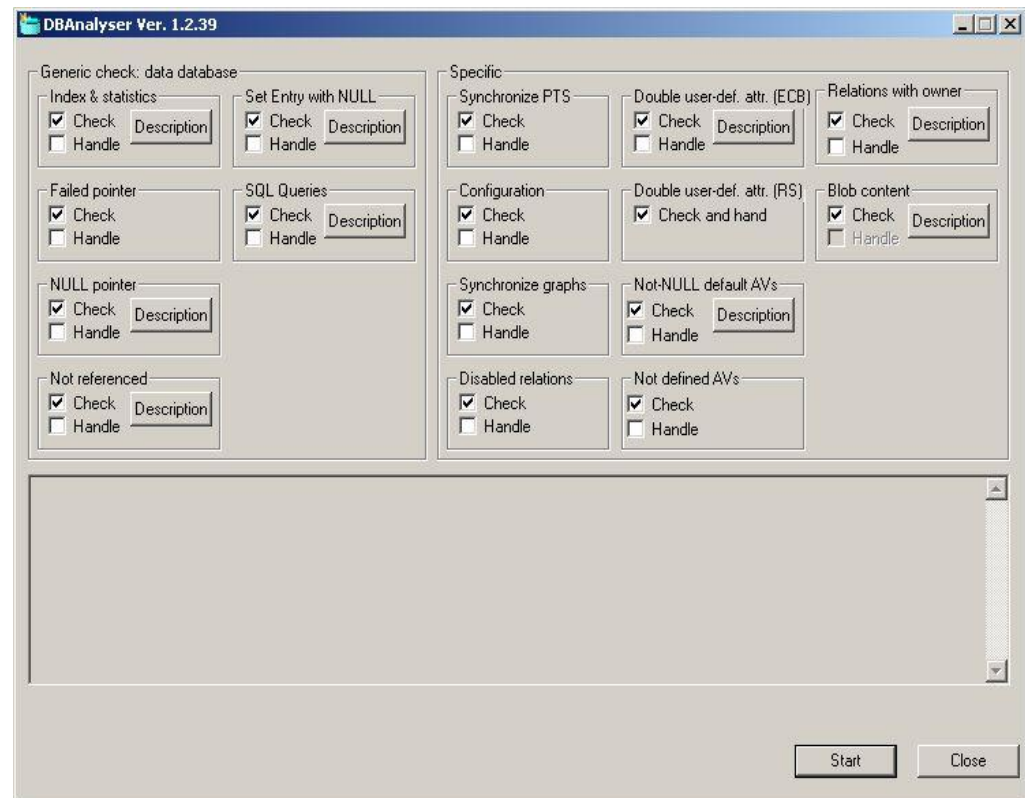


Figure 2: DBAnalyser - Dialog

PPR servers should not be started during the analysis phase.

- **Start the Test:** The test is started by clicking the **Start** button. The progress of the test is displayed in the lower part of the dialog. If errors are found during a test, a list of all errors is displayed for every test type (e.g. NULL pointer etc.).
- **Correcting Errors:** You can execute the recommended changes if, as mentioned in the introduction, the errors have been closely examined.
- **New Start:** If errors were found and corrected by the DBAnalyser, the DBAnalyser should be restarted. You must first close the DBAnalyser and start again.

All detected errors are logged in the file **DBAnalyser.log**. You need the log file even if you consult DELMIA Support.



Caution

Back up your database before starting the DBAnalyser. Improper use of the DBAnalyser can lead to data loss.

4. Operating Mode

4.1 General Information

The DBAnalyser has two different groups of test procedures:

Generic and Specific

Generic		Specific	
Index & statistics <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Set Entry with NULL <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Synchronize PTS <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Double user-def. attr.(all) <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>
Failed pointer <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	SQL Queries <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Configuration <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Double user-def. attr. (ECB) <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>
NULL pointer <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>		Synchronize graphs <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Blob content <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>
Not referenced <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>		Not defined AVs <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Unique GUID (syselem.) <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>
		Disabled relations <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>	Not-NULL default AVs <input checked="" type="checkbox"/> Check <input type="checkbox"/> Handle <input type="button" value="Description"/>

Figure 3: The Various Test Procedures

In order to find and if necessary correct, different error types, the databases of the DELMIA Manufacturing Hub are searched according to default criteria. A correction does not necessarily lead to completion of the files; this depends instead on the respective error type.

- The Generic test procedures can work with any database and are configured using the *ini* file.
- The Specific procedures cannot always be configured.

☒ Check

The checkbox **Check** enables the test of the respective error type. No Handle is executed without check, even if the control box is enabled.

☒ Handle

The checkbox **Handle** enables both the test as well as the possibility to correct the respective error type. Check must be enabled in order to be able to make corrections.

You can call up the settings for searching for errors of a certain type with the Description button. These settings are controlled by the *ini* file. *Please refer to Figure 4.*

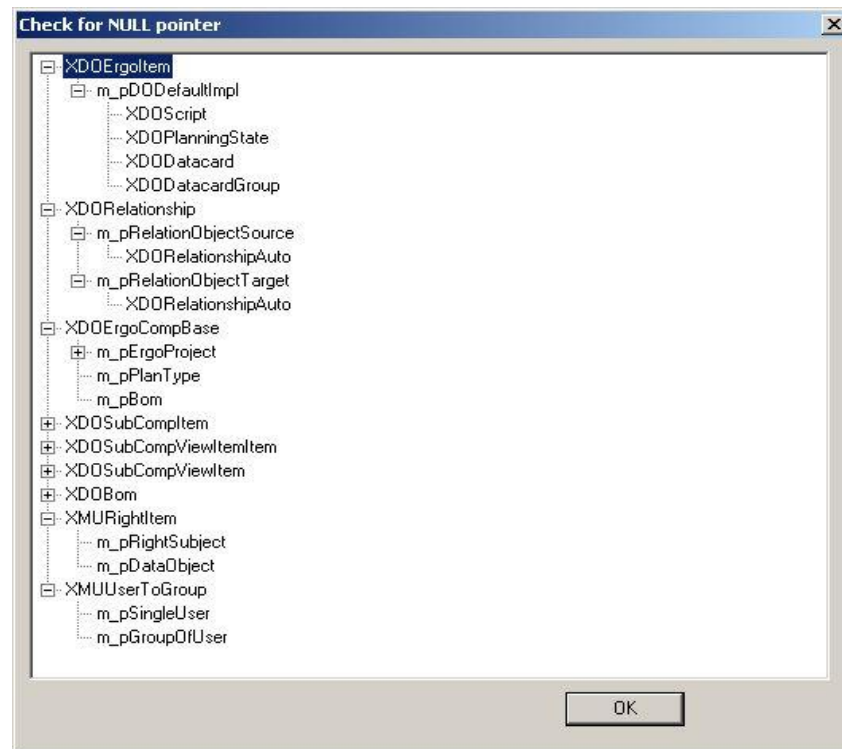


Figure 4: Dialog after Enabling the Description Button

DBAnalyser can also be run in batch mode. For details *please refer to the* [Start the DBAnalyser in Batch Mode](#).

4.2 Interactions with the ORACLE Database

The PPRServer (in other contexts also referred to as IPDServer) communicates with an ORACLE database, in which all data is saved permanently. The communication is performed via a POET interface.

The interface converts permanent class and attribute names that are independent of the version, into version related ORACLE table and column names. Unfortunately, the ORACLE names are quite cryptic and hard to memorize.

Sometimes it is very helpful to send ORACLE queries directly, to receive certain data from the database. SQL is a lot stronger than the OQL language used by POET. Additionally, there are other rules that must be observed to receive, for instance, all XDOErgoCompBase objects in which an m_pErgoProject attribute has a certain value.

To simplify this communication by the use of SQL, DBAnalyser has been enhanced with a function - the EasyViews.

EasyViews

For each POET class a view is additionally created that returns all instances of the class and has the same name as the class. The individual columns in this view have the same names as the corresponding POET attributes.

Additionally, each object contains two additional attributes, which show the POET ID, these are:

- The **OID** (unique number throughout database)
- The **CID** (class ID)

Once EasyViews have been created they may be used by the DBAnalyser as well as by the ORACLE client program (e.g. SQLPlus).

