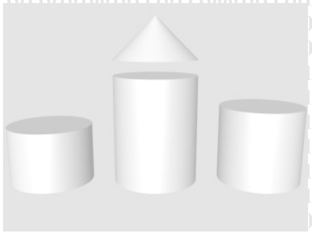
ODABA

Database Utilities



run Software-Werkstatt GmbH
Winckelmannstrasse 61
12784 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, June 2020

Contents

1 Introduction

ODABA^{NG}

ODABA^{NG} is an object-oriented database system that allows storing objects and methods as well as causalities. As an object-oriented database, ODABA^{NG} supports complex objects (user-defined data types), which are built on application relevant concepts.

ODABA^{NG} applications are characterised by a high flexibility that is achieved by supporting in addition to object (concept) hierarchy, multifarious relations between objects (master and detail relations, relations between independent objects and others). This way conditions and behaviour of objects in the real world can be represented considerably better than in relational systems.

ODABA^{NG} applications cannot only be drawn up as event-driven applications within the field of the graphical surface but also at the database level. This is one more way in which the application design is very close to the problem.

This makes ODABA^{NG} applications a favourite possibility to solve highly complex jobs as come up in administrative and knowledge areas.

Platforms

ODABA^{NG} supports windows platforms (Windows95/98/Me, Windows NT and Windows 2000) as well as UNIX platforms (Linux, Solaris).

You can build local applications or client server applications with a network of servers and clients.

Interfaces

ODABA^{NG} supports several technical interfaces:

- C++, COM as application program interface (this allows e.g. using ODABA^{NG} in VB scripts and applications)
- ODBC (for data exchange with relational databases)
- XML (as document interface as well as for data exchange)

User Interfaces

ODABA^{NG} provides special COM-Controls that easily allow building applications in Visual Basic. On the other hand ODABA^{NG} provides a special ODABA^{NG} GUI builder.

2 Common Features

Most of the ODABA tools are providing information via an option file. Option files contain information about the data source(s) and other run time parameters.

Most of utilities produce a protocol file that lists the actions performed when running the utility. Moreover, error information is written into an error file.

The following sections describe the common structure of option files for ODABA utilities and the way of using protocol and error information.

Run-time information

Most of the ODABA utilities provide two types of run time information. One is the process information listed in a protocol file, the other is error information listed in the error list.

Process information

Process information is written to a file “protocol.lst”, which is stored in a folder referenced in the TRACE-variable in the option file or in the system environment.

Process information refers to objects copied or processed in another way. Process information is also written to the console. You can suppress writing process information to console when defining the SUPRESS_ERROR variable in the utility section of the option file as:

SUPRESS_ERROR=YES

Most of the ODABA tools are providing information via an option file. Option files contain information about the data source(s) and other run time parameters.

Process information is structured as follows:

(2003/06/26 14:39:02) *message*.

Date and time are measures on the client side (when running in client/server mode). The message contains information about the action performed (as number of instances copied for a collection.

Error information

Information about errors encountered while running the process are written to the “error.lst” file, which is stored in a folder referenced in the TRACE-variable in the option file or in the system environment.

Each error message contains information about date and time, when the error was thrown, the sub-system (SDB) detecting the error, the error type (Error, Warning, Message, …), error code (0005), class and function detecting the error and a short verbal explanation of the error.

```
(2003/07/01 19:55:59) SDB  -Error  0005:
eb_RootBase :: Provide
(Open Error on data base 'l:\oma2\ode.sys'.).
```

More detailed information about the error is available with the ErrorHelp utility or function, that provides a more detailed error description (when available).

Error information can be displayed immediately when setting the SHOW_ERRORS variable in the option file or in the system environment.

SHOW_ERRORS=YES

This is intended mainly for testing purposes, since the error message box is displayed immediately, when the error occurs.

Options and parameters

When calling a utility, options and parameters might be passed to the program. Options are sort of keyword parameters preceded by a – or / character. Options may contain an option value, which may be passed in one of the following manners:

```
-I:odaba/oshell.ini
-I(odaba/oshell.ini)
-I=odaba/oshell.ini;
```

In the example above, I is the option name introduced by -. Instead of **–I**, **/I** could also be used. The option value is the path to the configuration file in this case.

There are some options, which can be passed to all utilities and which are not mentioned explicitly when describing the utility.

Quiet

In order to suppress program messaged, the quiet option might be passed (**-q**, **-Q** or **--quiet**). When passing the quiet option, system output for the program will be suppressed and no messages are written to the system output area (e.g. console). Quiet does not suppress messages created from within a program or script calling **Message()** or **Print()** functions.

Help

In order to get help information for calling a utility, you may call the utility passing the help option (**-h**, **-H** or **--help**). When passing the help option, no program massages will be displayed on system output.

Progress

Io order to activate default progress display, the progress option (**-p**, **-P** or **--progress**) might be set in the command line. This is the simplest way to activate progress display for utilities not referring to a configuration or ini-file.

In order to select a specific progress display type, you may pass one of the supported display type values as option value:

```
-p:percent
```

System output

Normally, program massages are written to system output area, which is the console for console programs and the output area control for GUI applications.

In addition, program output might be written to a protocol file, which has to be defined in the **SystemIO** option:

```
[SystemIO]
Protocol=temp/protocol.lst
```

When passing the quiet option, output will still be written to the protocol file when being defined.

Progress information

Several utilities support progress information. The way of displaying progress information depends on the settings for the progress options in the **SystemIO** section.

```
[SystemIO]
Progress=percent (dots, rotator, time, none)
```

Usually, progress information is configured in the **SystemIO** section in the configuration or ini-file passed to the utility. Some utilities, however, do not refer to a configuration file. In this case, progress options might be set as environment variables, e.g.

```
SystemIO.Progress=dots
SystemIO.Progress.Distance=100
```

The following progress display types are supported. Progress display type names are not case sensitive.

automatic	Automatic progress information display tries to select the optimal way of displaying progress information, which depends on the number of items to be copied.
percent	Progress messages in percent display the estimated percentage of work that has been done. The option will be changed automatically to rotator, when maximum number of items is not supported by the utility.
dots	The dots-progress display just writes a series of dots. Typically, dots are displayed, when a number of items defined in the **SystemIO.Progress.Modulo** option has been processed.
rotator	The rotator displays an rotating line in the system output in order to show progress speed.
time	The elapsed time will be displayed on system output area.

Progress options

In addition, some options might be set in order to alter progress information. Option names are case sensitive.

Distance	The value determines the frequency of displaying the progress message. In case of percentage, it is the distance between two percent values (e.g. displaying a message each 5 percent). For all other progress display types it is the number of items to be passed before displaying the next progress message (e.g. display a progress message each 100 records copied). Default: 10
NewLine	Display messages on the next line. When this option is set to false (NO), each progress message will be displayed on the same line deleting the content of the previous message. The option will be ignored, when the display type is **dots**.

	Default: NO
Remaining	In order to display the estimated remaining time in after the type specific progress information, the option might be set to true (YES). The option will be ignored, when the display type is **dots**. Default: NO
Trace	For displaying progress messages in the error log in addition, this option should be set to true (YES). Default: NO
Count	When the number of items to be processed cannot be determined by the program, an estimated number of items can be set as progress options. This allows configuring specific processes by passing the estimated number of items to the function. Default: 0

Configuration files

Most ODABA utilities are controlled by means of configuration or ini-files. Configuration files have a common structure, which may differ a little bit from utility to utility. The main task of a configuration file is defining data sources used in the application. Moreover, it provides system parameters as well as utility control information.

Options are read usually from the configuration file. Options not defined in the configuration file are read from the system environment. Thus, you may set (or export) environment variables as default values on your computer (e.g. for the TRACE variable or the data catalog), which might be overwritten by configuration file variable settings.

Sections

Each configuration file describes a number of standard or utility depending sections.

SYSTEM	The system section contains information for the system as data catalog location, location for error protocol, system dictionary etc. Usually (but not necessarily) the system section is defined on top of the option file.
SystemIO	System input/output options allow controlling the behaviour of writing messages to the system output area and receiving information from there. System input/output is directed to console for console applications and to designated controls in GUI applications.
utility	The utility section is a section that has the same name as the utility called. It contains utility specific runtime information. In many cases (when the utility is working with exactly one data source) the utility section acts as data source section, too.
Data sources	Usually a utility has one or more data source sections. Each data source section defines the location for a dictionary and a database. Moreover, it may contain many additional data source information.

General structure

Information for different purposes is stored in different sections of the configuration file beginning with a section name:

[SECTION_NAME]

The section name is followed by one or more section options defined as:

OPTION_NAME=string

The lines defining option names must not contain any comments. Comments can be inserted on additional lines, e.g. as

```
; this is a comment line
```

Local and client/server mode

Configuration files can be configured for local or client server mode. When running in client server mode, database paths for database and dictionary are defined as server paths (enclosed in %...%). The server maps the symbolic names to server paths when opening the database.

In principle it is possible to run a tool in a mixed environment, i.e. using local database for the SYSTEM section and client/server for the utility section. This will, however, cause problems when writing messages to the error log, since the message text cannot be resolved in this case. Hence, it is suggested to run an application either in local or in client mode.

[SYSTEM]

The system section refers to database system information. The minimum required is the DICTIONARY reference to the system dictionary. When running the application with a system dictionary stored on the server, server name and a port number have to be defined as well.

DICTIONARY

The path for the system dictionary usually refers to the **ode.sys** database in the installation path. When you receive strange error messages the reason can be an invalid path for the system database.

```
DICTIONARY=C:\ODABA\ode.sys
```

When running the system dictionary from the server the option refers to a symbolic database on the server:

```
DICTIONARY=%SYSTEM_BASE%
```

In a client server environment you may run the system dictionary also on your local machine. In this case you need to define the DICTIONARY option, only.

SERVER_URL

This option becomes necessary only when running in a client server environment (running an object server). In this case it should refer to the ODABA server name or its TCP/IP address.

```
SERVER_URL=DBServer
```

SERVER_PORT

This option becomes necessary only when running in a client server environment (object or replication server). In this case the port number must be identical with the port number passed to the server when starting it.

This variable is only required in connection with the SERVER_URL variable.

```
SERVER_PORT=6123
```

Default: 6123

TRACE

With this option the location for the error log can be defined. Often, this value is set in the system environment. It is, however, also possible to define the location in the configuration file.

```
TRACE=C:/temp
```

At the location defined in the TRACE variable an error.lst file is created that contains a detailed error log. This file should be checked in case of errors on the server side.

Default: Value for TRACE environment variable.

[SystemIO]

System input/output options allow controlling the behaviour of writing messages to the system output area and receiving information from there. System input/output is directed to console for console applications and to designated controls in GUI applications.

Protocol

Normally, program massages are written to system output area, which is the console for console programs and the output area control for GUI applications.

In addition, system output might be written to a protocol file. When defining a complete path to a protocol file, system output will be printed to this file, too.

```
Protocol=ODABA\output.txt
```

Suppress

In order to suppress program output to the system output area, the suppress option might be set to true (**YES**).

Default: **NO**

[Progress]

Several utilities support progress information. The way of displaying progress information depends on the settings for the progress options in the **SystemIO** section.

```
[SystemIO]
Progress=display_type
```

The **Progress** section is a subsection of the **SystemIO** section. Usually, progress information is configured in the **SystemIO** section defined in the configuration or ini-file passed to the utility. Within an ini-file, subsections are not supported directly and have to be defined by preceding section names:

Progress=dots

Progress.Distance=100

Some utilities, however, do not refer to a configuration file. In this case, progress options might be set as environment variables, e.g.

SystemIO.Progress=dots

SystemIO.Progress.Distance=100

The progress display types are <u>not</u> case sensitive. The following display types are supported:

<u>automatic</u>	Automatic progress information display tries to select the optimal way of displaying progress information, which depends on the number of items to be copied.
<u>percent</u>	Progress messages in percent display the estimated percentage of work that has been done. The option will be changed automatically to rotator, when maximum number of items is not supported by the utility.
<u>dots</u>	The dots-progress display just writes a series of dots. Typically, dots are displayed, when a number of items defined in the **SystemIO.Progress.Distance** option has been processed.
<u>rotator</u>	The rotator displays an rotating line in the system output in order to show progress speed.
<u>time</u>	The elapsed time will be displayed on system output area.

Distance

The value determines the frequency of displaying the progress message. In case of percentage, it is the distance between two percent values (e.g. displaying a message each 5 percent). For all other progress display types it is the number of items to be passed before displaying the next progress message (e.g. display a progress message each 100 records copied).

Distance=100

When not defining the **Distance** option, an optimal distance value is determined by the system.

Remaining

In order to display the estimated remaining time in after the type specific progress information, the option might be set to true (**YES**). The option will be ignored, when the display type is **dots**.

Default: **NO**

NewLine	Display messages on the next line. When this option is set to false (**NO**), each progress message will be displayed on the same line deleting the content of the previous message. The option will be ignored, when the display type is **dots**. Default: **NO**
Trace	For displaying progress messages in the error log in addition, this option should be set to true (**YES**). Default: **NO**
Count	When the number of items to be processed cannot be determined by the program, an estimated number of items can be set as progress options. This allows configuring specific processes by passing the estimated number of items to the function. Default: **0**
[*utility*]	The *utility* section refers to tool specific information. It starts with a section name that is identical with the utility name (the name of the executable for the utility without .exe extension). Since utility and database system usually reside in different paths, the utility section contains some standard options that should be provided in each utility section.
PROJECT_PATH	This option should refer to the project path when the project has created a context DLL. **PROJECT_PATH**=C:\ODABA\Sample
ODABA_PATH	This option should refer to the ODABA^{NG} installation path on the workstation. **ODABA_PATH**=C:\ODABA
PROGPATH	This variable should refer to the application path on the workstation. When not being defined it is set to the path from which the executable for the tool has been loaded. **PROGPATH**=C:\ODABA
MAINTENANCE_ PROCESS	Running a process as maintenance process will disable automatic time stamp settings when writing database objects. This option is set automatically for some database utilities as CopyDB or CopyResDB. **MAINTENANCE_PROCESS**=YES
	Beside these options the tool section may contain variables with specific meaning for the tool. In addition, it may contain the definition of a data source, which is typically the case for tools running with one data source, only.

[DataSource]

Data source sections may be defined explicitly or implicitly with the utility section. Each data source section is introduced by the data source name as section name. The minimum information for a data source is the dictionary and the database location.

DICTIONARY

The path for the utility dictionary usually refers to the application dictionary.

DICTIONARY=C:\ODABA\Sample\Sample.dev

When running the dictionary from the server this variable should refer to

DICTIONARY=%SAMPLE_DICT%

which is a symbolic name for the application dictionary defined as database file in the server file catalog.

When referring to a dictionary or resource database as database the system dictionary (ode.sys) must be defined as dictionary.

DICTIONARY_TYPE

The dictionary database can be stored in different data storage formats supported by ODABA.

DICTIONARY_TYPE=ODABA

ODABA storage format is the default because it is the most efficient format for storing dictionary information. The following formats are supported:

ODABA	- ODABA database format
XML	- XML file
ORACLE	- ORACLE database
MS_SQL	- Microsoft SQL database
ODBC	- Any relational database connected by ODBC
XSD	- XML schema

All formats except XSD are supported ODABA data storage formats, which simply store ODABA data in the requested format. XSD is an XML format, which contains an XML schema definition. The Schema definition may contain ODABA extensions (http://www.odaba.com/OXMLExtensions.xsd)

DATABASE

The path for the application database refers to the database created for the application.

DATABASE=C:\ODABA\Sample\Sample.dat

When using a local database the database should refer to a local database on your machine or another drive available in the network. When sharing data you should provide a database on a global machine or on a server.

For creating a new database you need to refer only to a new database name on a valid path where you are allowed to created and write new files.

When running the utility on a server, this variable should refer to the application database on the server, e.g.

DATABASE=%SAMPLE_DAT%

which is a symbolic name for the database defined in the file catalog on the server.

In a client server environment you may refer to dictionary or database location by means of symbolical path names (as %SampleBase%). In this case you must define the serve name and port number.

DATABASE_TYPE

The database can be stored in different data storage formats supported by ODABA.

