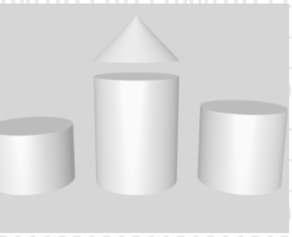


ODABA^{NG}

Workspaces

run Software-Werkstatt GmbH
Weigandufer 45
12059 Berlin

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

Berlin, October 2012

Content

1 Introduction 4
ODABA2 4
Platforms 4
Interfaces 4
User Interfaces 4

2 Workspaces 5
Workspace Support 5

1 Introduction

ODABA2

ODABA2 is an object-oriented database system that allows storing objects and methods as well as causalities. As an object-oriented database, ODABA2 supports complex objects (user-defined data types), which are built on application relevant concepts.

ODABA2-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.

ODABA2-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 ODABA2-applications a favourite possibility to solve highly complex jobs as come up in administrative and knowledge areas.

Platforms

ODABA2 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

ODABA2 supports several technical interfaces:

- C++, COM as application program interface (this allows e.g. using ODABA2 in VB scripts and applications)
- ODBC (for data exchange with relational databases)
- XML (as document interface as well as for data exchange)

User Interfaces

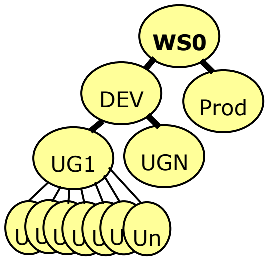
ODABA2 provides special COM-Controls that easily allow building applications in Visual Basic. On the other hand ODABA2 provides a special ODABA2 GUI builder.

2 Workspaces

Workspace Support

Workspaces can be used to store updates for metadata objects temporary until updating the metadata object has been finished. The administrator must enable the workspace feature explicitly for each database that should be used with workspaces.

Group workspace



You can share a workspace with a group of users. In this case the ini-file for the application must refer to the workspace that is the root for all workspace users in the group. Thus, you may have production (Prod) or development (DEV) workspaces, and below workspaces for sections or smaller user groups (UG1, …, UGN). Finally each user can create individual (user bound) workspaces below his root workspace.

The root for user group1 (UG1) could be defined as

WORKSPACE=DEV.UG1

In the ini-file (the system workspace WS0 must not be referenced in the workspace path).

Create

When workspaces are enabled you can create one or more individual workspace in the file menu of the application (**File/New workspace**). When a workspace is created it is opened automatically and all changes are stored in the opened workspace. All updates are visible only in the current workspace until they are discarded. Updated objects stored in a workspace can only be updated by the workspace owner and are locked for other users.

Workspaces allocated via the file menu are user bound, i.e. only the user that has allocated the workspace can open these workspaces.

You may also build hierarchies of individual workspaces by creating a new workspace while working in an open workspace.

Open

When working with different workspaces you can switch between workspaces by using the **Open workspace** action in the **file** menu of the main window.

Close

Closing a workspace will leave the current workspace and goes to the next higher level. The changes made in the current workspace are kept in the workspace and still not visible outside the workspace. You can close only those workspaces that have been opened in your application, i.e. you cannot close the root workspace defined in the ini-file. Closing the workspace is possible by selecting **Close workspace** from the **file** menu in the main window.

Consolidate

Consolidating a workspace will save all changes stored in the workspace on the next higher level or in the database when (consolidating a top workspace (workspace below WS0). The workspace is cleared up and can be used for further work. Consolidating a workspace will not delete it physically, but only remove all data from the workspace to the next higher level.

Discard

Discarding a workspace will remove all changes stored in the workspace without saving those on the higher level. Usually one should avoid discarding a workspace when global objects have been updated. Since database consistency checks are made against the updated versions of the metadata objects discarding a workspace may lead to rule consistency problems in the database, i.e. some special rules established in the Bridge^{NA} system might be violated. This is usually not the case when updating local or semi global metadata objects.

Discarding a workspace will not delete it physically, but only remove all data from the workspace.

Delete

Deleting a workspace is only possible when the workspace does not contain any data. This means that the workspace must be discarded or consolidated before deleting it. Delete will remove the workspace physically.

Administrating group workspaces

Group workspaces can be administrated with the **Workspace** utility that allows discarding, consolidating and deleting workspaces from a command line. You can use the Workspace utility also for administrating the individual workspaces. For more information see the ODABA "*Database Utilities*" documentation.