Only after EasyViews have been created, queries of the kind shown in the following example are possible:

As the result of a query all XDOErgoCompBase objects are to be returned which belong to the project “My first project”, but which are not used in a bill of materials.

Example

Example

```
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);
```

Generating EasyViews

There are two possibilities.

- The [COMMON_DATA] section is extended by one line (*Please refer to the [Configuration of the ini File](#)*):

```
CreateEasyViews=true
```

- Or start DBAnalyser with the file [md_createviews_database.ini](#) (or [md_createviews_configbase.ini](#)). Both files have been copied by the setup routine to the directory **PPRServer/program/bin**. DBAnalyser needs about 2 to 4 minutes to create the EasyViews (this depends on the computer hardware, not on the size of the database).

Deleting EasyViews

For this, too, there are two possibilities.

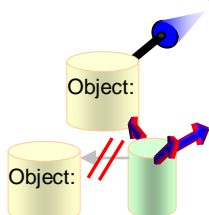
- As for creating, extend the [COMMON_DATA] section by one line (*Please refer to the [Configuration of the ini File](#)*):

```
CreateEasyViews=false
```

- Or start DBAnalyser with the file [md_dropviews_database.ini](#) (or [md_dropviews_configbase.ini](#)). These files are likewise found in the directory **PPRServer/program/bin**. DBAnalyser needs less than one minute to delete the EasyViews. This, too, depends on the hardware used.

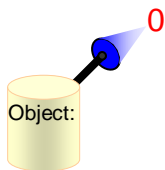
4.3 Error Types

4.3.1 Explanation of the Error Types



Difference between “not referenced” and “NULL Pointer” and “corrupted reference”

- If an object points to another non-existent object, this pointer is a corrupted reference. Thus if an XDOErgoCompBase object does not point to an existing plantype, it has a corrupted reference.



- If an object is not referenced by other objects, i.e. if no pointer *points to* the object, it is NOT_REFERENCED.
- If an object points to the figure 0 in a table (not "NULL"), it has a "NULL Pointer".

Corrupt References

Whenever a database object is deleted, all references which point to the object are set to NULL. The object is also removed from all sets. Errors lead to corrupt references.

Missing References

Some objects are complete and ready for use only if they are linked to other certain objects.

Examples of Missing References

- XDOErgoCompBase always needs one pointer to the project and one pointer to the plantype.
- XDORelationship always needs two pointers; one to the source object and one to the target object.
- XDOErgoltem needs (almost always; exceptions must be specified in the ini file) one pointer to the object to which the XDOErgoltem is attached.

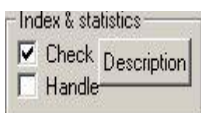
Further cases are more likely to be specific to a release or customer.

All error types are described in the following. You need the key group for configuration.

4.3.2 Generic Test Procedures

The *Generic* test procedures can work with any database and are configured using the *ini* file. With this you can adjust the *ini* file successively to your DPE - installation. There are six *Generic* test procedures. The *Generic* test procedures allow to define the database (DataBaseType=CONFIGBASE or DataBaseType=DATABASE).

4.3.2.1 Index and Statistics



(Key group [INDEX_STATISTICS]);

Checks:

- All required indices are actually existing
- Time at which the statistics for the existing indices were computed

Statistics should be queried regularly by a database administrator and missing indices should then be generated afresh. These tasks can only be performed by a database administrator.

Remedy: Creating statistics can be handled by DBAnalyser; click button **Gather Stats** in the result window as shown below.

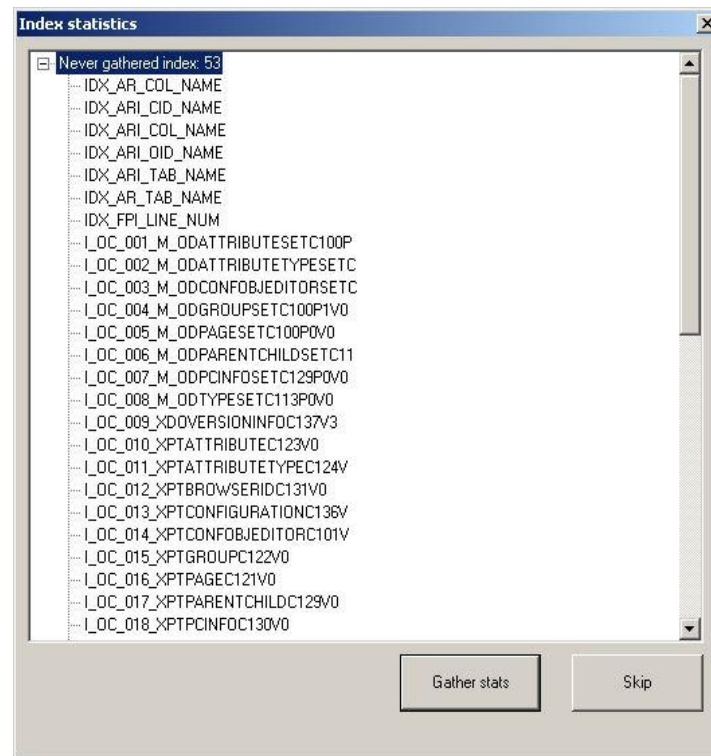


Figure 5: Index Statistics

The following command is executed for each index:

`DBMS_STATS.GATHER_SCHEMA_STATS(database_name, estimate_percent => percentage, cascade=> TRUE);`
 estimate_percent can be defined in the ini file. Default value = 5.

Example

Example:

```
[INDEX_STATISTICS_OPTIONS]
```

```
estimate_percent=8
```

4.3.2.2 Failed Pointer



(Key group [FAILED_POINTER]);

Finds all data objects with invalid pointers.

The invalid pointers are displayed in the result window, as shown below.

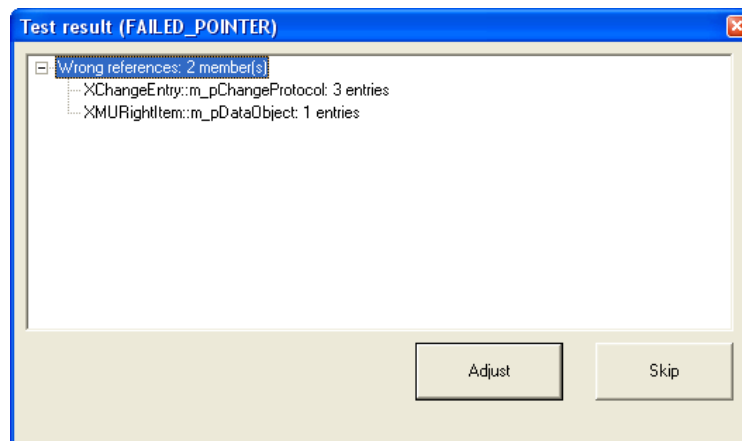


Figure 6: Failed Pointer

Remedy:

Click **Adjust** in the result window. All invalid pointers are set to null.

4.3.2.3 NULL Pointer

(Key group [NULL_POINTER]);

Finds all data objects of a given class with a NULL pointer

The NULL pointers are displayed in the result window, as shown below.

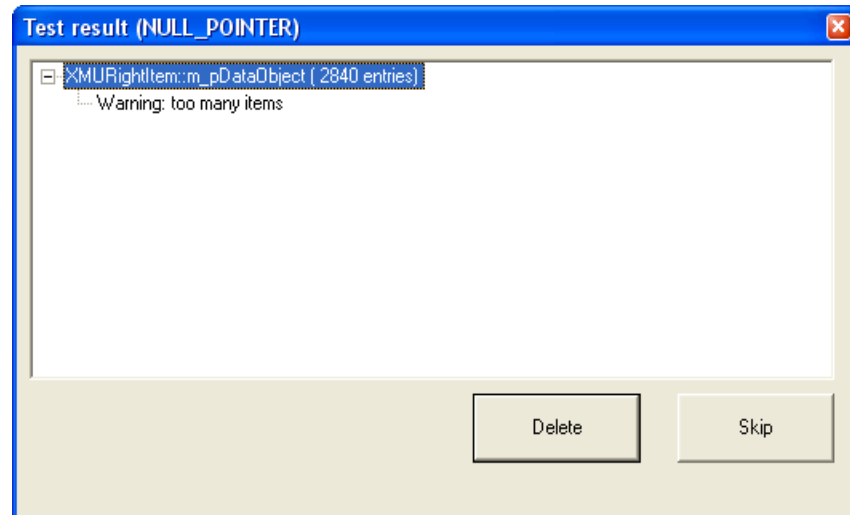
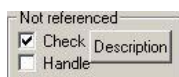


Figure 7: NULL Pointer

Remedy:

Click **Delete** in the result window. All objects found (pointer) are deleted.