DATABASE_TYPE=ODABA

ODABA storage format is the default because it's the most efficient format for storing complex data strucrues. The following formats are supported:

ODABA	- ODABA database format
XML	- XML file
ORACLE	- ORACLE database
MS_SQL	- Microsoft SQL database
ODBC	- Any relational database connected by ODBC

Relational database formats (ORACLE, MS_SQL and ODBC) require an ODABA relationship and transaction manager (RTM), which allows managing relationships and transactions efficiently. The RTM is a small ODABA database, which contains derived information from the original data. Since the RTM-Database does not add information to the underlying data, it can be reconstructed from the relational database at any time.

SERVER_URL

The server name refers to the ODABA server name or its TCP/IP address.

SERVER_URL=server_name

REPLICATION_ SERVER	For running a replication server, this variable holds the server name (instead of SERVER_URL). The variable refers to the replication server name or its TCP/IP address. **REPLICATION_SERVER**=server_name
SERVER_PORT	This variable is only necessary when running in a client server environment (object or replication server). In this case the port number must be identical with the port number passed to the server when starting it. This variable is only required in connection with the SERVER_URL or REPLICATION_SERVER variable. **SERVER_PORT**=6123 Default: 6123
OBJECT_SPACE	Often, a database consists only of the root object space and no object space must be defined. In some cases the database is, however, build of a number of hierarchical object spaces where each object space may contain a whole universe of object instances. Object spaces are referenced in the defined hierarchy. In order to access a subordinated object space, a path to the object space must be defined in the data source. Default: none (root object space) **OBJECT_SPACE=**section1.part2
ACCESS_MODE	The access mode defines whether the database will be used in write/update mode or read only. **ACCESS_MODE**=Write
NET	This option is required when running the database in a **file server** environment for using the database with more than one user (multi-user access). **NET**=YES This feature is supported under Windows, only. Under Linux, YES is used, always.
ONLINE_VERSION	This value enables online-versioning feature for the data source, which allows automatic upgrades to higher database model versions. **ONLINE_VERSION**=YES When this variable is not set or set to NO the application will not run with newer database versions.
VERSION	Internal database version number when the database is using version features. The version number allows seeting up the database according to a historical state. **VERSION**=version Default: current version

SCHEMA_ VERSION	Database version for the dictionary. When setting up an older version of the database you might run this with the appropriate dictionary version used at this time. Usually the system tries to detect the proper dictionary version. **SCHEMA_VERSION**=version Default: Version of the dictionary database
ENABLE_ CONTEXT	Specific object behaviour (e.g. when reading or writing objects) can be disabled. This option should be set to NO for database maintenance purposes, since context functions may cause errors in this case. **ENABLE_CONTEXT**=<u>YES</u> | NO Default: YES
WORKSPACE	The workspace option defines the basic workspace to be used by the utility. The utility can open workspaces on top of the base workspace but not below. The defined workspace is either the base for the operation (when passing a workspace name) or the workspace for running the operation (when no workspace name has been passed). A workspace can be defined only when the workspace feature is enabled. When not defining a workspace the database is opened directly. Default: none
[DATA_CATALOG]	Instead of explicitly defining the data source you may refer to a data source definition in a data catalog. The data catalog might be a local data catalog or a catalog defined on the server. When using a data catalog the data source can simply be defined as **DATA_SOURCE**=Sample Referring to a data catalog, the server (in case of client/server applications) or the application option file (in case of local applications) must define a [DATA_CATALOG] section that defines the location of the data source, or it must contain a section with the name of the data source.
DICTIONARY	The dictionary for the catalog is usually the system dictionary ode.sys, but it is also possible to use the project dictionary when the database contains more than the DataSource definitions. **DICTIONARY**=C:/ODABA/ode.sys

DATABASE

The database for the catalog is usually a separate maintenance database for the application or a specific data catalog database.

DATABASE=C:/ODABA/Server.cat

When you are running several applications on the server we suggest using a specific catalog database for the server.

ACCESS_MODE

Usually the data catalog is opened in read mode, only. If you want to allow update operations in the catalog (e.g. defining new data sources) you might run the data catalog in write mode as well:

ACCESS_MODE=Read | Write

NET

Since you are running the catalog usually in a network environment you should enable NET always:

NET=YES

This feature is supported under Windows, only. Under Linux, YES is used, always.

Create new database

New databases are created automatically (when not yet existing) when starting an application with write access to the database. When running multiple dataset databases, the database structure must be defined in advance (Class Editor) before creating the database.

Automatic database creation

A database will be created automatically when running an application that refers to the database as output database (write access). In this case you need to define the database path in the option file, which is passed to the application.

…/application *… ini_file …*]

Usually, the option file is passed as parameter to the application. The option file may refer to one or more data sources. The database is referenced in the data source as DATABASE (or DATDB in some older applications).

When a new database is created a message pops up notifying that a new database will be created. The message appears on the command line for command line applications or as a message box otherwise.

Data source

Depending on the application the data source is marked in the option file by a section name that refers to the name of the data source (e.g. [DATA_SOURCE1]). Applications based on a single data source use the application name as section name in several cases.

The data source can be defined also in a data catalog.

DICTIONARY

For defining a new database the dictionary for the database must be defined in the database. Older applications refer to the dictionary as RESDB.

DICTIONARY=C:/ODABA/Sample.dev

DATABASE

The database variable defines the path to the database. The folder that contains the database must point to a valid folder on the disk.

DATABASE=C:/ODABA/application.dat

ACCESS_MODE	A new database will be created only, when the access mode is set to Write. **ACCESS_MODE**=Write
PLATFORM_ INDEPENDENT	Usually, databases differ in the internal format that is stored on the disc, depending on the platform, where the database has been allocated. The consequence is, that a database cannot be copied as file e.g. from a SUN-station to an INTEL platform. To create a platform independent database that can run on any machine, you must set the PLATFORM_INDEPENDENT variable to YES: **PLATFORM_INDEPENDENT**=YES | NO Platform independent databases can be copied by simple file copy from one machine to another, independent on the hardware used on the machines. When this variable is not set to YES, databases must be converted using the DBCopy utility when hardware the platform for the database changes.

ODABA Server (Server)

When running ODABA applications in client/server mode you need to setup a server. Besides installing ODABA on the server machine, there are other steps necessary as catalog databases and configure client applications for running on server.

ODABA server supports two client types, i.e. the server can be accessed as object server or as replication server.

ODABA server access is faster in slow network environments, but requires administration resources in order to setup the server or service to run the server. Running applications on an ODABA server allows balancing resources between client and server. Business logic can be executed partially on the server and partially on the client side. Still, the database via an ODABA server is instance oriented, i.e. normally one server access per instance to be read is required. Buffered access and access via views may reduce the traffic load, but requires more afford in building the applications. You may run ODABA server via the internet, but usually, transfer speed is to low and it is more appropriate running an replication server in this case.

In contrast to the file server, the ODABA server sends update, delete and create events to all (registered) client property handles, which may react directly on those changes notified by the server. Clients may also sent application events and messages to other clients.

For enabling ODABA server access, the ODABA server location must be defined in the data source location and databases are referenced by symbolic names assigned by the administrator.

Object Clients

Accessing the server as object server provides access to persistent object instances for the object client. When running an object client, the server also executes business rules provided in appropriate context classes. Also queries (or access paths) are executed completely on the server side.

Object clients require a fast network and a powerful server.

Replication Clients

Replication clients do have a local copy of the database. The server receives updated from the clients, always, when an internal transaction has been finished (e.g. creating a new object instance). Transactions from each client are sent to all other clients that did not yet receive those transactions.

Replication clients need a replication database, which can be provided running the **DBReplication** utility.

When the client is not up to date, the system tries to transfer a complete backup of the database to the client. The backup must be located in the same folder as the database ending with .ozi (ODABA backup format). When the database is older then 2 days, the system automatically creates a new backup. It is suggested to run an automatic service, which creates a database backup each day to reduce the server burden at run-time.

Start Server

You can run the Server on a MS Windows machine or on a UNIX platform. The ODABA^{NG} server has been installed in "(installation)/bin" directory (UNIX) or in the ODABA^{NG} installation path (MS Windows). For running the server you start the server with an option file and a port number.

Server *port_number ini_file* [-D] [-q] [-h]

The same port number must be used later to connect to the server from the client.

port_number

The port number defines the communication port that is used for client connections.

Default: 6123

ini_file

The server can connect to several databases (applications). The databases for different applications are defined in the file catalog. This and other information is defined in the configuration or ini-file.

-D

You can run the server as service (demon) or in "Debug" mode. Pass the Debug option as third parameter to run the server in Debug-mode.

UNIX: In both cases you must ensure that the all libraries are stored in the default library path or you have to add the library path from the installation "(installation)/lib" to the LD_LIBRARY_PATH.

Stop Server

For stopping the server you just press “enter” when running in “Debug” mode. When running the server as Demon you just call

```
.../server --kill ini_file
```

ini_file is the complete path pointing to the server option file that has been used for starting the server.

Server Commands

Several server commands have been provided that can be used as command line tools in DOS or UNIX. Server commands act as clients and can be started from any computer in the network that has installed server commands or on the server itself.

All Commands are called like

.../**command** *server_name port_number parameter*

server_name	is the name of the machine the server is running on. The port number must be identically with the port number that has been passed for starting the server.
port_number	The port number must be identically with the port number that has been passed for starting the server.
parameter	Here, command specific parameters can be passed.

DBCheck

The DBCheck server command works similar as the CheckDB utility, but in client server mode. It allows checking a server database and produces a list with database problems.

.../**DBCheck** *server_name port_number*
dict_path db_path
[-C:*checks*] [-S:*srce*] [-T:*type*] [-R]
[-W] [-K:*time*]

In contrast to the CheckDB utility, the DBCheck server command does not require an option file. Instead, the administrator must know the exact location of dictionary and database on the server, which are passed as parameters to the command.

When running database checks, clients accessing the database to be checked are stopped, i.e. clients accessing the database must finish before DBCheck starts or will be killed by DBCheck. During check, it is also not possible to start new clients on the server.

The following topics contain a short description of the DBCheck parameters and options. More details are described in the CheckDB utility. .

dict_path	Exact dictionary location on the server.
db_path	Exact database location on the server.

C:*checks*	The list of check options determines the type of checks to be performed: I — Inverse reference check X — Index check G — GUID check Default: -C:IXG (running all checks) The order of these options does not play any role. You may also use capital or small letters except for the option key –C:
-S:*srce*	The source describes a property path to the collection or set of instances to be checked. Default: -S:* (check the whole database)
-T:*type*	The type defines the source type: C[ollections] — check all collections referenced by the source. Since the collection check makes sense for I and X, only, the GUID check (G) is ignored when being defined together with Type C. I[nstances] — Check all instances (or the references for all instances referenced in the source or in the database, when no source is defined. A[ll] — Check instances and collections. Default: -T:all (check instances and collections)
-K:*time*	The kill option allows stopping clients after a given time interval (time in seconds). When a time interval has been defined, the clients are informed, that the client will be stopped after ‘time’ seconds. When passing the –K option, only, the clients accessing the database to be checked are stopped immediately and without warning. When not passing the kill option, DBCheck will terminate with error when there are active clients using the database to be checked.
-R	Passing the repair option causes CheckDB repairing the problems detected, if possible.
-W	For showing warnings in the protocol, the warning option must be set.

When running backup, update clients accessing the database to be backed up are paused while running BackupDB, i.e. all updating transactions are paused until backup has finished.

DBBackup

The DBBackup server command works similar as the BackupDB utility, but in client/server mode. It allows creating a backup file from a database.

.../**DBBackup** *server_name port_number*
db_path [*location*] [wait]

In contrast to the BackupDB utility, the DBBackup server command does not require an option file. Instead, the administrator must know the exact location of the database on the server, which is passed as parameter to the command.

When running backup, update clients accessing the database to be backed up are paused, i.e. all updating transactions are queued until backup has finished.

The following topics contain a short description of the DBBackup parameters and options. More details are described in the BackupDB utility.

db_path

The database path defines the database location on the server, i.e. the path must be a valid server path.

location

The location is the path where the backup file will be stored. The location must be a valid path on the server.

wait

This is a timeout value in seconds for closing committing transactions. When there are (long) transactions not able to commit in the defined timeout interval, backup terminates with an error.

Usually, the transaction timeout can be limited to a few seconds, because the timeout applies to committing transactions, only. New commits will be blocked until the backup has finished.

Default: 1

DBRestore

The DBRestore server command works similar as the RestoreDB utility, but in client/server mode. It allows restoring a backup file to a database.

.../**DBRestore** *server_name port_number*
db_path [*location*] [*ktime*]

In contrast to the RestoreDB utility, the DBRestore server command does not require an option file. Instead, the administrator must know the exact location of the database on the server, which is passed as parameter to the command.

When running database restore, clients accessing the database to be restored are stopped, i.e. clients accessing the database must finish before restore or will be killed by DBRestore. During restore, it is also not possible to start new clients on the server.

The following topics contain a short description of the DBRestore parameters and options. More details are described in the RestoreDB utility. .

db_path The database path defines the database location on the server, i.e. the path must be a valid server path.

location The location is the path where the backup file has been stored. The location must be a valid path on the server.

ktime When running DBRestore in client/server mode, the database must be available exclusive for DBRestore, i.e. no clients must access the database. The kill time option allows stopping clients after a given time interval (time in seconds). When a time interval has been defined, the clients are informed, that the client will be stopped after '*ktime'* seconds.

When passing 0 for the *ktime* parameter, the clients accessing the database to be restored are stopped immediately and without warning.

When not passing the *ktime* parameter, DBRestore will terminate with error when there are active clients using the database to be restored.

KillAll The KillAll command will kill all clients. The KillAll command waits 'timeout' seconds (default: 300) before killing all clients. A message is sent to all clients (ODABA 8.0) immediately before the clients are killed.

.../**KillAll** *server_name port_number* [*timeout*]

Kill all sends a message to all active clients (ODABA 8.0) that the server will shut down within the time defined in the 'timeout' parameter. New clients cannot login anymore. A warning is sent to the clients that the client is going to be killed.

When clients are terminated abnormally no message can be sent. The result is that the server waits for the client to receive the kill message. For killing dead clients without message you can pass 0 as timeout value.

KillClient

The KillClient command can kill a specific client. The KillClient command waits 'timeout' seconds (default: 300).

.../**KillClient** *server_name port_number client_id* [*timeout*]

When the client is running several applications on his machine all applications are cancelled. The client_id is the number (id) for the client shown in the ShowClients command. A warning is sent to the client that the client is going to be killed.

When clients have terminated abnormally no message can be sent. The result is that the server waits for the client to receive the kill message. For killing dead clients you should pass 0 as timeout value to avoid sending a message to the client.

ShowClients

The ShowClients command will list all active clients. It will also list the time for last communication and other statistics.

.../**ShowClients** *server_name port_number*

The ShowClients command lists all clients with the current ID-numbers.

SendMessage

The SendMessage command will send a message to a specific client or to all active clients.

.../SendMessage *server_name port_number* [*client*]

Sending messages will work properly only when the application is reacting on messages sent from the server.

StartPause When pausing the server no more transactions can be committed until pausing the server is stopped (Stop-Pause). The server can pause only after finishing all running transaction commits. If any commit is still running after five minutes or a given number seconds (time_out) the server will not pause (error 323).

The …Pause commands can be used for keeping the database in a consistent state while backing up the database without closing the server. Pause commands should not be used when running long transactions as large imports or database reorganizations.

Transactions will not be committed anymore after pausing the server. The timeout interval for committing transactions is 10 minutes. When not being able to start committing the transaction within the time interval the transaction is cancelled.

Any application may access the database in the pause state as long as not writing to the database, i.e. as long as not storing transactions to the database.

.../**StartPause** *server_name port_number* [*time_out*]

When the server cannot pause after the given time_out interval or 5 minutes the command stops without pausing the server. For allowing storing data to the database again you must use the StopPause command.

StopPause This command stops pausing the server and reactivates the server, which allows committing further transactions and storing data to the database.

.../**StopPause** *server_name port_number*

ShutDown The command starts the server shut down. When beginning to shut down the server, no more transactions can be committed. The server tries to finishing all running transactions. If any transaction is running after five minutes or a given number seconds (time_out) the server will cancel those transactions (error 323).

To ensure database consistency before shutting down, you may run the **StartPause** command before calling **ShutDown**.