Workspace

A workspace is a long-time transaction that may exist even after process termination. Workspaces are used for storing temporary updates. As long as working in a workspace changes are not written to the database but to a special workspace area.

A workspace is like a transparent slide on top of the database. After finishing a series of updates that may take several days or weeks you may consolidate the changes, i.e. storing all updates to the database or discard them.

Workspaces are identified by names and allocated by the database system. Usually workspaces reside in the same location as the root of the database.

In contrast to process transactions several users may work in the same workspace but only one can consolidate the changes. After consolidating or discarding changes the workspace is cleared up, i.e. it will be empty.

Shadow database

One problem with long transactions is keeping the database consistent without blocking the whole system by locked instances and collections.

Example: One transaction adds an instance with a unique name "Paul" in a workspace. This is not visible from other workspaces but, nevertheless, no other user is allowed to add "Paul" again.

Hence the system must know in a way the state of the database, as it would look like when all workspaces have been consolidated. This is also important when checking update, insert or deletion rules, which may involve other instances.

For this purpose ODABA provides a shadow database when working with workspaces. The shadow database reflects the database state at any time as if all changes have been made directly in the database.

Thus, it becomes possible to refer to the states in the shadow database as well as to the states in the "real" database without updates saved in workspaces.

Nested workspaces

Within a workspace you may create another workspace below the current one. When consolidating the lower workspace the changes are stored in the upper workspace but not in the database. When consolidating the upper workspace while lower workspaces are opened only the consolidated changes from lower workspaces are stored into the database. In this case the upper workspace is not completely cleared up since all instances that are in use in lower workspaces are still locked in the database.

Ownership

A user can own workspaces. In this case only the user who has allocated the workspace is able to open it again. It is, however, also possible to create public workspaces that can be accessed by any user.

Usually private workspaces are created by a user from within an application. The application itself takes care about user control. Public workspaces are created by administrators by establishing a workspace for a certain user group (as system or developer).

Whether a workspace is private or public depends whether it is created with user identification or not.

Enabling workspace

Before using workspaces this feature must be explicitly enabled. Enabling the workspace feature will create the shadow database and installs a workspace 0 which is the base for all other workspaces. Workspace 0 is a system workspace and cannot be discarded or consolidated or deleted. It is allocates at the same location as the database with the extension .ws0.

```
DatabaseHandle dbhandle(dictptr,"C:/ODABA/test.db",NO");
                                            //exclusive database

dbhandle.EnableWorkspace("C:/ODABA/test.shadow");
```

Open workspace

You can create or open an existing workspace with a database handle.

```
dbhandle.OpenWorkspace("Wspace1");          //publ. workspace

dbhandle.OpenWorkspace("Wspace1","user27");//priv. workspace
```

After opening the workspace all updates are stored in the opened workspace. When the workspace is used the first time it is created automatically. When it does already exist the existing workspace is opened.

Private workspaces can be opened only when passing the same user name that has been passed when opening the workspace the first time. Opening a public workspace any username or none can be passed.

You can check whether a workspace exists using the LocateWorkspace() function, which returns *true* when the workspace has already been created.

```
dbhandle.LocateWorkspace("Wspace1");
```

After opening you are in the context of the workspace. When opening another workspace now this is created on top of the current workspace. I.e. in the context of the current workspace:

```
dbhandle.OpenWorkspace("Wspace1");
dbhandle.OpenWorkspace("Subspace11");
```

This will create *Subspace11* in *Wspace1*. The same you achieve with calling

```
dbhandle.OpenWorkspace("Wspace1.Subspace11");
```

Finish workspace

You can finish a workspace by consolidating or discarding changes.

Consolidate

ConsolidateWorkspace() will consolidate all changes made in the workspace. You can consolidate the currently opened workspace, only, i.e. you must open the workspace before consolidating.

For consolidating a workspace is must be opened with exclusive use. Only when no other user has access to the workspace it is possible to consolidate it.

```
dbhandle.OpenWorkspace("Wspace1",NULL,YES);
dbhandle.ConsolidateWorkspace();
```

Discard

If you want to through away all changes made in the workspace you can use DiscardWorkspace() for the currently opened workspace.

```
dbhandle.DiscardWorkspace();
```

Finish nested workspace

There is a difference finishing a top workspace that has no other workspaces above or when finishing a workspace with higher workspaces.