4.3.2.4 Not Referenced



(Key group [IS_NOT_REFERENCED]);

Finds all data objects of a given class which are not referenced by even one object of a given class.

Such data is displayed in the result window, as shown below.

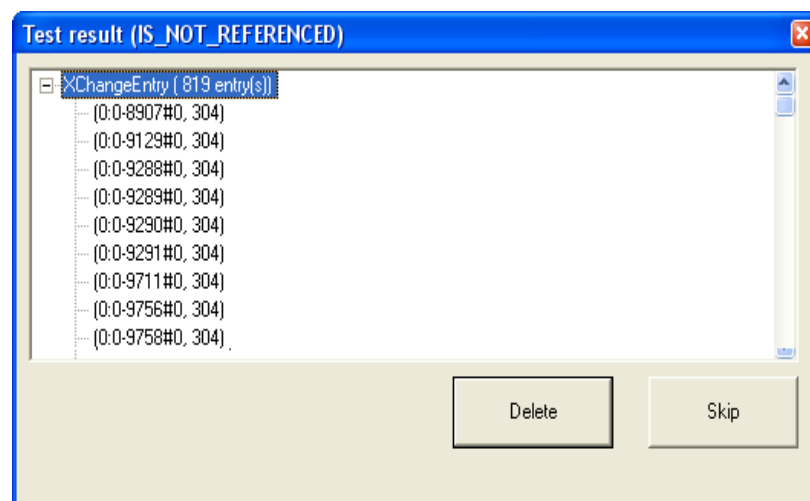
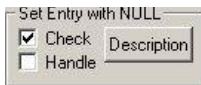


Figure 8: Not Referenced

Remedy:

Click **Delete** in the result window. All objects found are deleted.

4.3.2.5 Set Entry with NULL



(Key group [SET_ENTRY_WITH_NULL]);

Finds all data objects with sets that have corrupt references.

The corrupt references are displayed in the result window, as shown below.

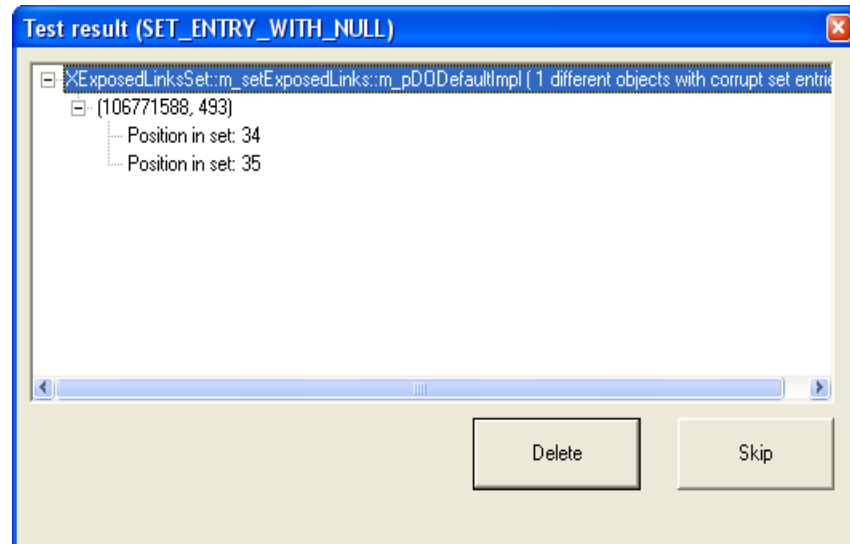


Figure 9: Set Entry with NULL

Remedy: Depending on the configuration of the .ini file, you may delete or change the found objects.

Notes on Set Entry with NULL

Some classes of the data model have sets that may contain transient objects.

These include, for example:

- XDOTimeAnalysisLineContainer/m_timeAnalysisLineSet
- XDOTimeAnalysisCoordinateContainer/m_timeAnalysisCoordinateSet
- m_setExposedLinks/XExposedLinksSet

Such sets may contain objects that have corrupt references. Since these objects are not permanent, they are found neither by FAILED_POINTER nor NULL_POINTER, nor by SQL_QUERY.

SET_ENTRY_WITH_NULL must always be configured first.

Configuring SET_ENTRY_WITH_NULL

The configuration can be explained by two examples:

Example

Example 1

ClassName=XExposedLinksSet

SetName=m_setExposedLinks

PointerToTest_=m_pDODefaultImpl

NullAllowed=false

Action=DeleteSetEntry

- For all objects of the class XExposedLinksSet, the entries are analyzed in the set m_setExposedLinks.
- With PointerToTest=m_pDODefaultImpl, every entry that points to a no longer existing object with the pointer m_pDODefaultImpl or contains a null (NullAllowed=false) will be considered erroneous.
- All erroneous entries are deleted (Action=DeleteSetEntry)

Example

Example 2

ClassName=XDOTimeAnalysisLineContainer

SetName=m_timeAnalysisLineSet

PointerToTest=m_pSubCompltem

NullAllowed=true

Action=SetToNull

- Entries are analyzed in the m_timeAnalysisLineSet for all objects of the class XDOTimeAnalysisLineContainer.
- Every entry that points to a no longer existing object with the pointer m_pSubCompltem is considered erroneous. (Zero values are allowed, since NullAllowed=true).
- The aforementioned pointer should be set to zero for all erroneous entries (Action=SetToNull).

4.3.2.6 SQL Queries



(Key group [SQL_QUERY]);

Performs SQL queries.

Recommended only to those familiar with the SQL and DELMIA data model.

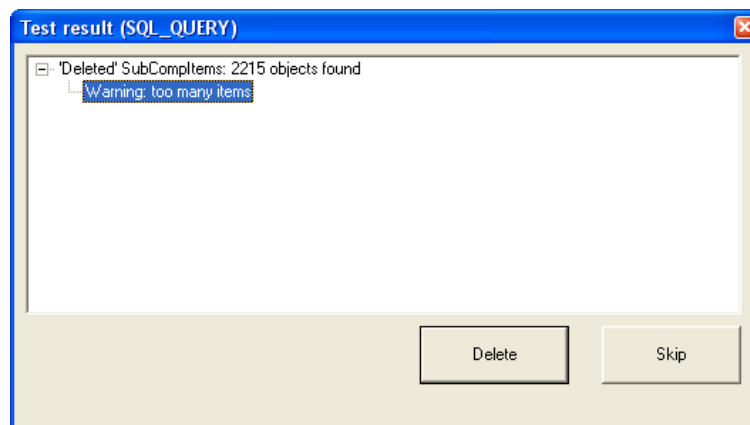


Figure 10: SQL Queries

Remedy:

Click **Delete** in the result window. All objects found are deleted.

Example

Examples can be found in the .ini files such as **md_tidy_function_rights.ini**, **md_fullcheck_database.ini** etc.

Additional Remarks Concerning SQL Queries

If you formulate SQL queries, POET objects are returned. The objects found are then displayed (you may configure if you wish to display only the object ID or its properties additionally, e.g. its name). All objects found can be deleted in one action, if required.

An SQL query can access the POET views as well as the EasyViews.

- It is recommended to access EasyViews at all times, because:
 - troubleshooting is easier (no cryptic names)
 - the same query can be used in different versions without having to change the text of the query.



Note

To create EasyViews you may use the two ini files *md_createviews_database.ini* and *md_createviews_configbase.ini*. These views can be deleted using the two ini files *md_dropviews_configbase.ini* and *md_dropviews_database.ini*.

Requirements

The SQL query can have any complexity and can use the possibilities of ORACLE SQL to the fullest. Just **one** requirement must be met: Always **two columns** must be returned. These columns must contain the OID and the CID of the objects queried.