.../**ShutDown** *server_name port_number*
[*time_out*]

Server Settings

Server settings are defined in a server option file. The server option file contains four sections for describing catalogs and files as well as run-time parameters for the server.

[SYSTEM]

The system section contains setting allowing the system to run properly.

DICTIONARY

Points to the system dictionary (ode.sys). If this path is not defined correctly the system is not able to display explanatory error messages, Instead you will get only error numbers and location.

DICTIONARY=C:/ODABA/ode.sys

PROGPATH

Contains the location from which the server is running. Some functions need this location for extended services.

PRIGPATH=C:/ODABA

ODABA_ROOT

Contains the location where the ODABA system resides (ODABA installation path). Some functions need this location for extended services.

ODABA_ROOT=C:/ODABA

TRACE

Here the location for the error log can be defined. Usually this value is set in the system environment. It is, however, also possible to define the location in the option file.

TRACE=C:/temp

Under the location defined in the TRACE variable an error.lst file is created that contains a detailed error log. This file should be checked in case of errors on the server side.

Default: Value for TRACE environment variable.

[SERVER]

The server section contains several option variables controlling the behavior of the server.

The server section contains a location for storing the server process identification number when running the server (**SERVERPID**) as demon process and other server options.

This section may also contain additional option variables that are referenced in the context-library (business rules) associated with the object model. If the context library

	refers to prefixed option variables, appropriate sections have to be added to the server's configuration or ini-file.
SERVERPID	Location to the path for storing the server’s PID when running the process as demon.
REQUEST_ TIMEOUT	The timeout interval defines the time the system will wait until a server request will be cancelled. Timeout intervals are used in order to limit locking requests from clients, which may have been stopped working. Default: 600 (seconds).
MAX_BUFFER_ SIZE	When accessing data in block mode the number of blocks requested by the application program might conflict with optimal package sizes. To synchronize the server with the optimal package size you can set the optimal network package size as buffer limit for block access mode. Default: 0 (no restriction).
DSC_Language	This is the language for displaying system messages. System messages can be provided in different languages. When this variable is not set, no text is displayed for system messages, but an “Undescribed Error …” message is displayed, instead. Default: English
[CACHE]	Activating the cache requires a [CACHE] section in the option file that is passed when starting the server. The server allows defining only one cache section that applies on all data bases opened by the server.
IGNORE_ CACHE_ LOCK	To open the database regardless of the state of the cache lock flag you must set this variable to YES. Default: NO
SIZE	The maximum cache size is passed in megabytes in the SIZE variable. The size variable must be set to activate the cache. Default: 0 (cache inactive).
RELEASE	When the maximum cache size is reached the cache will be reorganized by removing entries not recently used. The size of area to be released in the cache is the percentage of area that will be released. Default: 25.

CHECK	The interval in seconds for checking the cache for updates. This value must be set to enable the write cache. Default: 0 (write cache disabled) High check values will improve the performance but reduce the security since modifications that are stored in the cache only will get lost when the system crashes for some reason. Suggested check values are between 60 and 300 seconds.
INTERMIDIATE	To avoid risks when writing back updates from the cache to the disc the cache can be written to an intermediate file before being stored to the database by defining a path for writing an intermediate file. Default: “” (direct write to database).
STATISTICS	You may request cache statistics setting this variable to YES. Default: NO (no statistics created).
CONNECTIONS	Caching connections is a feature supported in order to increase performance for stateless applications as web applications. In order to allow connection caching, this option has to be set to YES. Default: NO (connection cache not supported).
[DATA_CATALOG]	ODABA^{NG} allows registering data sources in a data catalog. This is usually stored in a maintenance database (e.g. Sample.mnt). Defining data sources in a data catalog makes data access easier and more flexible, since the administrator can define the access parameter for each data source. When using a data catalog, clients do not refer to dictionaries and databases but to catalog entries, instead (e.g. DATA_SOURCE=Sample). When running the server with data catalog client and server must refer to the same catalog entries.
DICTIONARY	The dictionary for the catalog is usually the application resource database, since the maintenance may contain also application specific resources, **DICTIONARY**=C:/ODABA/Sample.dev You can, however, also refer to the system dictionary (ode.sys) if you define only the catalog in the database.

DATABASE

The database for the catalog is either the maintenance database for the application or a specific data catalog database.

DATABASE=C:/ODABA/Server.cat

When you are running several applications on the server we suggest to use a specific catalog database for the server.

ACCESS_MODE

Usually the data catalog is opened in read mode, only. If you want to allow update operations in the catalog (e.g. defining new data sources) you might run the data catalog in write mode as well:

ACCESS_MODE=Read | Write

NET

Since you are running the catalog in a network environment you should enable NET always:

NET=YES

This feature is supported under Windows, only. Under Linux, YES is used, always.

[FILE_CATALOG]

The file catalog allows defining file variables that can be referenced in the client data sources or in the data catalog. When running the system with a data catalog the file catalog is usually part of the data catalog. You can, however, provide the file catalog in the server ini-file as well. When not using the data catalog you must define the file catalog in the server ini-file.

The file catalog contains a number of file variable definitions. Each file variable consists of a variable name and the file location.

File_variable=*path*

The location (path) must be defined in a way that allows the server to locate the file correctly. File variables can be referenced in data source definitions as %file_variable%:

DICTIONARY=%SYSTEM_DICT%

In this case the file variable SYSTEM_DICT referenced in the client application will be resolved to the complete path defined for the file variable in the server's ini-file.

The file catalog should at least contain the following definition:

SYSTEM_DICT=…/ode.sys (system dictionary)

Option file for Server

This is an example for a typical Windows server option file running without data catalog (minimum configuration)

```
[SYSTEM]        system section
DICTIONARY= C:\ODABA\ODE.SYS
ODABA_PATH=C:/ODABA
PROGPATH=C:/ODABA
[SERVER]
TRACE=C:\temp
TIMEOUT=600
[FILE_CATALOG]
SYSTEM_DICT=C:\ODABA\ode.sys
SAMPLE_DICT=C:\ODABA\Sample\Sample.dev
SAMPLE_DAT=C:\ODABA\Sample/Data\Sample.dat
```

Samples

You will find a sample for a server option file in the ODABA installation folder under MetaServer.ini. Samples for server command scripts are provided in the installation folder under the command name.

3 ODABA Server Service (ODA-BAServer)

Running an ODABA server under Windows (NT, 2000 or XP) as service requires the registration of this service. You may register any number of ODABA Database services for running servers for different applications (via different ports). Since an ODABA Server may serve any number of applications, you may, however, serve all applications with only one server, as well.

In UNIX environments you may use the server demon instead, which runs the server as a demon process in the background.

Register Service

For registering the server on a Windows Server machine, you need to run the ODABAServer program.

…/ODABAServer –I ini_file port_number

port_number

The port number defines the communication port that is used for client connections. The same port number is used later to connect the client to the server.

Default: 6123

Ini_file

The option file contains the name and the description of the service and the definition of the data and file catalog (see ODABA Server). Registering with different inifiles containing different service names allows registering different ODABA servers as service.

Configure the service

After installing the service you may configure the service with the Windows Service Comtrol Manager.

Open the Services Control Manager by:

1. Windows 2000 Professional:

   Right-click **My Computer** on the desktop, and then click **Manage**. In the dialog box that appears, expand the **Services and Applications** node.

2. Windows 2000 Server:

   Click **Start**, point to **Programs**, click **Administrative Tools**, and then click **Services**.

3. Windows NT version 4.0:

   Open this **Services** dialog box from Control Panel.

You should now see your service listed in the **Services** section of the window. Select your service in the list, right-click it, and then click **Properties**.

Startup type

The **Startup type** is set to ‘Manual’ after installation, which requires an administrator action to start the service. You may change this option to ‘Automatic’, which causes the system to start the service when starting the system.

Recovery

On the ‘Recovery’ sheet of the properties form you may change the failure reactions. It is suggested to select ‘Restart The Service’ for first and second failure (or always). The default setting is ‘Take No Action’.

Actions

Standard actions are supported for the service. Thus, you may stop or restart the server. Pausing the service will flush all write transactions and block further write requests until the pause is stopped by ‘Resume’. Pausing or stopping the server is necessary before making a database backup.

After installing the server, the start option is set to manual. For starting up the server automatically, you must change the settings in the Windows service.

Unregister Service

For un registering the server on a Windows Server machine, you need to run the ODABAServer program again, with the uninstall option -U.

…/ODABAServer –U ini_file

The option file is required in this case too, for identifying the service. You must use the same service name as for registering.

ODABAServer Service Settings

ODABAServer Service settings are defined in a server option file. They are the same as the server settings for the ODABA Server whith an additional [ODABAServer] section for registering the service (see also “Server Settings”).

[ODABAServer]	This section contains the definition of the service performed by the ODABA server.
NAME	Defines the name for the service. The name is used to store and identify the service in the services database. The name should start with ODABA or the application name. **NAME**=ODABA Server
DISPLAY_NAME	The display name is the name being used for showing the service in applications (e.g. Service Control Manager). **DISPLAY_NAME**=ODABA Database Server
DESCRIPTION	The description contains a short description of the service, which is also shown in the Service Control Manager. It may contain a longer text (up to 255 characters), but no line breaks. **DESCRIPTION**=The ODABA server provides access to ODABA databases
[SERVER]	The server section is structured in the same way as the server section for the ODABA Server.
[CACHE]	The cache section is the same way as the cache section for the ODABA Server.
[DATA_CATALOG]	The data catalog section is structured in the same way as the data catalog section for the ODABA Server.
[FILE_CATALOG]	The file catalog section is structured in the same way as the file catalog section for the ODABA Server.

Option file for ODABAServer

This is an example for a typical Windows server option file running without data catalog (minimum configuration)

```
[SYSTEM]       system section
DICTIONARY= C:\ODABA\ODE.SYS
ODABA_PATH=C:/ODABA
PROGPATH=C:/ODABA
[ODABAServer]
NAME=ODABA Server
DISPLAY_NAME=ODABA Database Server
DESCRIPTION=The ODABA server provides access
             to ODABA databases
[SERVER]
TRACE=C:\temp
TIMEOUT=600
[FILE_CATALOG]
SYSTEM_DICT=C:\ODABA\ode.sys
SAMPLE_DICT=C:\ODABA\Sample\Sample.dev
SAMPLE_DAT=C:\ODABA\Sample/Data\Sample.dat
```

4 Setup Replication Database (DBReplication)

The ODABA replication server allows running databases on an internet server. Thus, clients distributed all over the world may access the same ODABA database.

Replication server access is transaction based and works on a local copy (replicate) of the master database. Replication server is the preferred access mode for distributed clients in the internet. In this case, the server must run on an internet server, which is known to the clients.

Since reading data happens with local access speed, replication server access is fast as long as update load is low. Long transaction or frequent updates (more than 10 transactions per second and user) may cause delay, since the replication server will serialize update requests from different clients.

Updates are sent to the clients when starting up a client. Running clients receive a dirty flag, which causes the client synchronizing its local database when executing the next database function. While synchronizing (receiving latest transaction data), appropriate update events are sent to registered property handles.

For running a database as replication database, it must be enabled as such.

Synchronize Database

When starting a replication client, the client is synchronized with the server by loading all missing transactions, i.e. all transactions registered after the last client session. While running the replication client, the local database copy is updated always, when starting a read transaction, i.e. when requesting data from the database.

Transaction limit

When the number of transactions becomes very large, starting up an application may take some time. Hence, the number of transaction stored for synchronisation can be limited (default: 5000). When a replication client starts that cannot be updated by the transactions registered, a backup of the database is sent to the clients.

The backup must be located on a WEB folder which is defined as SERVER_URL in the server.ini file. The backup file name must correspond to the database file name with the extension replaced by .ozi (ODABA backp format). It is suggested to run an automatic service, which creates a database backup each day (or night) to reduce the server burden at run-time.

Enable/disable

For supporting replication clients, the database must enable replication clients. This can be done by calling

DBReplication *path enable* [*ta_limit*]

You may disable the database for replication clients by calling

DBReplication *path disable*

After enabling the replication client, each internal transaction will be registered in the database regardless on the mode, in which the database is running. Thus, each replication database can be used as master database.

When explicitly disabling the replication database, the unique database number will be reset.

path

Path to the database location or ini-file. The database must be accessible on the local machine or as file share. In case of encoded database, the ini-file is required in order to provide the access key. The ini-file must contain a `DATABASE` section defining at least database and dictionary path (see example below). When using an external key file, `KEY_FILE` is required in addition.

```
; minimum DATABASE section for an en-
coded
; database
[DATABASE]
DICTIONARY=...dict path...
DATABASE=...database path...
ACCESS_MODE=Write
SHARE=NO
```

enable/disable

Enable or disable replication mode for database. After disabling the replication mode, the server will not provide replication services for this database anymore.

ta_limit

When enabling the replication mode for the database, you may define the maximum number of transactions stored for synchronization.

Re-enable

You may re-enable (DBReplication enable …) a replication database. This causes increasing the version number of the replication database. Clients running with an older version of the replication database will automatically reload the database when starting an application.

Re-enabling the database allows also changing the maximum transaction number for the database. Moreover, the transaction-log is cleared.

Auto-disable

In some cases, for maintenance reasons the replication master base is used in a local environment. To ensure database consistency, the replication feature for the database is deactivated automatically, when opening the database un update or write mode.

Auto-enable

When moving the maintained master database back to the server, the server detects a deactivated version and will not startup. When the server environment (or option file) provides a path to the local database backup location (SERVER_LOCAL_URL), which refers to the same location as the SERVER_LOCAL, the database will be reactivated automatically during server startup. While reactivating the master database, the server creates an actual backup file and stores this to the location defined in SERVER_LOCAL_URL.

Limitations

Replication databases do have some limitations, which will be removed in later versions.

PIF

Replication databases require compatible hardware on client and server side, i.e. both, client and server must be high-endean, or both must be low Endean machines.

Database objects

Replication databases do not support multiple database objects in this version, i.e. the database must consist only of the root object (or other sub-database objects are not of interest).

Versions

Replication servers do provide data for the current version, only, i.e. you may not access older versions via the server. This is not really a problem, since usually older versions are already present on the client database. It means, however, that the master database must completely be copied, when versioning has been made.

Configure Replication Clients

A client may use its database/databases in replication client mode, object client mode or local mode. Thus, an application may use any combination of access modes.

The access mode is defined for each data source in the application. When running an application database as replication client, usually only the database is defined as replication database, but not the dictionary.

Data source

```
[DATA_SOURCE]
REPLICATION_SERVER=www.run-software.com
SERVER_PORT=6123
DICTIONARY=c:\ODABA\my_appl\my_appl.dict
DICTIONARY.CONNECTION=LOCAL
DATABASE==%MY_DATABASE%
MY_DATABASE=c:\ODABA\my_appl\my_appl.dat
```

The database path refers to a file name, which must be catalogd in the servers file catalog. Also, the option name has to be defined as option (here MY_DATABASE), In order to make sure, that the correct dictionary is used, DICTIONARY.CONNECTION may be set to LOCAL or SERVER. Default is AUTOMATIC.

In a data source for a replication database one may refer to a local or server dictionary. Whether using a local or a server dictionary depends on the fact, whether the path referenced in the dictionary refers to a local file (local dictionary) or not (server dictionary). you cannot refer to a server dictionary, i.e. the dictionary path must always be a local path.

Using a replication database is indicated by using REPLICATION_SERVER instead of SERVER_URL.

Database

The database referenced on the server must be enabled for replication. The local database location for the replication database is defined as value for the variable name defined in the DATABASE reference.

Authentication

In order to download the database replicate from the WEB-server, authentication might be required. User name and password for downloading the database must be provided in following options:

REPLICATION_HTTP_USERNAME=*http_user_name*
REPLICATION_HTTP_PASSWORD=*http_password*

These options can be set by the application program before opening a replication database or in the database section of the client configuration or ini-file.

Force loading replicate

When testing replication server applications it might be useful reloading the database when starting the application. In order to force database reload rather than database synchronization, the RELOAD_RDB option can be set to one of the following values:

NO	The system tries to synchronize the local database copy with the replication server.
YES	The server database is reloaded (and overwrites the local database) when starting the application.
AUTO	The system prompts the user in order to decide whether to reload the database or not.