You may consolidate or discard a top workspace but you cannot discard a workspace that has one or more lower workspaces on top. When consolidating a non-base workspace, i.e. a workspace that is on top of another one, changes are stored in the lower workspace. Thus, an upper workspace contains all consolidations from its lower workspaces or changes made directly in the workspace.

When consolidating a workspace that has lower workspaces Only changes made directly in the workspace or consolidated changes from lower workspaces are stored.

Close workspace

Workspaces are closed when closing the database. It is possible, however, to close the active workspace explicitly.

```
dbhandle.CloseWorkspace();
```

When working in a hierarchy of workspaces you can close all workspaces in the hierarchy. In this case further updates are stored directly in the database.

```
dbhandle.CloseWorkspace(YES);
```

Listing workspaces

You can list available workspaces for a special user, all workspaces or public workspaces. Workspaces are retrieved hierarchically based on a work space or the database itself.

Get Workspace

GetWorkspace() returns the name of a workspace by index relative to a given root.

```
// first top workspace
dbhandle.GetWorkspace("",0);

// first workspace below Wspace1 for user 'user27'
dbhandle.GetWorkspace("Wspace1",0,"user27");
```

When passing a user name to the function it returns only workspaces for the selected user. Usually the function returns the workspace name, which is the name of the workspace path without the root path, in the result buffer, i.e. the value is removed when another database handle function is called. You may, however, pass an area for storing the workspace name.

```
char              name[40];
dbhandle.GetWorkspace("Wspace1",0,"user27",name);
```

The workspace list is buffered internally. It is created when calling GetWorkspace() or LocateWorkspace() the first time. To ensure that the list is up-to-date you may pass the refresh-option (YES or true) for updating the list.

Delete workspace

After creating a workspace it will remain until it is explicitly deleted. Even Discard or Consolidate will not remove the workspace but leave an empty one. For removing a workspace you can use DeleteWorkspace().

A workspace can be deleted only when there are no lower workspaces defined and when it is completely empty.

```
dbhandle.DeleteWorkspace("Wspace1","user27");
```

When the workspace had been allocated with user name the user name has to be passed also for deleting the workspace.

Performing consistency checks

Working with workspaces creates some problems for performing logical consistency checks (handling Store, Insert or Delete-Events). The application has to decide whether checks are performed against the original database (which does not reflect changes made in the workspace) or against the shadow database, which reflects all updates stored in workspaces. Assuming that most of the workspaces are consolidated later consistency checks against the shadow database are better than those against the original database. In any case there is a risk that discarding a workspace may create inconsistency.

For performing checks against the shadow database you can activate the shadow database.

Activate Shadow Base

Activating the shadow database will direct all read operations to the shadow database instead of the original database.

```
dbhandle.ActivateShadowBase();
```

Save consistency checks

To perform save consistency checks you must lock all instances involved in the checking. This can be done also when the shadow database is active as shown in the following example for the DBStored handler in a Person context class

```
Logical          sPerson::DBStore()
{
        logical         term = NO;
        PropertyHandle  *pers_pi = GetPropertyHandle();
        PropertyHandle  *boss_pi = pers_pi->GetPropertyHandle();
        Pers_pi->GetDataBaseHandle()->ActivateShadowBase();

        boss_pi = pers_pi->GetPropertyHandle("boss");
        if ( boss_pi->Get(FIRST_INSTANCE) )
           If ( boss_pi->GetPropertyHandle("income") <
                        boss_pi->GetPropertyHandle("income") }
              term = YES;             // update not possible
           else
              if ( boss_pi.Modify() ) // cannot lock instance
                 term = YES; ;        // update not possible
              else
                 poss_pi.Save();      // Lock instance in workspace

        Pers_pi->GetDataBaseHandle()->DeactivateShadowBase();
        Return(term);
}
```

Modifying and saving the instance will lock the instance for all other workspaces except subordinated ones. Thus no other user can modify the instance that has been involved in the consistence check until the workspace is consolidated. When Modify() fails the instance is locked by another user and the transaction is not save since the other user may discard the workspace or roll back a transaction.

Deactivate Shadow Base

At the end of the process you must deactivate the shadow database to ensure that processing continues with the original database.

```
dbhandle.DeactivateShadowBase();
```