Table 1: Configuring SQL query

Query	Description
QueryName	Short name. The name is displayed in the result window and in the log file. It should be as short as possible.
QueryDescription	Is meant for notation and contains the detailed description of the task to be performed by the query.
QueryBody	Contains the actual query. May occupy only one line (max. 2048 characters). May be closed with a semicolon.
LoadFullDescription	True or false. Indicates whether the objects found are to be loaded to the memory or not. It does not make sense to load all objects into the memory, as this will cost time and memory space.

Example

Example:

```
[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 XDOSubCompItem 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
```

4.3.3 Specific Tests

Some Specific test procedures can also be configured. However, for Specific test procedures you cannot define the database (CONFIGBASE or DATABASE). They do not need to be executed every time the DBAnalyser is started. For example, the synchronization of the plantype sets can be checked only if an import (project, PTS or configuration) has been executed or if operation failures occurred during the editing of a plantype set.

4.3.3.1 Synchronize PTS



Key group [SYNCHRONIZE_PTS];

Checks all plantype sets (PTS).

All plantype sets used in projects (slave-plantype sets) must have the same number of plantypes as the plantype set in the system library (master-plantype set).

Entries not satisfying this constraint are displayed in the result window, as shown below.

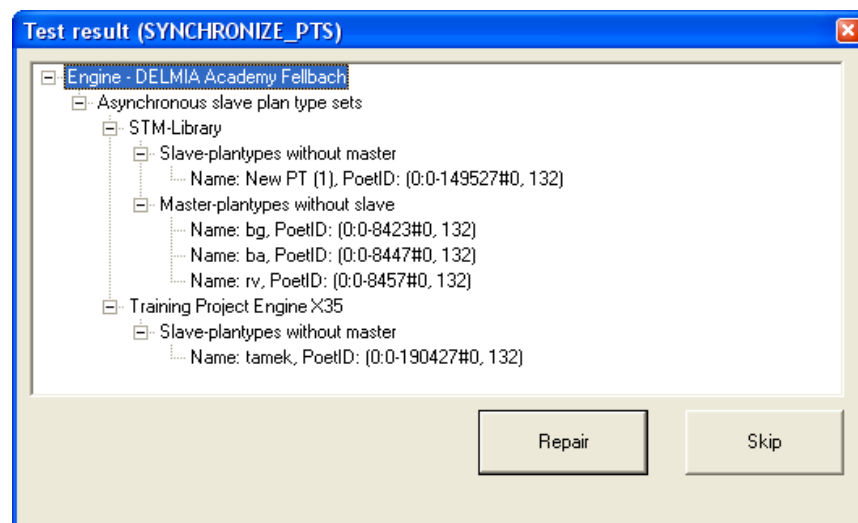


Figure 11: Synchronize PTS

Remedy:

Click **Repair** in the result window.

- All missing slave plantypes are generated.
- For every master-plantype set (XDOErgoPlanType) object there is exactly one XPtType object. Therefore, all missing XPtType objects are generated.

4.3.3.2 Configuration



Key group [CONFIGURATION];

All objects in the **CONFIGBASE** and **DATABASE** which define types and parent-child relations are checked. The following is checked:

- a) For every master-XDOErgoPlanType object, an XPtType object exists;
- b) For every XPtType object an XDOErgoPlanType object or a POET class exists;
- c) Every XPtType object points to a valid XPtType object (or 0) either with the “**m_odSuperType**” or the “**m_strBaseClassName**”;
- d) All XPtType objects have unique naes;
- e) All XPtParentChild objects are unambiguous (combination of **m_strChildListName**, **m_parent** and **m_child** must be unambiguous).

Objects not satisfying these constraints are displayed in the result window, as shown below.

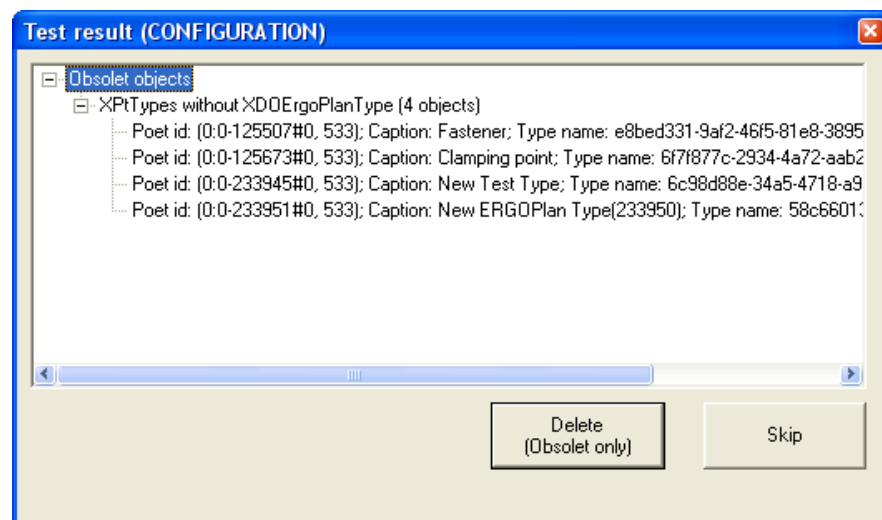


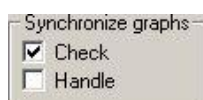
Figure 12: Configuration

Remedy:

Click **Delete(Obsolete only)** in the result window.

- All objects that are found faulty when checked against criteria (b), (d) and (e), are displayed in the directory “obsolete” and can be deleted automatically by DBAnalyser (button “Delete (Obsolete only)”).
- All others are saved to the directory “redundant” and must be repaired using the configuration tool or other DPE tools, as needed.

4.3.3.3 Synchronize Graphs



Key group [TOO_MANY_GRAPH_BOMS];

Checks whether, for every XDOGraph object exactly one XDOSubCompViewItemListPro (GraphBOM) object exists.

If an XDOGraph object has more than one GraphBOM, it is displayed in the result window, as shown:

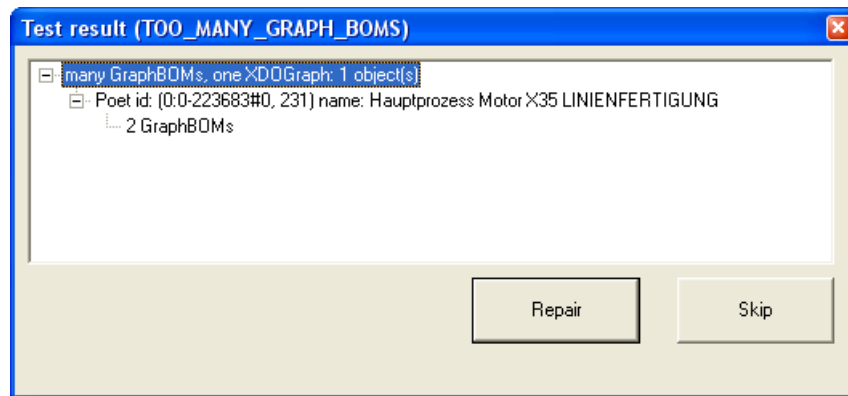


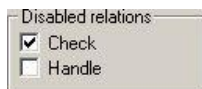
Figure 13: Synchronize Graphs

Remedy:

Click **Repair** in the result window.

- All missing XDOSubCompViewItemListPro objects are generated
- All superfluous XDOSubCompViewItemListPro objects are deleted
- All missing XDOGraph objects are generated.

4.3.3.4 Disabled Relations



(Key group [DISABLED_RELATIONS]);

Checks in the configuration if each XDORelationship object has an activated parent-child relation.

Reason: Parent-child relations can be deactivated but, deactivation does not delete the relation from the database. Thus, though the relation is no longer visible, it is still present in the database.

XDORelationships with deactivated parent-child relations are displayed in the result window as shown below.