When not setting the RELOAD_RDB option the local database will be synchronized with the server database (NO).

5 Database Set-up (SetupDB)

SetupDB allows upgrading or repairing a database. It works similar to the CopyDB utility but it is easier to use. SetupDB copies the database to a temporary database. After successfully copying the database the old database is deleted and the updated database is moved (or renamed) to the original position.

When there is not enough space on the disc for a duplicate of the database you can define a temporary path for creating the intermediate database.

Usually you only need to define the data source for the database to be reorganized. You may, however, define the target data source when you are going to keep the duplicate of the database in another location.

In contrast to CopyDB SetupDB maintains the object identities for the object instances and the database identifier, i.e. you are really going to create a duplicate. Since instances updated in one database are not updated in the other one, it is advised to define always one primary source when creating copies using SetupDB.

When setting up the database the current database version is copied and the new database version will be reset to 0. Older versions of the database are not copied.

Running SetupDB

Usage	You can run SetupDB from a command line in DOS or UNIX. Before running SetupDB make sure that the data sources are available. **SetupDB** *ini_file from* [*to*]
Ini_file	The option file defines the data sources for the data-bases and specific copy parameters. More details on how to define the option file you can find in "Option file for SetupDB"
from	is the data source name for the source database. The data source must be defined either in the catalog or in a section with the same name in the option file.
to	is the data source name for the target database. The data source must be defined either in the catalog or in a section with the same name in the option file. When no target source is defined a temporary copy will be created and renamed to the original database name after suc-cessful set up. In the case the drive where the database is stored must have enough space for the temporary database copy.
Samples	You will find sample procedure calls and option files for Windows environment in the ODABA^{NG} installation folder under SetupDB.bat and SetupDB.ini.

Defining data sources

SetupDB copies all instances from one data source to another one or into the same data source. There are two ways to refer to a data source. One way is to define the data source in the application option file. In this case the data source is defined in a section that is preceded by the data source name:

{DataSource1] Data source name

The other way is to refer to a data source by its data source name:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the “ODABA – Server” documentation.

Database and dictionary

A dictionary and a database define a data source. While the dictionary contains the data definitions the database contains the data. Dictionary and database can be stored in the same database file but usually they are not. In any case, each data source definition should contain a dictionary definition and a database definition:

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat

In special cases /when copying instances from one dictionary to another one) only the dictionary defines the data source.

Data sources can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=MetaSever
SERVER_PORT=6123

DICTIONARY=%SAMPLE_DICT%
DATABASE==%SAMPLE_DAT%

Options

Additional options that can be defined in a data source are:

ACCESS_MODE	The access mode defines whether the database will be used in write/update mode or read only. **ACCESS_MODE**= <u>Read</u> | Write Defaulr: Read
NET	This option is required when running the database in a file server or client/server environment for using the database with more than one user (multi-user access). **NET**=YES | <u>NO</u> This feature is supported under Windows, only. Under Linux, YES is used, always. Default (Windows): NO
ENABLE_ CONTEXT	specific object behaviour (e.g. when reading or writing objects) will be disabled. This option should be set to NO when setting up a database since context functions may cause errors when running setup. **ENABLE_CONTEXT**=<u>NO</u> | YES Default: NO

Defining Copy Sequences

Usually when copying a database the extents are copied in alphabetic order of the extent names. In some cases, however, it becomes necessary to copy extents in an order that differs from the alphabetic one. This is necessary when you define reference collections in your database that are based on extents that will be copied later.

Example: When defining the following structures:
Structure Person

attribute	pers_id (CHAR)
relationship	company (XCompany) based_on XCompany, secundary
relationship	section (XSection) based_on company.sections

When copying the database the companies are not copied with the person since they are defined as secondary, i.e. the link between XCompany and Person is maintained when copying XCompany. Since ‘section’ is based on the company relation sections can be copied only when a link to a company is stored. Hence you need to copy XCompany first.

Defining a copy sequence can be done via a copy collection in the dictionary. Create a SDB_Collection (e.g. ‘CopyDatabase’)that names the extents in the required order. Then refer to the collection in the option file as

COPY_COLLECTION=CopyDatabase

In the [SetupDB] section. Make sure that you refer to all extents because the SetupDB utility will copy only those extents that are referenced in the list.

Copy collections can be used as well for copying selected parts of the database. In any case, copy collections have to be defined in the dictionary. You can refer to copy collections also when running the SetupDB utility.

Option file for SetupDB

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters.

```
[SYSTEM]        system section
DICTIONARY= C:\ODABA\ODE.SYS
[SetupDB]
SYSTEM_ENVIRONMENT=…\MetaServer.ini[1]
REPLACE=database
COPY_COLLECTION=collection_name
[from_source]
DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
NET=NO
[to_source]
SERVER_URL=192.168.0.23
SERVER_PORT=6123
DICTIONARY=%SAMPLE_DICT%
DATABASE=%SAMPLE_DAT%
ENABLE_CONTEXT=NO
NET=NO
ACCESS_MODE=Write
```

When referring to data sources defined in the data catalog on the server you need not to define the data source sections for source and target data source.

[1] The MetaClient.ini file has been generated in the ODABA^{NG} installation folder when installing the client. When defining data sources on the server the dictionary must be available at the server <u>and</u> on the client machine. Please make sure that in this case the symbolic dictionary variable in the MetaClient.ini points to the correct location for the dictionary on your client machine.

6 Database Copy (CopyDB)

CopyDB allows copying complete databases or parts of it. You can use CopyDB for copying a single object instance from database to another one. CopyDB is a command line utility that can be used in Windows and UNIX environments.

With CopyDB you duplicate parts or the whole content of a database. Instances are considered “new” instances, i.e. usually when copying database instances they will get a new identity.

When copying the database you might copy the database with or without running the business rules. It is suggested to run the database copy without business rules (deactivated context).

When copying the database the database version will be reset to 0. Older versions of the database are not copied.

PIF

Platform independence is copied to a new database unless it is set explicitly using the PLATFORM_INDEPENDENT system variable.

Replication database

When the database is a replication database, RDB properties as maximum number of transactions are copied. The RDB version is increased by one.

Running CopyDB

Usage

You can run CopyDB from a command line in DOS or UNIX. Before running CopyDB make sure that the data sources are available.

CopyDB *ini_file from to* [-S[:*date*]]

The optional parameters can be defined in the option file as well. The content of a option file for CopyDB is described in the following section.

Ini_file	The option file defines the data sources for the databases and specific copy parameters. More details on how to define the option file you can find in "Option file for CopyDB"
from	is the data source name for the source database. The data source must be defined either in the catalog or in a section with the same name in the option file.
to	is the data source name for the target database. The data source must be defined either in the catalog or in a section with the same name in the option file.
-S[:*date*]	In order to synchronize two databases by date (updating the to-database), the option has to be set. Then, all newer instances inheriting from __OBJECT will be replaced including owned instances not inheriting from __OBJECT. When passing a date, all instances newer or equal to the date are copied from the source database. Date has to be passed as yyyy-mm-dd.

Samples

You will find sample procedure calls and option files for Windows environment in the ODABA^{NG} installation folder under CopyDB.bat and CopyDB.ini.

Defining data sources

CopyDB copies instances from one data source to another one. There are two ways to refer to a data source. One way is to define the data source in the application option file. In this case the data source is defined in a section that is preceded by the data source name:

{DataSource1] Data source name

The other way is to refer to a data source defined in a database catalog by its data source name:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "ODABA^{NG} – Server" documentation.

Database and dictionary

A dictionary and a database define a data source. While the dictionary contains the data definitions the database contains the data. Dictionary and database can be stored in the same database file but usually they are not. In any case, each data source definition should contain a dictionary definition and a database definition:

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat

In special cases (when copying instances from one dictionary to another one) only the dictionary defines the data source.

Data sources can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=ProjectServer
SERVER_PORT=6123

DICTIONARY=%SAMPLE_DICT%
DATABASE==%SAMPLE_DAT%

Database objects

This data source definition refers to the whole database. If you want to refer to parts of the database you can define the following restrictions. Usually a database consists only of the root object. In some cases the database itself is, however, build of a number of hierarchical database objects where each database object may contain a whole universe of object instances. Database objects are referenced in the defined hierarchy:

OBJECT_SPACE=section1.part2

Extents or collections

You can also restrict the data source to a certain extent in the database or in a database object. For this purpose you need only to define the extend name in the data source:

ACCESS_PATH=State

Object instances

Special object instances can be defined by setting the identification key in the data source:

ACCESS_KEY=Sweden

Options

Some options that can be defined in a data source are:

VERSION

Internal database version number when the database is using version features. The version number allows seeting up the database according to a historical state.

VERSION=version

Default: current version

SCHEMA_ VERSION

Database version for the dictionary. When setting up an oldel version of the database you might run this with the appropriate dictionary version used at this time. Usually the system tries to detect the proper dictionary version.

SCHEMA_VERSION=version

Default: Dictionary version for the database

ONLINE_VERSION

This value enables online-versioning feature that allows automatically upgrading to higher database model versions.

ONLINE_VERSION=YES

When this variable is not set or set to NO the application will not run with newer database versions.

ACCESS_MODE

The access mode defines whether the database will be used in write/update mode or read only.

ACCESS_MODE= Read | Write

Default: Read (for from-data source), Write (for to-data source)

NET	This option is required when running the database in a file server or client/server environment for using the database with more than one user (multi-user access). **NET**=YES **|** NO This feature is supported under Windows, only. Under Linux, YES is used, always. Default (Windows): NO
ENABLE_ CONTEXT	specific object behaviour (e.g. when reading or writing objects) is usually disabled. In order to enable context functions, this option has to be set to YES. This option should be set to NO when setting up a database since context functions may cause errors when running setup. **ENABLE_CONTEXT**=NO | YES Default: NO

Defining Copy Sequences

Usually when copying a database the extents are copied in alphabetic order of the extent names. In some cases, however, it becomes necessary to copy extents in an order that differs from the alphabetic one. This is necessary when you define reference collections in your database that are based on extents that will be copied later.

Example: When defining the following structures:
Structure Person

attribute	pers_id (CHAR)
relationship	company (XCompany) based_on XCompany, secundary
relationship	section (XSection) based_on company.sections

When copying the database the companies are not copied with the person since they are defined as secondary, i.e. the link between XCompany and Person is maintained when copying XCompany. Since ‘section’ is based on the company relation sections can be copied only when a link to a company is stored. Hence you need to copy XCompany first.

Defining a copy sequence can be done via a copy collection in the dictionary. Create a SDB_Collection (e.g. ‘CopyDatabase’) that names the extents in the required order. Then refer to the collection in the option file as

COPY_COLLECTION=CopyDatabase

In the [SetupDB] section. Make sure that you refer to all extents because the SetupDB utility will copy only those extents that are referenced in the list.

Copy collections can be used as well for copying selected parts of the database. In any case, copy collections have to be defined in the dictionary. You can refer to copy collections also when running the SetupDB utility.

Replace options

replace_opt	The replace option defines the way to handle existing instances in the target database. Replace options have to be defined in the REPLACE option in the option file as suboption for CopyDB.
REPLACE	The following values might be set for the replace option:

all	-	overwrite all existing instances and referenced instances
none	-	do not replace existing instances
local	-	overwrite all existing instances and referenced instances that are owned by the instance. Instances that are referenced but not owned are not replaced.
no_create	-	overwrite all existing instances but do not create new instances.
database	-	when copying the whole database to a new one (database reorganization) you should define this option. This is using a more efficient copy technique and avoids recursive copy operations.

Option file for CopyDB

ODABA^NG

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[CopyDB]

SYSTEM_ENVIRONMENT=…\MetaServer.ini[2]
REPLACE=database

COPY_COLLECTION=*collection_name*

[from]
DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
NET=NO
ACCESS_PATH =* **(**copy all extents)
ACCESS_KEY=* (copy all instances)
ENABLE_CONTEXT=NO

ONLINE_VERSION=YES

[to]

SERVER_URL=192.168.0.23
SERVER_PORT=6123
DICTIONARY=%SAMPLE_DICT%
DATABASE=%SAMPLE_DAT%

ENABLE_CONTEXT=NO

When referring to data sources defined in the data catalog on the server you need not to define the data source sections for source and target data source.

[2] The MetaClient.ini file has been generated in the ODABA^NG installation folder when installing the client. When defining data sources on the server the dictionary must be available at the server and on the client machine. Please make sure that in this case the symbolic dictionary variable in the MetaClient.ini points to the correct location for the dictionary on your client machine.

7 Resource Database Copy (CopyResDB)

CopyResDB allows copying complete resource databases or parts of it. You can use CopyResDB for copying a single resource object instance or structure definition from one resource database to another one. CopyResDB is a command line utility that can be used in Windows and UNIX environments.

With CopyResDB you duplicate parts or the whole content of a resource database. Instances are considered an "new" instances, i.e. usually when copying database instances they will get a new identity.

When copying the resource database the schema version will be reset to 0. This requires a database set-up (SetupDB) for databases running with the old resource database when the old resource database has a version higher then 0.

PIF

Platform independence is copied to a new resource database unless it is set explicitly using the PLATFORM_INDEPENDENT system variable. Big endian

Replication database

When the database is a replication database, RDB properties as maximum number of transactions are copied. The RDB version is increased by one.

Running CopyResDB

Usage

You can run CopyResDB from a command line in DOS or UNIX. Before running CopyResDB make sure that the data sources are available.

CopyResDB *ini_file from* [*to*]

The content of a option file for CopyResDB is described in the following section.

Ini_file	The option file defines the data sources for the databases and specific copy parameters. More details on how to define the option file you can find in “Option file for CopyResDB”
from	is the data source name for the source database. The data source must be defined either in the catalog or in a section with the same name in the option file.
to	is the data source name for the target database. The data source must be defined either in the catalog or in a section with the same name in the option file.

When not defining the to parameter the database is copied “in place” (reorganisation of the resource database). This means that the database is copied to a database with the same database name in the same location by changing the suffix to #_o (e,g, to sample.#_o).

When the copy has been finished succesfully the original database is deleted and the new database is renamed to the original database name. This may fail when the database is in use by another application. In this case you may remove and rename the files manually after closing all applications using the old resource database.

Samples

You will find sample procedure calls and option files for Windows environment in the ODABA^{NG} installation folder under CopyResDB.bat and CopyResDB.ini.

Defining data sources

CopyResDB copies structure definitions or other resource instances from one resource database to another one or into the same data source. There are two ways to refer to a data source. One way is to define the data source in the application option file. In this case the data source is defined in a section that is preceded by the data source name:

[DataSource1] Data source name

The other way is to refer to a data source by its data source name:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "ODABA^{NG} – Server" documentation.

Database and dictionary

A dictionary and a database define a data source. When copying a resource database the application dictionary is considered as database while the system dictionary (ODE.SYS) plays the rule as dictionary. The system dictionary is part of the ODABA^{NG} installation and usually stored in the ODABA^{NG} installation path.

DICTIONARY=C:\ODABA\ode.sys
DATABASE=C:\ODABA\Sample/Sample.dev

Data sources can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=ProjectSever
SERVER_PORT=6123

DICTIONARY=%SYSTEM_DICT%
DATABASE==%SAMPLE_DICT%

Copy Structures

For copying schema definitions (structures or enumerations) you can define the DATA_TYPE variable

DATA_TYPE=*data_type_name*

The *structure_name* refers to the structureor enumeration to be copied. When copying a structure defintion all related resources as extent definitions or persisten structure definitions are copied as well as all referenced structure definitions. Thus, copying a single structure may take some time.

For copying all structures you can define the STRUCTUE variable as

DATA_TYPE=*

Extents or collections

You can copy other resource database instances by defining the extend to be copied. For this purpose you need only to define the extend name in the data source:

ACCESS_PATH =p_err (copy error definitions)

Instances

Special object instances can be selected for copying by defining the identification key in the data source:

ACCESS_KEY=100 (copy p_err 100)

The ACCESS_KEY option is valid only in connection with the ACCESS_PATH variable. For copying all instances for a selected extent you might define

ACCESS_KEY=*

Or just skip the the ACCESS_KEY variable in your option file.

Do not copy schema definitions as SDB_Structure or SDB_Codeset using the extent and instance variables. This will create invalid structure definitions in the target database. For copying schema definitions you should always use the DATA_TYPE variable.

Data source Options

Some more options that can be defined in the CopyResDB section of your option file are:

VERSION

Internal database version number when the database is using version features. The version number allows copying the resource database according to a historical state.

VERSION=version

Default: current version

ACCESS_MODE

The access mode defines whether the database will be used in write/update mode or read only.

ACCESS_MODE= Read | Write

Default: Read (for from-data source), Write (fot to-data source)

NET

This option is required when running the database in a file server or client/server environment for using the database with more than one user (multi-user access).

NET=YES | NO

This feature is supported under Windows, only. Under Linux, YES is used, always.

Default (Windows): NO

CopyResDB Options

Some more options can be defined in the CopyResDB section of your option file. These options are valid only when copying structure or enumeration definitions from one resource database into another. You should use this option only when copying into an empty resource database.

RETAIN_SID

Structures are identified internally by numbers created in the database system. This internal type number is used to identify the type of a stored database instance. Whe you are going to use the copied dictionary with an already existing database you must define

RETAIN_SID=YES

You should not use this option when copying single structure definitions from one resource database to another one.

RETAIN_SCHEMAVERSION

This option will copy the schema version for each structure definition. This is necessary to guaranty the compatibility with old databases when using schema versioning features.

RETAIN_SCHEMAVERSION=YES

Otherwise the schema version for all structure definitions is set to 0 and ols databases running with the dictionary must be reorganized (SetupDB).

Copyinf the the schema version does not mean that the history is copied. The consequence is the after copying the database ONLINE VERSIONING will not work anymore for older databases.

Option file for CopyResDB

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[CopyResDB]

RETAIN_SID=YES

RETAIN_SCHEMAVERSION=YES

[from]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample.dev
NET=NO
DATA_TYPE=*
ACCESS_PATH =* **(**copy all extents)

ACCESS_KEY=* (copy all instances)

[to]

SERVER_URL=192.168.0.23
SERVER_PORT=6123
DICTIONARY=%SYSTEM_DICT%

DATABASE=%SAMPLE_DICT%

When referring to data sources defined in the data catalog on the server you need not to define the data source sections for source and target data source.

Use Cases for CopyResDB

This section describes some typical use cases for running CopyResDB.

Reorganize

Reorganizing the resource database becomes necessary when the database is damaged for some reason. Damages may affect indexes or links to their objects. In most cases reorganizing the resource database can repair such damages. Another case for running CopyResDB is the release of a new ODABA system version. Usually this is supported by online versioning features for schema. Since the system level supports, however, only one historical version we suggest to run CopyResDB in this case to make sure that your application is ready for the next system release.

Option file

```
[SYSTEM]        system section
DICTIONARY= C:\ODABA\ODE.SYS
[CopyResDB]
RETAIN_SID=YES
RETAIN_SCHEMAVERSION=YES
[from]
DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample.dev
NET=YES
[to]
DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample.dev.new
NET=NO
```

Start copy

```
C:\ODABA\CopyResDB ini_file  from to
```

Remarks

This example allows reorganizing the database when the dictionary is still in use (NET=YES in the from data source). After the reorganization has been completed successfully you should rename the database to the name of the original database.

When setting up a new system version you should save the old system dictionary ode.sys before installing the new ODABA^{NG} version (e.g. ode.old). Than you should use the old system dictionary as DICTIONARY in the from data source section.

Do never run SetupDB for reorganizing a resource database. This will fail because structure definitions are not copied properly.

Copy Structure with rename

If you are going to use a structure for another application you can copy the structure from one dictionary to another. Doing this you can also rename the structure definition.

Option file

```
[SYSTEM]        system section

DICTIONARY= C:\ODABA\ODE.SYS

[CopyResDB]

[from]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample1.rot
DATA_TYPE=Person

NET=YES

[to]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample2.rot
DATA_TYPE=ContactPerson

NET=YES
```

Start copy

```
C:\ODABA\CopyResDB ini_file  from to
```

Remarks

Only for renaming the structure you must define the DATA_TYPE variable in the [to] data source section. Structure references to a new named structure are maintained for structures copied subsequently.

You might copy structure definitions also when databases are in use.

When copying a structure definition all referenced structures are copied as well. You can, however, rename only the structure that is copied “on top”.

Copy Instances with rename

If you are going to copy specific resources as e.g. window definitions (swn) or document templates into another resource database you might copy a single instance with or without rename.

Option file

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[CopyResDB]

[from]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample1.rot
ACCESS_PATH =ADK_Class
ACCESS_KEY=Person

NET=YES

[to]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\Sample2.rot
ACCESS_KEY=PersonForm

NET=YES

Start copy

C:\ODABA\CopyResDB ini_file from to

Remarks

Only for renaming the instance you must define the ACCESS_KEY variable in the [to] data source section.

You might copy resource instances also when databases are in use.

When copying instances or extents all linked instances are copied recursively except those that are defined as "secondary" in the relationship definition.

8 Pack Database (PackDB)

The PackDB utility creates a compressed copy of the database by removing unused (deleted) database entries. The copy of the database will be created in the same folder. If there is not sufficient space on the disk you can use temp-path for defining another location for the intermediate file. When finishing the compressed database will be moved or renamed to the original position.

Running PackDB

You can run PackDB from a command line in DOS or UNIX. Before running PackDB make sure that the database is available.

PackDB.exe *db_name* [*temp_path*] [-q] [-h] [-p:*type*]

PackDB will compress the database. After compressing the database deleted instances cannot be 'revived'.

db_name

Is the complete name (path) of database to be packed.

You may pack all types of databases as application databases, resource databases or system databases.

temp_path

There should be sufficient space in the current location (folder) of the database for storing a copy of the database. If there is not sufficient space you can pass a path to a location where a temporary copy of the database is stored.

Samples

You will find sample procedure calls in a Windows environment in the ODABA^{NG} installation folder under PackDB.bat.

Database Backup (BackupDB)

The tool supports an ODABA specific database backup. Backing up an ODABA database creates a compressed database copy, which can be restored using RestoreDB.

Since BackupDB backs up nearly 10 MB per second on a 3 GHz Intel processor with 2GB memory, the interrupt for updating clients might be acceptable, i.e. BackupDB can be started also when clients are working actively on the database.

One should, however, not run BackupDB while importing large amount of data, since this may lock the import process for a while.

For restoring a backup file the RestoreDB utility can be used.

Client/server

The BackupDB utility works successfully only, when no other application or server is accessing the database.

It may run locally, only, but not in client/server mode. Since BackupDB requires a consistent database state, it cannot run in parallel with other database applications. BackupDB ensures that it runs exclusively and will terminate with errors, when other applications are accessing the database.

Making backups in client/server environment is possible with the similar server command **DBBackup**.

Running BackupDB

You can run BackupDB from a command line in DOS or UNIX. Before running BackupDB, make sure that the data base is available. It is suggested to run BackupDB exclusively, but you may also run it in Client/Server mode. You cannot run BackupDB locally, when other applications are accessing the database updating.

BackupDB *ini_file* [*target*] [-q] [-h] [-p:*type*]

The details for the data source are described in the configuration or ini-file. The utility backs up the database, only, but not the dictionary.

ini_file

The configuration or ini-file defines the data source for the database. More details on how to define the option file you can find in “Option file for BackupDB”

target

The target is the file path for storing the backup file. When passing a directory instead of a complete file path, a backup file name is the same as the database name with the extension **ozi**, which is the suggested extension for backup files. When not passing a location, the backup file is stored in the same directory as the database.

When running in client/server mode, the location must be a valid location on the server.

Option file for BackupDB

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[BackupDB]

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
ACCESS_MODE=Write
NET=NO

Database Restore (RestoreDB)

RestoreDB supports restoring an ODABA specific database backup. The backup of the database must have been created using BackupDB.

Client/server

The RestoreDB utility works successfully only, when no other application or server is accessing the database.

It may run locally, only, but not in client/server mode. Since RestoreDB restores a complete database, it cannot run in parallel with other database applications. RestoreDB ensures that it runs exclusively and will terminate with errors, when other applications are accessing the database.

Restoring a database in client/server environment is possible with the similar **DBRestore** server command.

Running RestoreDB

RestoreDB supports restoring an ODABA specific database backup, which has been created calling BackupDB.

You can run RestoreDB from a command line in DOS or UNIX. Before running RestoreDB, make sure that the data base is available. RestoreDB must run exclusively, i.e. no other clients may access the database to be restored.

You may also run RestoreDB in Client/Server mode. In this case RestoreDB checks the server for running clients and stops clients, when necessary.

You cannot run RestoreDB locally, when other applications are accessing the database.

RestoreDB *ini_file* [*source*] [-q] [-h] [-p:*type*]

The details for the data source are described in the option file. The utility backs up the database, only, but not the dictionary.

ini_file	The configuration or ini-file defines the data source for the database. More details on how to define the option file you can find in “Option file for RestoreDB”
source	The source is the path where the backup file has been stored. The path must refer to a valid ODABA backup file, which usually (but not necessarily) have the extension **ozi**. ODABA backup files have a four byte identification (“SOSZ”) in the file.

When no filename is passed, the backup file name is supposed to be the same as the database file name, just with the different extension ozi.

Option file for RestoreDB

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[RestoreDB]

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
ACCESS_MODE=Write
NET=NO

Check Database Consistency (CheckDB)

The database consistency check allows performing different types of consistency checks for the whole database or specific parts of the database. It provides also features for repairing inconsistent database states.

Client/server

The CheckDB utility may run locally, only, but not in client/server mode. It works successfully only, when no other application or server is accessing the database.

Since CheckDB may change the database on a low access level, it cannot run in parallel with other database applications. CheckDB ensures that it runs exclusively and will terminate with errors, when other applications are accessing the database.

Checking a database in client/server environment is possible with the similar **DBCheck** server command.

Check functions

Several types of checks are supported by CheckDB.

Inverse reference check

In some cases, inverse references might be inconsistent in a way, that an instance A points to B, but B does not have the inverse reference to A. Such problems can be detected (and repaired) by running the Inverse Reference Check.

Index check

Each collection in the database is based on one or more indexes. The Index Check detects corrupt indexes and repairs them.

Even though ODABA takes care about reference consistency, there might be situations, in which references point to instances that have already been deleted. The index check determines such ‘dangling’ pointers and removes them from the index.

GUID check

Global unique identities (GUID) are unique identifier, which are maintained also when copying objects between databases or when reorganizing a database. GUIDs are specific keys, which are created on demand.

For some reasons, there might be problems with GUIDs, which are not stored in the GUID index or not created properly. To detect GUID-problems and repair them, you may run the GUID Check.

Running CheckDB

Usage

You can run CheckDB from a command line in DOS or UNIX. Before running CheckDB, make sure that the data base is available. It is suggested to run CheckDB exclusively, but you may also run it in Client/Server mode.

CheckDB *ini_file* [-C:*checks*] [-S:*srce*] [-T:*type*] [-R] [-W] [-K:*time*]

The details for the data source are described in the option file.

ini_file

The option file defines the data source for the database. More details on how to define the option file you can find in “Option file for CheckDB”

C:*checks*

The list of check options determines the type of checks to be performed:

I	Inverse reference check
X	Index check
G	GUID check

Default: -C:IXG (running all checks)

The order of these options does not play any role. You may also use capital or small letters except for the option key –C:

-S:*srce*

The source describes a property path to the collection or set of instances to be checked. You may define an extent as

Persons

but also a more complex path as

Persons().accounts

which represents the accounts for all persons.

When no source is defined, the whole database is checked.

Default: -S:* (check the whole database)

-T:*type*	The type defines the source type:	
	C[ollections]	check all collections referenced by the source. Since the collection check makes sense for I and X, only, the GUID check (G) is ignored when being defined together with Type C. When no source is defined, all global collections (extents) are checked. When an extent name (e.g. Persons) is passed as source, only the extent is checked. When a path is passed (e.g. Persons().accounts(), the accounts collection for each person is checked.
	I[nstances]	Check all instances (or the references for all instances referenced in the source. When no source is defined, all instances in the database (G) or the reference collections for all instances (I, X) are checked. When an extent is passed (e.g. Person), all instances stored in the extent are checked (for G) or the reference collections for all instances are checked (for I, X). When a path is passed (e.g. Persons().accounts), all account instances (G) or all reference collections for all account instances (I, X) are checked.
	A[ll]	Check instances and collections.
	Default: -T:all (check instances and collections)	
-K:*time*	When running CheckDB in client/server mode, the database must be available exclusive for CheckDB, i.e. no clients must access the database. The kill option allows stopping clients after a given time interval (time in seconds). When a time interval has been defined, the clients are informed, that the client will be stopped after ‘time’ seconds. When passing the –K option, only, the clients accessing the database to be checked are stopped immediately and without warning. When not passing the kill option, CheckDB will terminate with error when there are active clients using the database to be checked.	

-R

Running CheckDB without repair option, the database is checked for errors, only. Passing the repair option causes CheckDB repairing the problems detected, if possible. The protocol file contains information, whether repairing the problem has been succeeded or not.

-W

For showing warnings in the protocol, the warning option must be set. Otherwise, error are displayed, only.

Output

The output is written to console. For storing the output to a file, you must re-direct the output to a file. When running CheckDB on a server database (in client/server mode), the output is written after CheckDB has been terminated completely. Locally, problems are written to the protocol as soon as an error has been detected.

Samples

You will find sample procedure calls and option files for Windows environment in the ODABA^{NG} installation folder under CheckDB.bat and CheckDB.ini.

Maximum check

The maximum check includes all possible checks for a database and can be called as:

CheckDB *ini_file* -C:IXG -S:* -T:all –R

or, referring to defaults

CheckDB *ini_file* –R

The repair option must be passed to repair problems found.

Checking GUIDs for all persons

CheckDB *ini_file* -C:G -S:Persons

This call will report GUID problems for all person instances but not repair them (because of the missing -R option).

Repair references for persons

CheckDB *ini_file* -C:X -S:Persons -T:Instances

This call will check and repair the reference collections for all person instances, i.e. it will remove dangling pointers from all references in person instances and repair indexes when required. When not passing the Instance type option -T:Instances, the check is performed for the Persons extent, only, by removing dangling person references from the collection.

Checking Inverse references

CheckDB ini_file -C:I -R

This call will inverse references for the complete database. This check is time consuming and takes about 1 minute per 10 MB database.

Option file for CheckDB

An option file defines the data source, input and output files and other process specific parameters.

```
[SYSTEM]        system section

DICTIONARY= C:\ODABA\ODE.SYS

[CheckDB]

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
NET=NO
ENABLE_CONTEXT=NO

ACCESS_MODE=Write
```

When running CheckDB on a server, you need to define the server location (SERVER_URL and SERVER_PORT). In this case you must set NET to YES.

The business rules should always be disabled (ENABLE_CONTEXT=NO), since they might be time consuming and may disturb the repair process.

9 Reset pending key locks (ResetKeyLocks)

The ResetKeyLocks utility allows resetting pending key-locks, which may result from applications terminated abnormally. You may use the ResetKeyLocks also for displaying pending key locks.

Make sure, that all other applications using the database have been closed. Since the function requires exclusive database access, you cannot run ResetKeyLocks from a client on the server or when the database is used by an other application.

Running ResetKeyLocks

You can run ResetKeyLocks from a command line in DOS or UNIX. Before running ResetKeyLocks make sure that the database is available.

ResetKeyLocks *db_path* [-ID:*session_id*] [-L]

ResetKeyLocks will remove all pending key locks or the key locks for the session identifier passed to the function.

db_path

Is the complete path to the database.

You may remove key locks in all types of databases as application databases, resource databases or system databases.

session_id

The owner or session number it a consecutive number created for each database opened with write access. This allows removing key locks created in a specific session. Usually, after closing all database applications, no key locks should be registered and no session number should be passed in order to delete all key locks.

-L

The list option can be set in order to display pending key locks without resetting. .

You may remove key locks in all types of databases as application databases, resource databases or system databases.

10 Database System Information (DBSystemInfo)

DBSystemInfo allows displaying database system information as database versions and basic locations for root objects. Especially database versions allow detecting errors when opening the database with a wrong software version or an invalid replication client.

Running DBSystemInfo

You can run DBSystemInfo from a command line in DOS or UNIX. Before running DBSystemInfo make sure that the database is available.