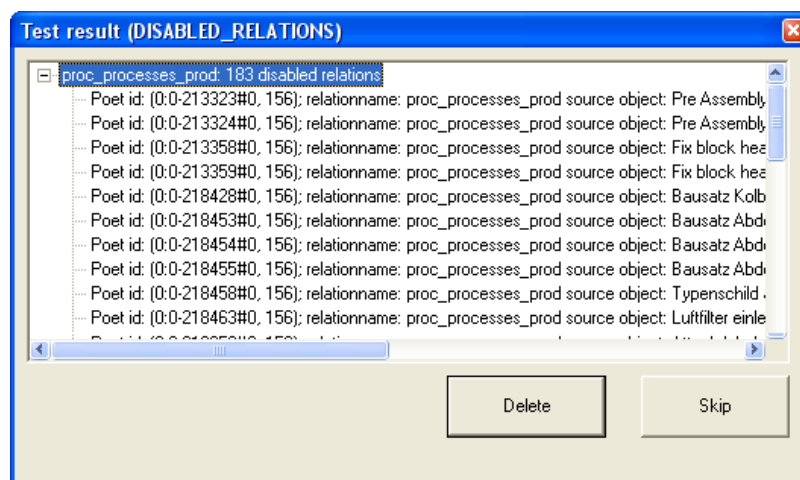


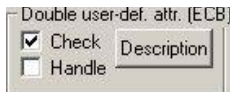
Figure 14: Disabled Relations

Remedy:

Click **Delete** in the result window. This deletes the XDORelationships with deactivated parent-child relations.

4.3.3.5 Double user-def. Attr. (ECB)

(Key group [USER_DEF_ATTR_ECB]);



Note

This search can only be started if so-called EasyViews have been created previously. To do so, you may use the two ini files [md_createviews_database.ini](#) and [md_createviews_configbase.ini](#). These views can be deleted using the two ini files [md_dropviews_configbase.ini](#) and [md_dropviews_database.ini](#).

Checks whether an XDAttributeValue with the same pendant exists in XDORgoCompBase. This is the case if user defined attributes and pre-configured attributes have been used for the same purpose (for instance: "name" and "m_name"). If such attributes exist, then they are displayed in the result window as shown below.

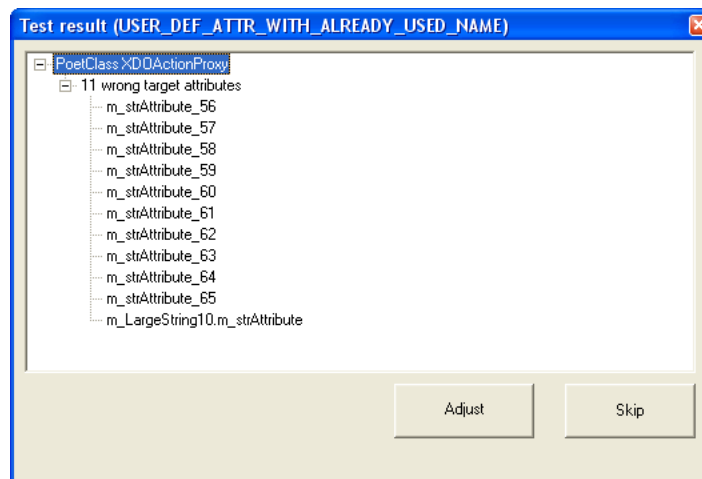


Figure 15: Double user defined attributes

Remedy:

Click **Adjust** in the result window. This deletes all such XDAttributeValues found.

You may selectively exclude individual attributes from the check (*Please refer to the [Structure of an INI file](#)*).

4.3.3.6 User Defined Attributes (RS)

(Key group [OTHER_PROBLEMS])

Checks whether the attributes 'Name', 'Shortname', 'Modificationdate' or 'pprloaderhistoricalid' are marked as "user-defined attribute".

Remedy:

Click **Adjust** in the result window. This marks the attribute as 'not user-defined'.

Checks whether the XDAttributeValue objects which correspond to the attributes mentioned above exists on every XDORelationship object.

Transfers the values to the XDORelationship object and deletes the XDAttributeValue objects.

4.3.3.7 Not-NULL default AV's



(Key group [DEFAULT_AVS])

As of version PE 5.13 SP5 and up to version PE 5.14, no values (AVs) were created in the database for special attributes of the ERGOcomponents. This affected all attributes of the type

- Integer or Double of which the value was 0 and
- String attributes of which the value was empty.

If in these versions the standard value of this attribute has been changed (unequal to 0 or not empty) and new components that use this attribute have been created, the values of the database and those of the configuration do not correspond any longer.

Example

Example

- In attribute_22 the standard value was an empty string, i.e. no entry.
- All objects that use this attribute and in which the value of the attribute was not changed explicitly have no entry in attribute_22.
- The standard value for attribute_22 has been changed to "department x", that is, it is no longer an empty string.
- All objects that are created after the change and that use this attribute, have the entry "department x" in attribute_22 as a value.

All though there is a standard value for attribute_22 in the configuration, there now exist objects in the database that have different values.

With the help of Key group [Not-Null default_AVs] it is possible to adjust the values in the database to those of the configuration. Which attributes are to be taken into consideration for a correction are specified in the *-ini* file. Such a correction is especially useful for attributes that are added.

Remedy:

Click **Adjust** in the result window. All entries of the attributes specified in the *-ini* file in which there is no value available in the database are overwritten with the standard value of the configuration.

Example

Example

```
[DEFAULT_AVS_AV_DATA_1]
Plantypeset=Default PTS
Plantype=test process
NumberOfAttributes=2
Item_1=OldFunction_1
Item_2= DummyFunction_2
```

In this case the attributes **attr_1** and **attr_2** are checked for components that build on the plantype 'test process' in the plantype set 'Default PTS'.

- By using the checkbox 'Check' one can receive information as to which AVs of the various ERGOcomponents are affected.
- If 'Handle' is activated the AVs are set to the standard value of the configuration.

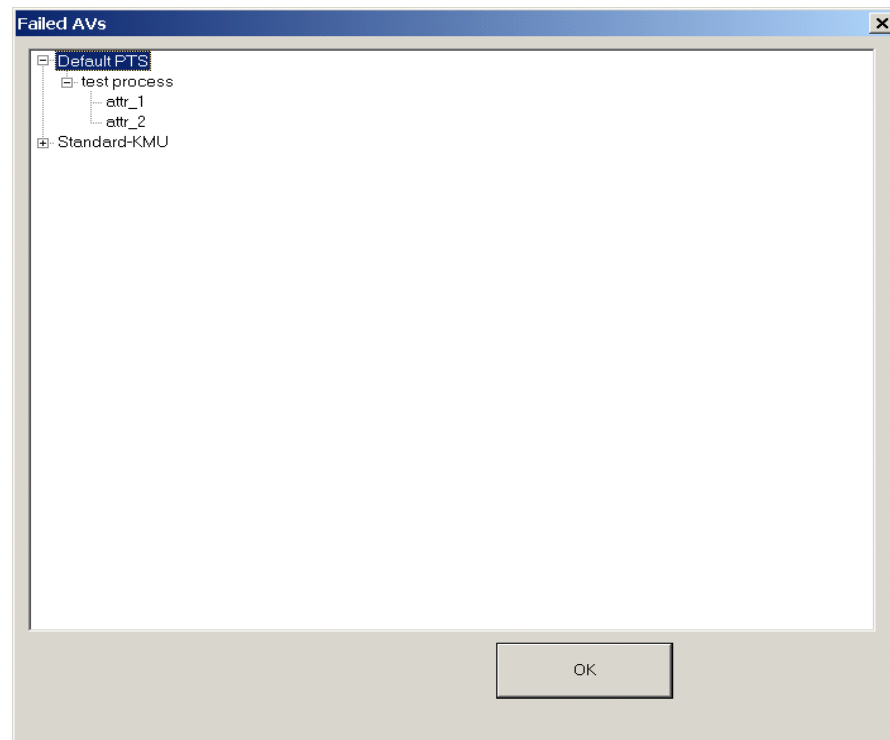


Figure 16: Failed AVS

4.3.3.8 Not defined AVS



(Key group [NOT_DEFINED_AVS])

Searches the database for user-defined attributes that are no longer used.

In DELMIA Process Engineer® it is possible to define user-defined attributes for certain ergocomponent types. Such attributes make the DELMIA Process Engineer® more flexible on one hand, but slower and more demanding in terms of storage space, on the other. Unnecessary user-defined attributes can be deleted, and are therefore no longer visible in the Configuration Manager. But the corresponding database object continues to exist even if an attribute is deleted, and thus no storage space is gained. Even more confusing is the fact that, if a new attribute with the same name as the deleted attribute is created, the old values appear again.