> **c:\ODABA\DBSystemInfo.exe** *db_name* [*print_path*]

DBSystemInfo provides system information for the passed database in the file defined in the print_path vparameter,

db_name

Is the complete name (path) of database to be displayed.

You may display system information for all types of databases as application databases, resource databases or system databases.

print_path

Is the path and file name for writing the system information.

If print_path is not defined the DBSystemInfo.txt on the current folder will contain the result.

Samples

You will find sample procedure calls in a Windows environment in the ODABA^{NG} installation folder under DBSystemInfo.bat.

11 Database Statistics (DBStatistics)

DBStatistics allows displaying detailed database statistics as used and deleted objects instances, cluster statistics and space requirements for instances and indexes. This can be helpful for detecting the requirement for compressing the database (PackDB).

Running DBStatistics

You can run DBStatistics from a command line in DOS or UNIX. Before running DBStatistics make sure that the database is available.

C:\ODABA\DBStatistics.exe *db_path* [*print_path*]

DBStatistics provides database statistics for the passed database in the file defined in the print_path parameter,

db_path

Is the complete name (path) of database to be displayed.

You may display database statistics for all types of databases as application databases, resource databases or system databases.

print_path

Is the path and file name for writing the database statistics.

If print_path is not defined the DBStatistics.txt file on the current folder will contain the result.

Samples

You will find sample procedure calls in a Windows environment in the ODABA^{NG} installation folder under DBStatistics.bat.

12 Dictionary Statistics (DBDictStatistics)

DBDictStatistics allows displaying dictionary statistics about types defined via the dictionary. It lists all complex data types used for storing data an application database or external medium such as binary or csv files. Beside type name, the tool lists internal type number, current schema version number, indicators for system type, enumeration and persistence as well as internal and external length for instances of the type.

All this can be helpful for detecting errors in structure definitions or schema version mismatch.

Running DBDictStatistics

You can run DBDictStatistics from a command line in DOS or UNIX. Before running DBDictStatistics make sure that the database is available.

C:\ODABA\DBDictStatistics.exe *dict_üath*

[*print_path*]

DBDictStatistics provides dictionary statistics for the passed database in the file defined in the print_path parameter,

dict_path

Is the complete name (path) of dictionary database to be displayed.

Allthough you may display dictionary statistics for all types of databases as application databases, resource databases or system databases, it makes not much sense for application databases, except you hane created runtime type definitions in an application database.

print_path

Is the path and file name for writing the dictionary statistics.

If print_path is not defined the DBDictStatistics.txt file on the current folder will contain the result.

Samples

You will find sample procedure calls in a Windows environment in the ODABA^{NG} installation folder under DBDictStatistics.bat.

13 Version administration (DBVersion)

The version utility provides a number of operations for maintaining database entry versions. This includes creating and removing versions as well as displaying managed versions.

Running DBVersion

One may run DBVersion from a command line in DOS or UNIX. Before running DBVersion, make sure that the database is available.

…/DBVersion.exe *poth* [*operation*] [-**M**:*mode*] [**-V:***version*] [**-T:***time*]

One may perform different operations for the database version list.

path

The path refers to the database location or ini-file. The database must be accessible on the local machine or as file share. In case of encoded database, the ini-file is required in order to provide the access key. The ini-file must contain a `DATABASE` section defining at least database and dictionary path (see example below). When using an external key file, `KEY_FILE` is required in addition.

```
; minimum DATABASE section for an en-
coded
; database
[DATABASE]
DICTIONARY=...dict path...
DATABASE=...database path...
ACCESS_MODE=Write
SHARE=NO
```

operation

One of the operations described below has to be specified as operation to be performed. When no operation is passed, L[ist] is assumed.

M[ode]

Allows setting or changing the versioning mode to the value passed in the mode option. The mode .

DBVersion.exe *db_path* **Mode** [**-M:**vm]

C[reate]	Create new version slice. Usually, creating a new version closes the last version slice by setting the current date and time as termination data. In order to close the last version slice later on, the termination time (or starting time for the new version) has to be passed. **DBVersion.exe** *db_path* **Create** [**-T:***time*]
U[pdate]	Update version slice. In order to update a version slice, version number and When not passing a version number, the last version slice will be removed. In order to remove several version slices at ones, the oldest version number to be removed might be passed as version number. **DBVersion.exe** *db_path* **Update** [**-V:***version*] [**-T:***time*] The operation removes all data stored for the version(s) higher or equal to the version number passed and resets the version number.
L[ist]	In order to list predefined and previous versions, list has to be passed as operation.. **DBVersion.exe** *db_path* **List** [**-V:***version*]
-T:time	In order to update or create version slices, the termination time for the version slice (timestamp with a date or date/time value) might be passed: **-T**(2009-12-01 12:00:00,00) or **-T**:2009-12-01 When the time value contains spaces, **-T**(…) should be used rather than **-T**:… .

-M:mode	Allows setting or updating versioning mode. Since ODABA supports only one version mode in a database, the version mode can be set explicitly or implicitly.

DBVersion.exe *db_path* **-M:***mode*

Versioning modes are combined from settings for three features:

- Versioning levels: Versioning levels support hierarchical or simple versioning
  Managed - major (managed) and minor versions are supported. Major versions are always consistent.
  Default – simple (non hierarchical) versioning
- Versioning consistency: Different levels of consistency may be requested
  Consistent - Database consistency is guaranteed for all versions
  Synchronized - version numbers are synchronized (temporal order)
  Default - each scope defines its own version numbers
  - Versioning scope: The versioning scope defines the scope for consistent versioning

  Local - instance scope
  Individual - owner scope

  Default - database

Versioning modes are set by passing the first letter for each selected feature. One may select none or one feature from each feature category. Features must be passed in the sequence as listed above (e.g. -M:MCI)

When no version mode is set, the version mode is determined automatically with the first versioning request. E.g. when creating a version slice, version slice mode will be activated.

-V:version	The version number refers to the version to be reset or updated. The version number has to be defined as valid version in the version list.

14 Workspace Utility (DBWorkspace)

The workspace utility provides a number of operations for maintaining workspaces. This includes creating, removing workspaces as well as displaying the workspace hierarchy.

Running DBWorkspace

You can run DBWorkspace from a command line in DOS or UNIX. Before running DBWorkspace, make sure that the database is available.

> **DBWorkspace** *ini_file data_source operation ws_name* [*user*] [*options*]

You can perform different operations for the workspaces in a database.

ini_file	The option file defines the data source for the databases and specific parameters. More details on how to define the option file you can find in “Option file for DBWorkspace”
data_source	is the data source name for the source database. The data source must be defined either in the catalog or in a section with the same name in the option file.
operation	One of the operations described below has to be specified as operation to be performed.
ws_name	Is the name of the workspace the operation should be performed for.
user	Workspaces that are owned by users need a username for authorizing the operation. .
options	The options depend on the operation performed.

Enable

Running a database with workspaces required enabling the database for workspace processes. When enabling the database a shadow database will be created and a system workspace (WS0) is allocated. WS0 cannot be referenced explicitly as a workspace but is the base for all other workspaces.

DBWorkspace *ini_file data_source* **Enable** [-R:*path*]

The shadow database is copied from the original database to the location defined in *path*. When no location is passed the shadow database will be created in the same location as the original database with the extension **.sdw**. The system workspace is located in the same location as the original database with the extension **.ws0**.

Disable

This operation disables the workspace feature for a database. The current state of the shadow database is copied to the original database, which practically means that all open workspaces are consolidated. All workspaces including WS0 are deleted.

DBWorkspace *ini_file data_source* **Disable**

After disabling workspaces you cannot use workspaces until this feature is enabled again.

Create

Creates a new workspace. When passing a hierarchical workspace name where the workspaces on different levels are separated by '.' (e.g. "wspace.subspace") all workspaces that do not exist in the hierarchy are created.

DBWorkspace *ini_file data_source*
Create [-W:*ws_name* [-U:*user*]]

For creating a user specific workspace a user name has to be passed. The workspace name (ws_name) is a workspace within the workspace defined in the data source (**WORKSPACE**). When not passing a workspace name the workspace defined in the option file is created.

Delete

Deletes an existing workspace. An existing workspace can be deleted only when it is the lowest in the hierarchy, i.e. when the workspace does not have subspaces. Moreover, the workspace must be empty. When the workspace is not empty use the Discard or Consolidate operation to clear the workspace.

When passing a hierarchical workspace name where the workspaces on different levels are separated by '.' (e.g. "wspace.subspace") only the last workspace in the hierarchy is deleted.

DBWorkspace *ini_file data_source*
Delete [-W:*ws_name* [-U:*user*]]

When a user owns the workspace the user name must be passed for deleting the workspace.

Consolidate

Consolidates an existing workspace. Consolidating a workspace will store all updates made in the workspace on the higher workspace and empty the workspace.

When passing a hierarchical workspace name where the workspaces on different levels are separated by '.' (e.g. "wspace.subspace") only the last workspace in the hierarchy is consolidated.

DBWorkspace *ini_file data_source*
Consolidate [-W:*ws_name* [-U:*user*]]

When a user owns the workspace the user name must be passed for consolidating the workspace. The workspace name (ws_name) is a workspace within the workspace defined in the data source (**WORKSPACE**). When not passing a workspace name the workspace defined in the option file will be consolidated.

Discard

Discards an existing workspace. Discarding a workspace will undo all updates made in the workspace and empty the workspace. A workspace can be discarded only when it is the lowest in the hierarchy, i.e. when the workspace does not have sub-spaces that contain data.

When passing a hierarchical workspace name where the workspaces on different levels are separated by '.' (e.g. "wspace.subspace") only the last workspace in the hierarchy is discarded.

> **DBWorkspace** *ini_file data_source*
>
> **Discard** [-W:*ws_name* [-U:*user*]]

When a user owns the workspace the user name must be passed for discarding the workspace. The workspace name (ws_name) is a workspace within the workspace defined in the data source (**WORKSPACE**). When not passing a workspace name the workspace defined in the option file is discarded.

List

Lists the workspaces that exist below the passed workspace. For displaying the workspaces for the database (top workspaces) * can be passed as workspace name.

> **DBWorkspace** *ini_file data_source*
>
> **List** [-W:*ws_name* [-U:*user*]]] [-T]

The workspace name (ws_name) is a workspace within the workspace defined in the data source (**WORKSPACE**). When not passing a workspace name, workspaces for the workspace defined in the option file are listed. For displaying the workspaces for a specific user the user name can be passed. When passing '*' as user name or no username all workspaces below the passed workspace are displayed.

For displaying the complete workspace tree (hierarchy) you can add the -T option.

Defining data sources

The workspace utility allows maintaining workspace hierarchies for a database, which is defined as data source in an option file. There are two ways to refer to a data source. One way is to define the data source in the application option file. In this case the data source is defined in a section that is preceded by the data source name:

{DataSource1] Data source name

The other way is to refer to a data source defined in a database catalog by its data source name:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "ODABA^{NG} – Server" documentation.

Database and dictionary

A dictionary and a database define a data source. While the dictionary contains the data definitions the database contains the data. Dictionary and database can be stored in the same database file but usually they are not. In any case, each data source definition should contain a dictionary definition and a database definition:

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat

A data source can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=ProjectServer
SERVER_PORT=6123

DICTIONARY=%SAMPLE_DICT%
DATABASE==%SAMPLE_DAT%

WORKSPACE	The workspace variable defines the basic workspace to be used in the application. The application can open workspaces on top of the base workspace but not below. The defined workspace is either the base for the operation (when passing a workspace name) or the workspace for running the operation (when no workspace name has been passed. A workspace can be defined only when the workspace feature is enabled. When not defining a workspace the database is opened directly. Default: none
ACCESS_MODE	The access mode defines whether the database will be used in write/update mode or read only. The workspace utility requires write access except for the list operation. **ACCESS_MODE**= Read **|** Write Default: Read (for from-data source), Write (for to-data source)
NET	This option is required when running the database in a file server or client/server environment for using the database with more than one user (multi-user access). **NET**=YES **|** NO This feature is supported under Windows, only. Under Linux, YES is used, always. Default (Windows): NO

Option file for DBWorkspace

ODABA^NG^

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[DBWorkspace]
DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat
WORKSPACE=Basic_WS

NET=NO

When referring to data sources defined in the data catalog on the server you need not to define the data source sections for source and target data source.

15 ODABA Script Interface (OSI)

The ODABA Script Interface supports data definitions and methods (expressions). OSI can be used for running ad hoc queries or for updating the database.

An OSI script may include local functions and class or structure definitions, bit it may also refer to resources defined in the dictionary.

Running OSI

Usage

You can run OSI from a command line in DOS or UNIX.

```
OSI          script_file ini_file
             [ -SP:parameters ]
             [ -E:entry_point ] [-DB]
```

Besides the script file, OSI requires at least a dictionary, which can be defined explicitly or in the option file.

script_file

The script file refers to a location where the ODABA Script file is stored. When not passing an option file or dictionary path, the dictionary location must be defined in the script file.

ini_file

Instead of a dictionary path, an option file can be passed. The When containing a dictionary location, this will replace the dictionary location defined in the script file. When containing a database location, the data base location in the option file is replaced as well.

Besides database and dictionary, the option file may contain definitions of system variables, which can be referenced in the script.

parameters

Any number of script parameters separated by comma might be passed to the script file. When the list contains spaces, the option must be enclosed in “”:

“-SP:parm1, message text, parm3”

Parameters are passed in the same sequence to the OSI-function referenced as entry point.

entry_point

The entry point is the name of the OSI function that is called. When no entry point is defined, “main” is supposed to be the entry point.

Entry point names are case sensitive, i.e. when not defining an entry point, the script file must contain an OSI function with the name ‘main’.

-DB

In order to run OSI scripts in debug mode, the debug option has to be passed.

Samples

You will find sample scripts and option files in the ODABA^{NG} installation folder under OSI.bat, OSI.ini and Sample*nn*.osi.

Defining data sources

OSI requires a database defined in a data source. The data source definition includes at least a dictionary, but usually it consists of dictionary definition and database path.

There are different ways of providing data source definitions. Typically, the data source is defined in the script file or in an option file passed to OSI. In both cases, there are two ways to refer to a data source. One way is to define the data source implicitly by defining dictionary and database in the script or in the OSI section of the ini file.

The other way is defining the data source explicitly in a separate section of the ini file. In this case, the data source is defined in a section that is preceded by the data source name:

[DataSource1] Data source name

Now, the data source can be referred to by its data source name as:

DATA_SOURCE=DataSource1

This way, it is also possible to refer to data sources defined in the database catalog. When referring to a database catalog, the option file must contain a catalog section, that defines the location of the data catalog:

[DATA-CATALOG]

Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "ODABA^{NG} – Server" documentation.

Multiple data sources

Functions within an OSI script may refer to multiple data sources. When referring to more than one data source, the option file should define the required data sources.

The OSI script may refer directly to dictionary and database paths. In this case, the script becomes dependent on the data location, which can be avoided by referring to data sources in an option file.

Option file for OSI

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters. The following example refers to the specification of the sample database source based on an ODABA^{NG} database.

```
[SYSTEM]        system section

DICTIONARY= C:\ODABA\ODE.SYS

[OSI]

DICTIONARY=C:\ODABA\ Sample\Sample.dev
DATABASE= C:\ODABA\Sample\Sample.dat
NET=YES
ACCESS_PATH =Company.employee

[DATA-CATALOG]

DICTIONARY=C:\ODABA\ode.sys
DATABASE= C:\ODABA\Sample\catalog.dat
NET=YES
ACCESS_MODE=Write
```

16 ODABA Definition Loader (ODL)

The ODABA Definition Loader is used for loading OSI model definitions. Syntactically, an ODL database model is based on the OSI script interface language, which is an extension of ODMG ODL (2003). In contrast to the OSI utility, ODL does not run queries, but loads a schema (object and functional model) to an ODABA resource database (dictionary).

With ODL you may create a new resource database or update an existing one.

Running ODL

Usage

You can run ODL from a command line in DOS or UNIX.