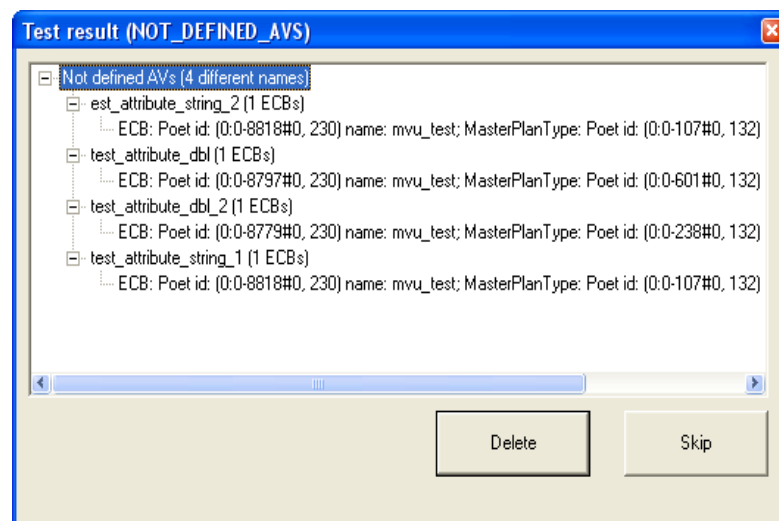
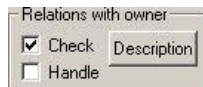


Figure 17: Not Defined AVS

Remedy:

Click **Delete** in the result window. This deletes, or, to be more specific, cleans the database of all user-defined attributes that no longer appear in the configuration or in the plantype set.

4.3.3.9 Relations with Owner



(Key group [RELATIONSHIP_OWNER_OPTIONS]);

As of version PE5.16 all relations have an attribute m_pOwner. Which object this attribute will point to is defined on the corresponding XPtParentChild object in the Configuration Manager. There are 5 variants from which one can choose:

- **No Owner** (m_pOwner = 0)
- **Source Owner** (m_pOwner = m_pRelationObjectSource);
- **Target Owner** (m_pOwner = m_pRelationObjectTarget);
- **Explicit Owner** (m_pOwner points to every dodefaultimpl object);
- **Common Parent Owner** (m_pOwner points to the ERGOcomponents, to which the bill of materials entries as well as the source and target object belong -> common parent).

Changes to the configuration or server errors can lead to m_pOwner not pointing to the object to which it is supposed to point. Relations with owner searches through the configuration for objects of which the attribute m_pOwner points to incorrect objects.

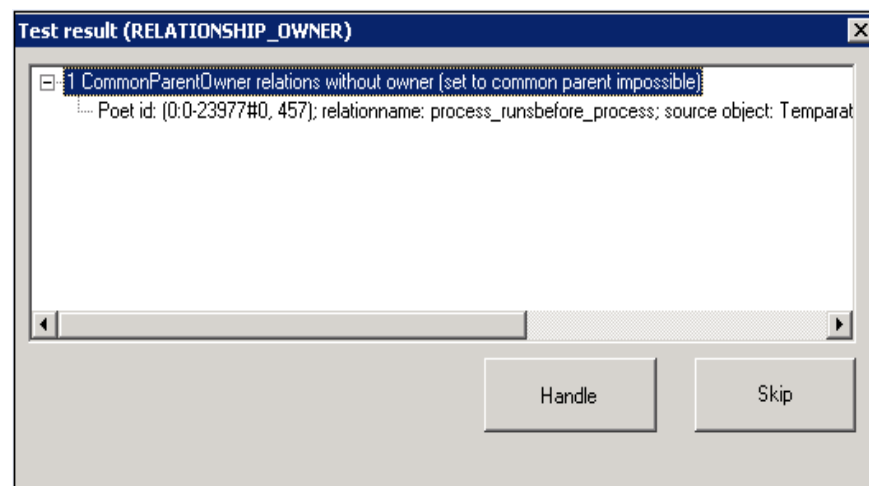


Figure 18: Relations with owner

Remedy:**Click**

- For NoOwner relationship: the pointer is set to null.
- For SourceOwner relationship: the pointer is set to source object.
- For TargetOwner relationship: the pointer is set to target object.

- For ExplicitOwner relationship: Whether the pointer is ignored, whether it points to the project, or whether it is deleted is defined in the ini file.
- Two cases are distinguished from each other for the CommonParentOwner relationship:
 - **1st Owner is null:** Whether the pointer is ignored, whether it points to the project, or whether it is deleted is defined in the ini file.
 - **2nd Owner is not null and is not a common parent:** Whether the pointer is ignored, whether it points to the project or the only parent, or whether it is deleted is defined in the ini file.



Note

If changes were made to the configuration of the attribute m_pOwner, the DBAnalyser should always be started and the relations should be checked.

How is SET_ENTRY_WITH_NULL configured?

First set whether the function group SET_ENTRY_WITH_NULL is to be started.

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

Then define whether all or only selected relations are to be checked.

In the last step you must determine what is to happen to erroneous relations if they cannot be repaired.

An example from the ini file:

```
[RELATIONSHIP_OWNER_OPTIONS]
RelationTypes=*
ActionByExplicitOwner=SetToProject
ActionByCommonParentWithoutOwner=SetToUniqueCommonParent
ActionByCommonParentWithWrongOwner=SetToUniqueCommonParent
```

The individual items:

- 1) **RelationTypes:** either * (i.e. all relations are checked) or individual relation types that are separated by semicolons, e.g.:
RelationTypes=process_runsbefore_process;nodes
- 2) **ActionByExplicitOwner:** Defines what is to happen to every Explicit Owner relation that has an empty m_pOwner attribute.
The following options are available:
 - **Ignore:** The relation is not modified
 - **SetToProject:** m_pOwner points to the project to which the source object (or its parent, if the source itself is not an ERGOcomponent) belongs.
 - **Delete:** Relation is deleted
- 3) **ActionByCommonParentWithoutOwner:** Defines what happens to every CommonParentOwner relation for which the m_pOwner = 0

4) **ActionByCommonParentWithWrongOwner**: Defines what happens to every CommonParentOwner relation that points with m_pOwner to an object that is not the common parent of the source and target (i.e. either the source or target object or both are missing in the owner bill of materials). The following options are available:

- **SetToUniqueCommonParent**: m_pOwner points to the ergocomponents in which bills of materials as well as source and target objects appear. In order for the parent to be recognized as "UNIQUE", there may only but one such object. It is possible that the UniqueCommonParent does not even exist.
- **Ignore**: The relation is not modified
- **SetToProject**: m_pOwner points to the project to which the source object (or its parent, if the source itself is not an ERGOcomponent) belongs.
- **Delete**: Relation is deleted.

All relations that have problems are sorted according to types. A directory is created and the corresponding relations are displayed for every type.

Up to eight directories are possible:

- **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: see ActionByCommonParentWithoutOwner)
- **CommonParentOwner**: relations with wrong owner (action: see ActionByCommonParentWithWrongOwner)
- **CommonParentOwner**: relations without owner (set to common parent impossible)
- **CommonParentOwner**: relations with wrong owner (set to common parent impossible)

The last two directories can only exist if

ActionByCommonParentWithoutOwner = SetToUniqueCommonParent and

ActionByCommonParentWithWrongOwner = SetToUniqueCommonParent .

- It is possible to make repairs in all other cases.

The differences between **CommonParentWithoutOwner** and **CommonParentWithWrongOwner** arise for the following reasons:

A **CommonParentWithoutOwner** error occurs if a NoOwner relation type is reconfigured into a CommonParentOwner. In this case information is lost if the relation is simply deleted. Therefore it is sensible to consider these relations individually and to set them manually (or with the help of a script) to the correct pointer.

A **CommonParentWithWrongOwner** error occurs if one or several invalid relations are found. The corresponding relations should then simply be deleted.

4.3.3.10 Blob Content



(Key group [BLOB_CONTENT]);

Searches blob's in which scripts have been saved.