ODL [:*ini_file* | *dict_path*] *script_file* [-R|-N] [-C] [-S] [-A]

Besides the script file, ODL requires a dictionary, which can be defined explicitly, in the option file or in the script.

script_file

The script file refers to a location where the ODL Script file is stored. When not passing an option file or dictionary path, the dictionary location must be defined in the script file.

dict_path

The dictionary path provides the location for the dictionary. It becomes necessary, when no dictionary is defined in the script file. When passing the dictionary location, the dictionary location in the script file will be ignored.

ini_file

Instead of a dictionary path, an option file can be passed. The containing a dictionary location, this will replace the dictionary location defined in the script file.

Options

In order to maintain dictionary state and complex data type states properly, several options might be passed. When not passing any of those options, the schema loader fails, when the dictionary is in production state. When this is not the case, no checks will be performed and affected data types remain unchecked and not ready for use.

-R|-N

When the dictionary is in production state, which becomes necessary in order to store data in a database, the schema cannot be updated. In order to update the schema, the production state might be reset by passing –R or a new schema version might be created by passing –N.

-C

After importing schema updates, affected complex data types (structures) are not considered as checked and ready. This will cause errors when trying to access data in a database. In order to prepare structures for database access, the check option (-C) might be passed, which causes ODL to check the schema, update schema versions and sets structures to checked and ready.

-S	After checking the database schema successfully, complex data types are marked as checked and ready, but the dictionary is not yet in production state. In order to run a production system with the dictionary, the production state should be set, which prevents the dictionary from critical schema updates. For setting the production state, this option might be passed.
-A	Passing this option causes the schema loader to prompt the user for executing actions, which have not been explicitly requested by corresponding options.

Samples You will find sample scripts in the …/ODABA/Sample folder.

17 ODABA Shell (OShell)

OShell is a command line utility that allows running most of the ODABA access functions from a command line. In contrast to OSI, OShell is not a query tool, but a way to navigate through a database as you may navigate through the directory structure on a disk.

OShell needs a data catalog that contains all data sources to be accessed from the shell. The data catalog can simply be provided in the option file passed as parameter to OShell or in an ODABA data catalog, which is an ODABA database that holds available data source definitions.

Running OShell provides access to all data sources defined in the data catalog.

You can use OShell for testing or examine your databases but also for maintenance purposes or within the production process. OShell provides reading and writing access to the database and allows you to do nearly every thing that you could do within a MS Visual Basic or C++ program.

Running OShell

Usage	You can run OShell from a command line in DOS or UNIX. Before running OShell make sure that a data catalog is available.
	OShell *ini_file* [*script_file* [E:*entry_point*]] [-DB]
	When not passing a procedure name OShell will request commands from the console.
ini_file	The option file defines the data sources for the databases and specific OShell parameters. More details on how to define the option file you can find in "Option file for OShell".
script_file	You may pass the location of an OShell procedure to be called when starting OShell. The procedure is a path name that refers to a OShell script file.
entry_point	In order to run a specific part of the procedure passed in *script_file*, an entry point might be passed.
-DB	In order to run imbedded OSI scripts in debug mode, the debug option has to be passed.
Samples	You will find sample procedure calls and option files for Windows environment in the ODABA^{NG} installation folder under OShell.bat, OShell.ini and OShell.prc (procedure).

OShell Command Overview

You may run OShell from a command line but also as batch procedure. When running OShell it one or more contexts might be opened that can be accesses independent on each other, i.e. you may run OShell several times with several data sources simultaneously in one process.

Command syntax

Command lines consist of a command of function name, a parameter list and additional options. The complete syntax is described below:

```
CMDLine                 := name [parameter(*)] [option(*)] [redir]
redir                   := '>>' file_name
file_name               := string | file_path
file_path               := [ drive ] folders
drive                   := name ':'
folders                 := [ fsep ] fname [ fext(*) ]
fext                    := fsep name
fsep                    := '/' | '\\'
fname                   := name [ fnamext(*) ]
fnamext                 := '.' name
parameter               := value [connector]
connector               := '=' | ':=' | ':'
value                   := path | constant | '*' | '&'
option                  := '-' name
path                                    :=  [navi]  [path_element]
[path_extension(*)]
navi                    := '/' | dot(*)
dot                     := '.'
path_extension          := '.' path_element
path_element            := path_operand [ operand_list ]
path_operand            := path_name | coll_operand
coll_operand            := '[' path ']'
path_name               := ['@'] name [ name_index ]
name_index              := '[' value ']'
operand_list            := '(' [parm_list] ')'
parm_list               := value [parm_ext(*)]
parm_ext                := ',' value
```

Comments

Command script files may contain comments. Lines starting with // are considered as comment lines. Comments may also be appended at end of command line.

Command name

As command names one of the supported command might be used. Also, property handle functions might be called as soon as a data collection has been opened (cc).

Besides, some block commands (**begin**, **do**) are provided in order to enter multiple line commands.

Command names may include option variables enclosed in %...%, which will be replaced by current settings before analysing the command.

Command names are not case sensitive, but function names are.

parameters

Up to 32 parameters can be passed to a command. Parameter values may include option variables enclosed in %...%, which will be replaced by current settings before analysing the command. Parameters containing non-alphanumeric values have to be quoted ("..." or '...').

Parameters do not require separators. Some commands, however, use properties and values, in which case parameters might be connected by connectors:

set OPTION=abc (same as: set OPTION abc)

In fact, the command above gets two parameters (OPTION and abc), which can be passed using a connector in order to increase readability of the script.

options

Options are introduces by – (e.g. –D1). The options supported depend on the command called. As well as parameters and command name, options may contain option values which will be replaced before analysing the command.

Command context

When running OShell the shell runs in one or more contexts defined by the opened data sources and collections.

When starting OShell it enters the command context, which is indicated by the

```
ODABA>
```

command prompt. In this context you may list available data sources or open a data source. The commands available in this context you will get when enter 'help'.

Strings and numeric values

Parameters passed to a command can be passed as numerical value or string (name, expression). When a string parameter contains spaces or other non-alphanumeric characters, it must be enclosed in “” as:

sf “name > ‘a’ and name < ‘z’”

Numerical values must be passed just as a number. String values should be enclosed in single quotes ‘’ as

loc ‘Miller|Paul’

Values which are not enclosed in ‘’ are interpreted depending on the context. Within expressions they are considered as object variables. When being passed as parameter as in

loc Miller|Paul

it is considered as string value, since it does not start with a numeric character.

Data source context

After opening a data source (‘cd’ command) the shell changes into the data source context, which is indicated by the data source name in the command prompt:

```
ODABA>cd SampleDB
SampleDB>
```

In this context you may list available collections or open a collection. The commands available in this context you will get when enter ‘help’.

You may close a data source context using the change data source (cd) command again:

```
SampleDB>cd .
ODABA>
```

Collection state

After opening a collection (cc command) the shell changes into the collection context, which is indicated by the data source name appended by the collection path in the command prompt:

```
SampleDB>cc Person
SampleDB/Person>
```

In this state you may run collection specific commands as create instance (crt) or copy (cpy) as well as functions supported by OShell.

When OShell is in a data source or collection context, you can open another data source context without closing the one currently active.

Multiple data source context

When a data source context is opened you may create another data source context using the ‘cd’ command again.

```
SampleDB >cd SampleDict
SampleDict>
```

In this case the SampleDB context remains and the SampleDict context is opened in addition. The shell switches to the new context. The available data source contexts can be listed using the ‘cd’ command again:

```
SampleDict>cd
1: SampleDB
2: SampleDict
```

The numbers listed in front of the data source name can be used to redirect command to an inactive data source or to refer to an inactive data source e.g. in a copy (cpy) command for the –Dn option.

You may switch to any opened data source context using the context number:

```
SampleDict>cd 1
SampleDB>
```

Collection hierarchy

When a collection has been opened for a data source you may open another subordinated collection (e.g. children for Person)

```
SampleDB>cc Person
SampleDB/Person>cc children
SampleDB/./children>
```

This will activate the children collection for an activated person. Each collection in the hierarchy gets a context number, which can be displayed using the ‘cc’ command again:

```
SampleDB/./children>cc
-0 - Person
*1 - children
```

The numbers listed in front of the collection name can be used to redirect command to an inactive collection in the hierarchy or to refer to an inactive collection e.g. in a copy (cpy) command.

The ‘*’ indicates the active data collection in the hierarchy. You may refer to inactive collections in most of the commands, e.g. for locating an instance in the Person collection:

```
SampleDB/./children>loc 0 –C0
SampleDB/./children>cc
+0 - Person
*1 - children
```

'-' or '+' in front of the context number indicates whether an instance is selected for the collection (+) or not (-).

You may also switch to another collection in the hierarchy using the cc command with the context number:

```
SampleDB/./children>cc 0
SampleDB/Person>cc
*0 - Person
-1 – children
SampleDB/Person>cc/children
SampleDB/./children>
```

Switching between collections will not close collections in the hierarchy. You may close a collection using the change collection (cc) command passing one or more dots (.):

```
SampleDB/./children>cc .
```

The command will close as many collections in the hierarchy as dots have been passed in the command. Thus you may close any number of collections in a hierarchy.

The change collection command refers always to the last opened collection context, i.e. independent on the collection selected as active collection the command will close or open collections relatively to the last collection opened.

Commands

OShell allows running a number of pre-defined commands as well as most of the ODABA access functions (odaba API). OShell commands are a number of predefined statements that allow you to connect to data sources and collections.

All commands are not case sensitive, i.e. you may enter command names in capital letters or not. Command parameters, however, are usually case sensitive, except you refer to command names as parameter (as for the help command).

When typing an empty command (enter), the last command entered is called again.

Parameters

Commands can be called with different parameters and options. Parameters passed with the command must be passed in a defined sequence. Parameters are usually case sensitive and must not contain blanks. Parameters containing blanks (e.g. expressions for conditions) must be enclosed in quotes (“parameter with blanks”).

In some cases commands differ between parameter types. Thus numbers are usually interpreted as numerical values. Thus, the locate command (loc) works differently when being called with 0 or ‘0’ as parameter::

```
SampleDB>cc Person
SampleDB/Person>loc 0
SampleDB/Person>loc ‘0’
```

When calling the locate command with 0 it will locate the first instance in the collection. When calling it with ‘0’ it tries to locate an instance with the key value 0, which is usually different from the first instance.

Numerical parameters

Any parameter beginning with a numerical value is interpreted as integer number. For passing negative numerical values the minus (-) must be appended to the value.

String parameters

Parameters beginning with ‘ are interpreted as strings. String parameter, which contain ‘-characters again, must duplicate the ‘_sign.

Context variables

Any parameter beginning with a non-numerical character and not with – or ‘ is considered as context variable. The value of a context variable is determined as follows:

1. When the run state is a collection state and the variable is a structure variable defined in the structure of the active collection, the parameter value is taken from the selected instance.
2. Otherwise, it is checked, whether the variable is a defined system variable (e.g. set by the ‘set’ command). In this case the parameter value is taken from the current value of the system variable.
3. When neither (1) nor (2) return a variable value, the variable is considered as string variable, i.e. the parameter value is the string passed.

Options

Options can be passed in any order and at any position

	after the command name. Options are preceded by a ‘-‘ character (as –D0).
Common OShell commands	Common commands may be called without an opened data source.

Call procedure file	call file_name[@entry_point]
Change data sources	cd [dsname|*|dsid|.] -Dn
Run command block	begin | do (also in connection with fa, if, while)
Leave block	LeaveBlock | lb
Do when true	if expression command -Dn - Cn
Do while true	while expression command -Dn -Cn -In
Terminate block	end
Return from procedure	return
Exit application	exit
Do for all instances	fa command -Dn -Cn -In
List data sources	ld
Load procedure file	load fname|file_name
Pause processing	pause
Terminate OShell	quit | q
Redirect output	redir [path]
Set variable	set var_name [value] | set [var_name=value]
Show help	help [command] -d -a -e
Show commds	echo [on|off|'any text']
Display formatted data	format | f fstring [parm ... parm] -Dn -Cn

Data source commands	Data source commands are provided in order to manage data sources under OShell

Change collection	cc prop_path|coll_id|*|& -Dn
Run database action	databaseAction name [parm1...parmN] -Dn
List collection names	lcn [mask] -Dn -Cn
Run object space action	objectSpaceAction name [parm1...parmN] -Dn
run OSI function	osi funct_name

run OSI statements	osi do
	   expression statements
	end

Collection commands	Collection commands are provided in order to manage collections under OShell

Change sort order	co key_name [gen_attr_val]-Dn -Cn
Copy instance	cpy keyval | position | . | * [new_key] -Dn -Cn
Create instance	crt [keyval] -Dn -Cn
Delete instance	del keyval | pos | . | * -Dn -Cn del keyval | pos -E
Run instance action	instanceAction name [parm1...parmN] -Dn -Cn
List attribute names	lan [mask] -Dn -Cn
List instances	li [p[osition]] -Dn -Cn -In

List keys	lk -Dn -Cn
List sort orders	lo -Dn -Cn
Locate instance	loc [key_value|pos] -S -Dn -Cn
Move instance	mov keyval|position|.|* [new_key] -Dn -Cn
Position foreward	next [count] -S -Dn -Cn
Position backward	prev [count] -S -Dn -Cn
Show attribute values	print|p [varname | *] -Dn -Cn
Run property action	propertyAction name [parm1...parmN] -Dn -Cn
Set attribute list	sal attrname1 ... attrnameN -A -Dn -Cn
Set attribute value	sav attrname [=] value -Dn -Cn -Q
Set filter condition	SetFilter | sf [condition] -Dn -Cn

Debug commands — When running OShell with debug option (-DB), several debug commands are available

List call stack	backtrace | bt [count]
Set break point	break | b [[class::]fname] [line] [proc_name|]
Break at each statement	breakAlways | ba
Watch variable	watch|w
Delete watch variable	deletewatch|dw
Continue application	continue | c
Reset break point	disable | d [[class::]fname] [line_number]
Leave function	finish | fi
Change stack frame	frame | f [number]
Go to line	jump | j [number]
Skip statement	jumpOver | o
List current function	list | l [[class::]fname]
List current line	listCurrent | lc
Load procedure file	load [class::]fname | file_name
Execute next statement	n
Show attribute values	print | p [varname | *] -Dn -Cn
Terminate debugging	quit | q
Run application	run | r
Change stack limit	stackLimit | sl [number]
Step into function	step | s

General command options — Most commands support two additional options for identifying the data source or re-directing the output.

Output re-direction — The output from a command can be re-directed to a file or stream. Output re-direction is passed as last option in a command line.

```
ODABA/Sample/Person>li >>c:/temp/list.txt
```

This directs the output of the list command to the file list.txt. Redirection options will not delete or empty the file, when it does already exist and contains data.

For redirecting the output permanently, you may use the "redir" command.

```
ODABA/Sample/Person>redir c:/temp/list.txt
```

In this case the output is redirected to the file defined in the redirection path until you define another permanent redirection or terminate the redirection with

```
ODABA/Sample/Person>redir
```

Permanent redirection of the output is also reset at the end of a do-block, when being specified within the block or at the end of a called procedure, when being activated within a procedure.

Data source

By default the data source is defined by the selected database context, which is displayed in the promt line. Most commands, however, allow referring to another opened data context by the data source (-D) and the collection (-C) option.

The data source option –Dn refers to the data source, which is shown as data source n with the "cd" command. The collection option –Cn refers to the collection in the referenced data source context, which is listed as collection number n with the "cc" command. When passing the collection option without a data source option (-Cn, only), a collection in the current collection hierarchy can be referred to.

Procedures

Moreover, you may define batch files, which can be called from the command line or passed as parameter.

Within a batch file you may define any number of sub-procedures. Sub-procedures can be called from within the batch file but also from other batch files.

Besides commands acting directly on data sources or collections OShell supports a number of meta-commands. Meta commands allow controlling the processing in a procedure (batch file).

call

To invoke a procedure or sub procedure you can use the call command.

```
ODABA>call c:\ODABA\sample.prc
```

For running the procedure from a certain entry point in the procedure you may append the entry point name to the file name:

```
ODABA>call c:\ODABA\sample.prc@ListPersons
```

load

For running a procedure several times you may load procedures to the procedure cache. In this case the procedure is read once and removed from the storage when closing OShell.

```
ODABA>load c:\ODABA\sample.prc
ODABA>call @ListPersons
```

When calling a loaded procedure you must always use the entry point character ‘@’ with the entry point name.