If one script function is replaced by another in course of a version update, it is very time consuming to find all scripts affected. With the aid of the key group [BLOB_CONTENT] you may locate all entries affected.

- **Remedy:** Does not overwrite or delete ANY entries. You will only receive an information in which script and at what position the entry in question is used. Scripts often depend on the plantype set used and on the configuration. Therefore, each entry should be checked individually before it is replaced.

For each pair of ClassName and AttributeName a directory is created in which all entries found are saved (sorted by ItemName). For each occurrence information on the position in the blob is saved as well (line + column).

All entries found are listed in the result window and the log file.

Structure of ini file

Table 2: INI File Structure

Item	Description
ClassName	Defines which POET class is analyzed (XDOScript).
AttributeName	Defines which attribute is analyzed.
NumberOfItems	Number of terms searched for.
Item_/_	Defines the name of the I-th term.
Comment_/_	A comment text. Is output for each term found.
CaseSensitive_/_	True or false; defines whether uppercase/lowercase spelling is to be considered while searching.
WholeWordOnly_/_	True or false. Defines whether only complete words are to be considered while searching.

Example

General Example

```
[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

```

Example**Example**

```

[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_source
NumberOfItems=1
Item_1=proc_processes_prod
Comment_1=replace by 'proc_processes_prod_reverse'
CaseSensitive_1=true
WholeWordOnly_1=false
Item_2=String 2
Comment_2=performance problem?
CaseSensitive_2=false
WholeWordOnly_2=false

```

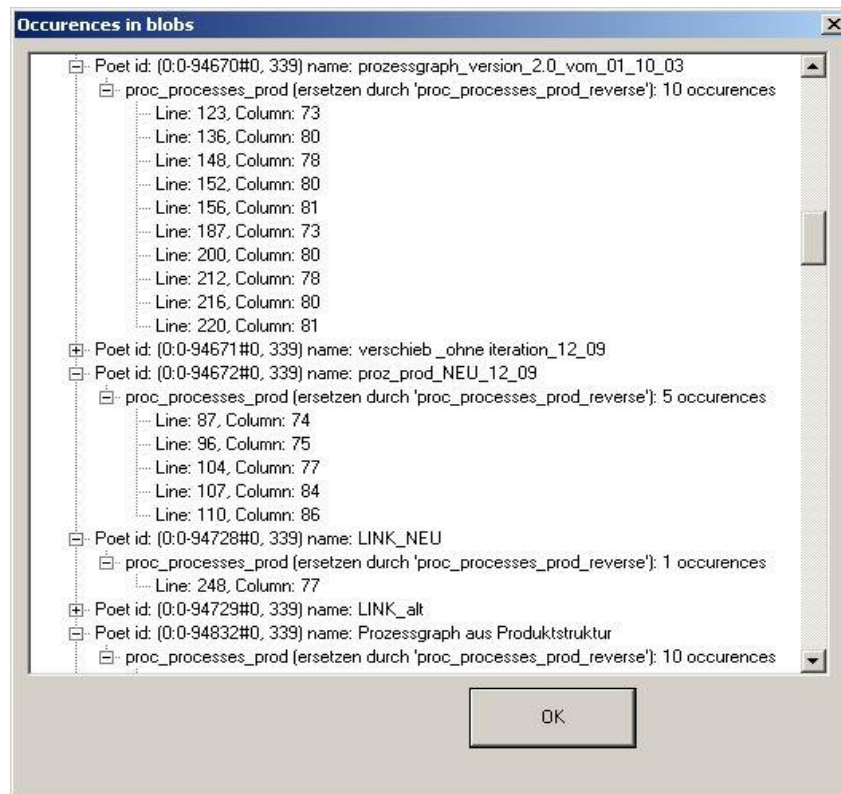


Figure 19: Occurence in Blobs

5. Configuration of the *ini* File

The DBAnalyser can be configured using an .ini file.

You can create various ini files, depending on what you want to check with the DBAnalyser.

In general one can distinguish between two types of ini files:

- Ini - files which check the **configuration** and
- Ini - files which check the **database**

The setup is identical in both ini files. They differ only in the first section, the [COMMON_DATA].

5.1 Concrete Task

The concrete task is defined by creating an INI file. This INI file must contain all details.

5.1.1 Structure of an INI file

[COMMON_DATA]

The first section contains the [COMMON_DATA]. Here you can specify the type of the database to be checked.

The key is **DataBaseType**, with the two possible values

- DATABASE and
- CONFIGBASE

In the third line the switch **Interactive**=true or false defines whether the dialog for DBAnalyser is to be displayed (=true) or not (=false) and if user input is to be made.

When (=false) is selected, only the dialog is displayed, changes cannot be made however. This option is sensible only if the DBAnalyser is run in batch mode or overnight, since after the test all errors are rectified without a query - or faulty objects are deleted.

Creating the Key Groups

The key groups are created in the next sections.

The following key groups can be entered as the error type:

[INDEX_STATISTICS]

[FAILED_POINTER]

[NULL_POINTER]

[IS_NOT_REFERENCED]

[SQL_QUERY]

[SYNCHRONIZE_PTS]

[CONFIGURATION]

[TOO_MANY_GRAPH_BOMS]

[DISABLED_RELATIONS]
 [USER_DEF_ATTR_ECB]
 [OTHER_PROBLEMS]
 [DEFAULT_AVIS]
 [NOT_DEFINED_AVIS]
 [RELATIONSHIP_OWNER_OPTIONS]
 [BLOB_CONTENT]

How the key group is to be treated is set in the in the next two lines.



Check = true

- Means that the specified error type is being searched for. If 'false' is entered, this test is skipped.

Handle = true

- Means that for the specified error type the opportunity to correct the respective error type is offered in addition to the test. For example, in the case of error type 1, all faulty references are set to NULL, in the case of error type 2 all objects with faulty references are deleted and in the case of error type 5, all plantype sets are synchronized.

normal
mode

If 'false' is entered, the errors cannot be corrected (the button **Delete** is disabled  ).

Check must be enabled in order to be able to make corrections.

- Press **Skip** button to move to the next test without making corrections.

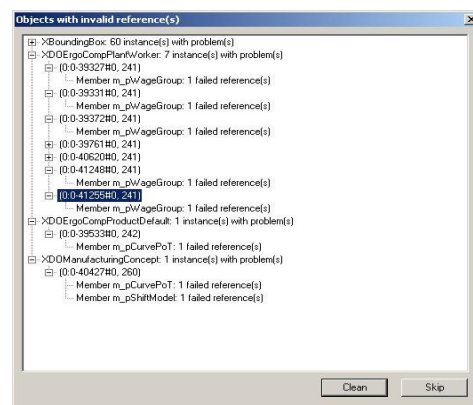


Figure 20: Dialog if Errors have been found

Batch
mode

Batch Mode

All errors found are immediately corrected in batch mode if *Handle* = true.

5.1.2 Defining Missing Reference

Example of a Single Definition:

- All instances of the class "**ClassToTest**" and the classes derived from them for which a NULL – value is in the member variable "**m_Member1**" are searched for.
- Ignore all instances of the classes "**ClassException1_1**", "**ClassException1_2**", ..., "**ClassException1_L**".

- - Repeat the search for the member variable "***m_Member2***".
- - Skip all instances of the classes "***ClassException2_1***", "***ClassException2_2***", ..., "***ClassException2_M***".
- - Repeat the search for the member variable "***m_MemberK***".
- - Skip all instances of the classes "***ClassExceptionN_1***", "***ClassExceptionN_2***", ..., "***ClassExceptionK_N***".
- The definition is displayed as follows:

[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***

Then for the next class a section with the name

[NULL_POINTER_POET_CLASS_2] is created, etc...

If the sequence is interrupted, i.e. ...POET_CLASS_1 is followed by ...POET_CLASS_3, the DBAnalyser ignores this entry and searches for the next valid key, e.g. **[IS_NOT_REFERENCED_POET_CLASS_1]**.

The names are the same both ini – files with regard to the [configuration](#) and [database](#).

The files are different only in the second line: `DataBaseType = CONFIGBASE` or `DataBaseType = DATABASE` (and at the classes and attribute name).

5.1.3 Examples

5.1.3.1 Example of NULL pointer

Several classes are derived from the class `XDOErgoltem`. Now all instances of the class "`XDOErgoltem`", and the classes derived from them for which a NULL – value is in the pointer "`m_pDODefaultImpl`" are searched for.