Comments

You may insert comment lines at any point in the procedure. Comment lines must begin with one of the following characters ! * / # ; or ‘ in the first position of a command line. Comments are not executed but displayed when echo is on.

echo

The echo command allows controlling whether loaded procedure statements or entered statements will be displayed on the console or not.

Entry points

Entry points in a procedure can be used to run statements in a procedure from a certain entry point. Entry point names are preceded by an entry point character ‘@’ and must begin in the first column of a command line.

```
...
@Sample1
  cd SampleDB
  cc Person
  return
```

Now you may call the procedure with entry point sample1 (e.g. after loading the procedure with the load command):

```
ODABA>call @Sample1
SampleDB/Person>
```

The result in this case is the opened Person collection. Note, that entry point names are case sensitive.

return

When calling a sequence of statements within a procedure, it should be terminated with a return command, which causes the call-command to terminate the processing. No return is required, when the procedure or sub-procedure terminates at end of file.

begin, do, end

OShell supports two block commands, which allow defining blocks of statements within a procedure.

```
@Sample_do
do
  cd SampleDB
  cc Person
end
return
```

The do-block loads the current state on a stack, and reloads it when leaving the block. Thus, changes of the run state made in the procedure are not visible when leaving the procedure. Calling this entry point would return the following state:

```
ODABA>call @Sample1
ODABA>
```

In this example, no data collection is opened after leaving the do block, since the previous state has been reloaded from the stack. Using a begin-block instead:

```
@aSmple1
begin
  cd SampleDB
  cc Person
end
return
```

And calling it from the command line, will result in an opened data source and collection instead.

```
ODABA>call @Sample1
SampleDB/Person>
```

leave

You may leave a do- or begin-block at any point using the ‘leave’ command. The ‘leave’ command will leave the current block only and continue with the command after the ‘end’ command. For leaving the called procedure you may use the ‘return’ command.

For all (fa)

The ‘for all’ command (fa) allows running a single command or a command block for all instances in a collection.

```
@aSmple_fa0
begin
  cd SampleDB
  cc Person
  fa lav name
end
return
```

This procedure will list the name attribute value (lav) for each person in the person collection. If you want to run more than one command for each instance in the collection, you can define a ‘do’ or ‘begin’ block instead.

```
@aSmple_fa01
begin
  cd SampleDB
  cc Person
  fa begin
    lav name
    lav first_name
  end
end
return
```

‘do’ blocks should be used only, when you change the run state in the block, since do blocks will reduce the performance always. When calling a ‘do’ block for an ‘fa’ command will create only one stack for the complete loop. If you want to restore the run-state for each instance you need to define an inner fo block:

```
@Sample_fa1
begin
  cd SampleDB
  cc Person
  fa begin
    do
      cc children
      fa begin
        lav name
        lav first_name
      end
    end
  end
end
return
```

This example will list the names and first names for all children of all persons. When not using the inner ‘do’ block, in which the collection context is changed, you must close the children collection before leaving the loop:

```
@Sample_fa2
begin
  cd SampleDB
  cc Person
  fa begin
    cc children
    fa begin
      lav name
      lav first_name
    end
    cc .
  end
end
return
```

Since no stack is provided in this example, the procedure must close the collection before opening it in the next iteration step. Since opening collections is rather time consuming, the best way for defining nested loops is using the collection context:

```
@Sample_fa1
begin
  cd SampleDB
  cc Person
  cc children
  cc 0
  fa begin
    cc 1
    fa begin
      lav name
      lav first_name
    end
  end
end
return
```

if

The ‘if’ command enables conditional processing of statements in a procedure. The condition passed to an ‘if’ command is either an expression or the variable ‘success’. The ‘success’ variable contains the success value from the last command executed, which is usually true (success) or false in case of an error.

```
@Sample_if1
begin
  cd SampleDB
  if success cc Person
end
return
```

You may also check the ‘error’ variable, which is the opposite of the success variable (true in case of an error and false otherwise).

Instead of ‘success’ or ‘error’ you may define an ODABA OQL expression as condition. This is possible, however, only, when the run state is a collection state.

```
@Sample_if2
begin
  cd SampleDB
  if success cc Person
  loc 0
  if "children.GetCount > 0" cc children
end
return
```

In this case the children collection is collected only, when the first person in the collection has children.

Expressions must be enclosed in quotes (“”), since the syntax within the expression is unknown to OShell. You cannot refer to success or error variables in an expression, since those variables are special OShell variables and not known in the OQL environment.

As well as for the ‘for all’ command (fa) a ‘do’ or ‘begin’ block can be defined for the if command, when a number of commands should be processed conditionally.

```
@Sample_if3
begin
  cd SampleDB
  if success cc Person
  loc 0
  if "children.GetCount > 1" do
    cc children
    fa lav name
  end
end
return
```

This example will list the names for all children for persons that have more than one child.

while

The ‘while’ command allows defining conditional loops. A command or block of commands is executed as long as the condition in the while command is true.

```
@Sample_while1
begin
  cd SampleDB
  cc Person
  loc 5
  while success begin
    next
    lav name
  end
end
return
```

This loop shows the name for all persons except the first five ones. Since next returns success false after the last instance the loop can be terminated by means of the success value. Practically the while command covers the 'for all' (fa) command, but in many cases the 'fa' command is more comfortable.

Instead of the success or error value the condition can be expressed in terns of ODABA OQL expressions.

osi

The osi-command allows invoking an OSI query definition. OQL queries can be invoked as simple commands but also in an begin/do block. An OSI query differs from a property path in its syntax, which is compliant to the OMG standards.

Passing a simple query can be specified as follows

```
@Sample_oql1
oql from(Person) select(name,street)
return
```

More complex OQL statements can be defined in an oql-block, which is either a do- or a begin-block

```
@Sample_oql2
oql begin
  select(Name = name,
         char(100) Address = street+number)
  from(Person)
  where(name >= ’B’ and name < ’C’
  ToFile(e:\result.csv)
end
return
```

An OQL statement without an output clause (ToFile) creates a result collection, on which further collection commands can be applied.

Similar to the cc-command, the oql-command creates a subordinated collection relative to the current data source context. You may close the collection with

```
cc .
```

Otherwise, the collection is closed automatically at the end of the block, when the block is a do-block. It remains open after the end of a begin-block.

When using the ToFile clause, the collection is an elementary value, the number of selected instances, which is returned by the ToFile-operation.

Queries can also be expressed as property or operation paths:

```
cc "from(…).select(…).where(…).ToFile(…)"
```

More details for the OQL specification rules you may find in the "ODABA Query Language".

Functions

Many functions from the **odaba** interface access classes can be called directly from the command line. In contrast to command names, function names are always case sensitive.

When passing parameters to functions, those may refer to constant values or properties. I most cases, you cannot pass expressions or access path as property parameters. This limits the use of function calls on the command line.

When calling a function directly from the command line becomes impossible because of these restrictions, one bay always use an **osi do** … **end** block. Within **do** and **end**, command line limitations do not exist.

Property

Property functions can be called when the shell is in the collection state, only.

Database

Database and object space functions can be called as soon as a data source has been opened.

Defining data sources

OShell works on a number of data sources defined in a data catalog. The data catalog is either part of the option file, where each data source is described in a separate section beginning with the data source name or an ODABA database catalog, which is an ODABA database that contains data source definitions. You may either refer to a data catalog or to data sources defined in the option file, but not both.

Data catalog

Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "Option files" documentation.

Option file

Defining data sources in the option file instead of a data catalog requires a DATA_SOURCES variable in the [OShell] application section that lists all data source names separated by comma:

DATA_SOURCES=DataSource1,DS2,….

For each data source name in the list an appropriate section with the same name must be defined. The name list provides the data source names listed when using the 'ld' command. You may access data sources defined as section but not listed in this list by using the 'cd' command, but they are not displayed when listing all available data sources.

Option file for OShell

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters. The following example refers to the specification of the sample database source based on an ODABA database.

[SYSTEM] system section

DICTIONARY=C:\ODABA\ODE.SYS
ODABA_PATH=C\ODABA

PROGPATH=C\ODABA

[OSHELL]

[DATA_CATALOG]

DICTIONARY=C:\ODABA\ODE.SYS
DATABASE= C:\ODABA\Sample\Server.Cat

NET=YES

When referring to data sources defined in the data catalog on the server you need not to define the data source sections for source and target data source.

18 Simple stress test (StressTestS)

StressTestS allows testing the performance in a local or client/server environment. The simple stress tests is usually reading instances from an extent or any other collection, which can be defined as an access or operation path.

The test may run in single or multiple thread environment.

Running the test will read the instances from the collection one by one until the number of instances to be read has been exceeded.

The test program allows starting a number of concurrent threads, which are sharing the connection to the server. In order to run multiple connection tests, several instances of StressTestS might be started.

In order to perform more sophisticated tests, you may call StressTestM, which allows running multiple query tests.

Running StressTestS

Usage	You can run StressTestS from a command line in DOS or UNIX. Before running StressTestS, make sure that the data source is available. **StressTestS** *ini_file query_path* [**-D:***distance* [**-C:***clients* [**-O:***objects*]]]
ini_file	The option file defines the data sources for the data-bases and specific test parameters. More details on how to define the option file you can find in "Option file for StressTestS"
query_path	The *query* path (access path) defines the collection to be processed in the performance test. Usually, the access path refers to an extent. Instead of an extent you may define an access or operation path that refers to more complex queries. Since the performance depends much on the instance type (i.e. on the number of shared base structures and generic attributes).
distance	is the time in 1/10 seconds between starting two sub-sequent queries. Default: 10 (1 second)
clients	is the number of clients that are running in the test. Default: 1000
objects	is the maximum number of object instances to be read per client (only when running with *extends*). Default: 100

Defining data sources

StressTestS works on data base defined in a data source in an option file. You can define the data source directly in the option file or refer to an existing data source in the data catalog. The section in the option file describing the data source must start with

[StressTestS]

For referring to a data source by its data source name you must define the data source reference in the option file as:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the "ODABA^{NG} – Server" documentation.

Database and dictionary

A dictionary and a database define a data source. While the dictionary contains the data definitions the database contains the data. Dictionary and database can be stored in the same database file but usually they are not. In any case, each data source definition should contain a dictionary definition and a database definition:

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample/Data\Sample.dat

Data sources can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=ProjectServer
SERVER_PORT=6123

DICTIONARY=%SAMPLE_DICT%
DATABASE=**=**%SAMPLE_DAT%

Options

Some options that can be defined in a data source are:

VERSION

Internal database version number when the database is using version features. The version number allows copying the resource database according to a historical state.

VERSION=version

	Default: current version
ACCESS_MODE	The access mode defines whether the database will be used in write/update mode or read only. When updating data in the expression the from-data source must be defined with Write option. **ACCESS_MODE**= Read | Write Default: Read (for from-data source), Write (fot to-data source)
NET	This option is required when running the database in a file server or client/server environment for using the database with more than one user (multi-user access). **NET**=YES | NO This feature is supported under Windows, only. Under Linux, YES is used, always. Default (Windows): NO

Option file for StressTestS

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters. The following example refers to the specification of the sample database source based on an ODABA^{NG} database.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[StressTestS]

DICTIONARY=C:\ODABA\ Sample\Sample.dev
DATABASE= C:\ODABA\Sample\Sample.dat

NET=YES

When referring to data sources defined in the data catalog on the server you need not to define the data source sections.

19 Multiple Query Stress Test (StressTestM)

StressTestM allows testing the performance in a local or client/server environment. The multiple query stress tests is usually referring to one or more queries (OSI expressions) defined in the resource database or in a folder.

Running the test will process the defined queries as many times as requested in the repetition parameter.

The test program allows starting a number of concurrent threads, which are sharing the connection to the server. Providing multiple queries in a folder will assign one query to each thread. When there are more threads than queries, one query might be assigned to several threads.

In order to run multiple connection tests, several instances of StressTestS might be started.

Running StressTestM

Usage	You can run StressTestM from a command line in DOS or UNIX. Before running StressTestS, make sure that the data source is available.

```
StressTestM ini_file query_path
            [-D:distance [-C:clients [-R:repetitions] ] ]
```

The content of the option file for StressTestM is described in the following section.

ini_file	The option file defines the data sources for the databases and specific test parameters. More details on how to define the option file you can find in “Option file for StressTestM”
query_path	The query path defines the location where the query to be executed is stored. Instead of defining a file path, you may also define a scoped query name containing the class and the expression name (e.g. Person::Test). The query path may also refer to a folder that contains queries for each client. The folder must contain query definitions, only, which have been prepared for the test. When more clients are executed than there are queries in the folder, StressTestM starts from the beginning after running all queries.
distance	is the time in 1/10 seconds between starting two subsequent threads. Default: 10 (1 second)
clients	is the number of clients that are running in the test. Default: 1000
repititions	is the number of query calls to be processed. Default: 100

Defining data sources

StressTestM works on data base defined in a data source in an option file. You can define the data source directly in the option file or refer to an existing data source in the data catalog. The section in the option file describing the data source must start with

[StressTestM]

For referring to a data source by its data source name you must define the data source reference in the option file as:

DATA_SOURCE=DataSource1

In this case the data source must be defined in a data catalog. Data catalogs can be provided locally and on the server side. In one application, however, you can refer to only one data catalog. How to define data sources and file locations in the data and file catalog you can find in the “ODABA^{NG} – Server” documentation.

Database and dictionary

A dictionary and a database define a data source. While the dictionary contains the data definitions the database contains the data. Each data source definition should contain a dictionary definition and a database definition:

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE=C:\ODABA\Sample\Data\Sample.dat

Local configuration

Running StressTestM in a local environment will re-use the dictionary on the client side. For the database copies for database objects are created for each thread.

Client/server configuration

Data sources can be located on a server. In this case the data source definition refers to the server and a symbolic database path that is resolved by the servers file catalog:

SERVER_URL=ProjectServer
SERVER_PORT=6123

DICTIONARY=%SAMPLE_DICT%
DATABASE==%SAMPLE_DAT%

Running StressTestM in a client/server environment will create a new connectin, i.e. a new dictionaty and database handle for each client,

Options

Additional options that can be defined in a data source

	are:
VERSION	Internal database version number when the database is using version features. The version number allows copying the resource database according to a historical state. **VERSION**=version Default: current version
ACCESS_MODE	The access mode defines whether the database will be used in write/update mode or read only. When updating data in the expression the from-data source must be defined with Write option. **ACCESS_MODE**= <u>Read</u> **|** Write Default: Read (for from-data source), Write (fot to-data source)
NET	When running StressTestM with more than one thread, this option should be set to YES to enable multi user access for the database. **NET**=YES **|** <u>NO</u> This feature is supported under Windows, only. Under Linux, YES is used, always. Default (Windows): NO

Option file for StressTestM

ODABA^{NG}

An option file defines the data source, input and output files and other process specific parameters. The following example refers to the specification of the sample database source based on an ODABA^{NG} database.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[StressTestM]

DICTIONARY=C:\ODABA\Sample\Sample.dev
DATABASE= C:\ODABA\Sample\data\Sample.dat

NET=YES

When referring to data sources defined in the data catalog on the server you need not to define the data source sections.

20 License Utility (Licence)

Usually ODABA^{NG} applications require a licence from the software provider. Licences are a combination of user name and licence number. Each licence number works for the defined user name, only.

For licensing an ODABA^{NG} product you have to run the licence utility with the licence option file provided by the software vendor.

Running Licence

You can run Licence from a command line in DOS or UNIX. Before running Licence make sure that the data sources are available.

C:\ODABA\Licence.exe *owner_ini_file*

Licence.exe requires a licence.ini file in the same folder where Licence.exe is located. Such an option file has been provided with the installation on the ODABA^{NG} installation path. When ODABA^{NG} is not installed on the default installation path this file may need some modifications. It is not updated automatically.

We suggest to store the Licence.ini, the owner_ini_file and the Licence.bat to your ODABA^{NG} installation folder and to run Licence.bat.

owner_ini_file

The owner option file defines the owner and the licence numbers for the packages that need licences. This option file is provided by the software vendor.

Option file

An option file defines the data source, input and output files and other process specific parameters.

[SYSTEM] system section

DICTIONARY= C:\ODABA\ODE.SYS

[Licence]

DICTIONARY=C:\ODABA\Sample\Sample.dev