Which classes should be examined?

- `XDOErgoltem`

How many pointers are searched for?

- 1

Which pointer is this?

- `m_pDODefaultImpl`

How many classes should be ignored in the test? (the number must always be specified, even if no class is excluded.) The number must be zero;

`NumberOfExceptionClassesForPointer_1=0`)

- 4

If the number of ignored classes is greater than zero, which classes should be ignored?.

- `XDOScript`
`XDOPlanningState`
`XDODatacard`
`XDODatacardGroup`

In the ini file it looks like this:

```
[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
```

If further pointers are to be considered, they must be subsequently listed, e.g. `PointerToTest_2=m_XY`.

5.1.3.2 Examples of NOT_REFERENCED

1

`XDOBlob` needs *one* pointer to the object **XBlob**. Without `XDOBlob` and the pointer, **XBlob** objects can not be read, and they take up unnecessary disk space.

Which classes should be examined?

- `XBlob`

How many classes reference this class?

- 1

Which classes are they?

- XDOBlob

Which pointer is it?

- m_realBlob

In the ini file it looks like this:

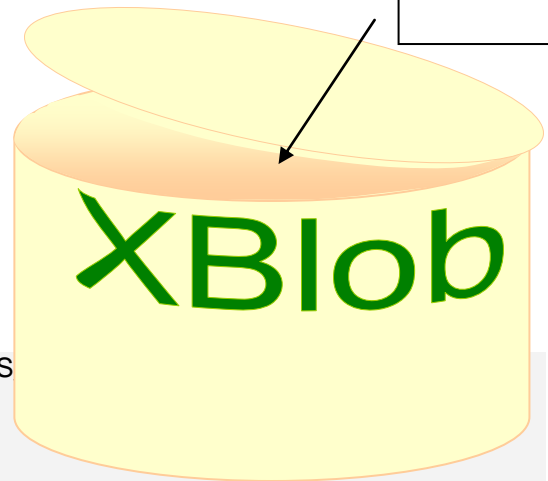
```
[IS_NOT_REFERENCED_POET_CLASS
```

```
ClassName=XBlob
```

```
NumberOfReferencingClasses=1
```

```
ReferencingClass_1=XDOBlob
```

```
ReferencingPointer_1=m_realBlob
```



2

The class *XDOAttributeValueSet* (the class of the user-defined attribute) is referenced by several classes. An *XDOAttributeValueSet* must be referenced by another class. If, for example, a *XDOErgoCompBase* - object is deleted, the corresponding *XDOAttributeValueSet* is also deleted.

Which class should be examined?

- XDOAttributeValueSet

How many classes reference this class?

- 3

Which classes are they?

- XDOErgoCompBase
XDORelationship
XDOCalcResult

Which pointers are used?

- m_pAttributeValueSet
m_pAttributeValueSet
m_pAttributeValueSet



Caution

*In this example, one can see the pure editing of the ini file as well as further special features. If a referenced class is excluded or forgotten, the DBAnalyser interprets all references to this class as errors, and deletes the objects of the class *XDOAttributeValueSet*.*

In the *ini* file it looks like this:

```
[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
```

Example of an INI file:

```
[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 XDOPBlob referenziert werden
QueryBody=SELECT OID, CID FROM XBlob WHERE OID IN (SELECT OID FROM
XBlob MINUS SELECT m_realBlob FROM XDOPBlob);
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
```

Example of DataBaseType = CONFIGBASE

```
[COMMON_DATA]
DataBaseType=CONFIGBASE
Interactive=true

[FAILED_POINTER]
Check=true
Handle=true

[NULL_POINTER]
Check=true
Handle=false

[IS_NOT_REFERENCED]
Check=true
Handle=false

[IS_NOT_IN_A_SET]
Check=true
Handle=false

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

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

[PROJECT_OF_ECB_AND_PT]
Check=true
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
```

5.2 Additional Notes

5.2.1 Start the DBAnalyser in Batch Mode

If you need a template for editing the ini file, open the prompt (The name of the MS-DOS prompt has as of Windows 2000 been changed to Command Prompt and it can be found in the *Accessories* menu).

- As shown in [Figure 21](#), you can find out what parameters are available in the director of the DBAnalyser with the **/h** or **/?** switch.

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

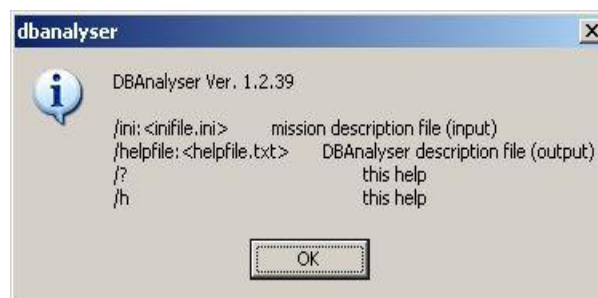


Figure 21: Prompt

- With **ini: <filename.ini>** you can start the DBAnalyser with the specified ini file (filename.ini).
- With **helpfile: <filename.txt>** you can generate a text file in the directory *...program/bin*, with the previously entered name (*filename*); the file can be used as a template, and it could contain additional descriptions of the respective error types.

```
DELMIA_R16\PPRServer\program\bin>dbanalyser /helpfile:dbanalyser_help.txt
```


**Note**

An executable file named DBAnalyser_Static.exe is included with the Process Engineer installation. This program is used by DBAssistant when upgrading a database, and is not intended for direct use by end-users.

List of Figures

Figure 1: File Selection Window for the ini File.....	5
Figure 2: DBAnalyser - Dialog.....	6
Figure 3: The Various Test Procedures	7
Figure 4: Dialog after Enabling the Description Button	8
Figure 5: Index Statistics.....	11
Figure 6: Failed Pointer.....	11
Figure 7: NULL Pointer	12
Figure 8: Not Referenced.....	12
Figure 9: Set Entry with NULL.....	13
Figure 10: SQL Queries	14
Figure 11: Synchronize PTS	16
Figure 12: Configuration	17
Figure 13: Synchronize Graphs	18
Figure 14: Disabled Relations	18
Figure 15: Double user defined attributes	19
Figure 16: Failed AVS.....	21
Figure 17: Not Defined AVS.....	21
Figure 18: Relations with owner.....	22
Figure 19: Occurence in Blobs.....	27
Figure 20: Dialog if Errors have been found.....	29
Figure 21: Prompt.....	37

List of Tables

Table 1: Configuring SQL query.....	15
Table 2: INI File Structure	25

Index

B

Batch mode 29, 37
 prompt 37
 Batch Mode 4, 8
 BLOB CONTENT 25
 Button Gather stats 10

C

Check 29
 COMMON_DATA 28
 CONFIGBASE 28, 36
 Configuration 17
 Configuration of the ini file 28

D

DATABASE_POET_CLASS_1 30
 DBAnalyser.log 4, 6
 DBAnalyser.Log 3
 Delete 29
 delete button 29
 Disabled Relations 18

E

EasyViews 8, 19
 error type
 Display error 29
error types
 set entry with NULL 13
 error types
 SQL queries
 Structure of 15
 Error Types
 Configuration 17, 18
 Disabled Relations 18
 No References 13
 Not-NULL Default AV's 20
 Null Pointer 12
 SQL Queries 14
 Error Types 10
 Corrupt References 10
 Index & Statistics 11
 Error Types
 SQL Queries
 Requirements 15
 Error Types 17
 Error Types 18
Error Types 18
 Error Types 25

G

Generic 7

H

handle 29

I

ini file 15, 25, 28
 diverse 5

K

Key groups 28

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
 missing reference 29, 31
 MS-DOS prompt 37
 /? switch 38
 /h switch 37
 /helpfile switch 38
 /ini switch 37

N

Nonliability ii
 Not defined AVs 21
 Not-NULL default AV's 20

P

prompt 37

S

Set Entry with NULL 13
 ShowResult 29
 skip button 29
 Synchronize Graphs 17
 Synchronize PTS 16

U

User Defined Attributes (ECB) 19
 User defined attributes (RS) 19

X

XBlob 32
 XDOErgoltem 31

