Moving your Omnis App to the Web & Mobile

What you need to move your Omnis app from desktop to Web & Mobile.

By Andreas Pfeiffer and Richard Mortimer

Senior Omnis Consultants

White Paper

August 2019

No part of this publication may be reproduced, transmitted, stored in a retrieval system or translated into any language in any form by any means without the written permission of Omnis Software.

© Omnis Software, and its licensors 2019. All rights reserved.
Portions © Copyright Microsoft Corporation.
Regular expressions Copyright (c) 1986,1993,1995 University of Toronto.

© 1999-2019 The Apache Software Foundation. All rights reserved.
The Omnis product includes software developed by the Apache Software Foundation (http://www.apache.org/).

OMNIS® and Omnis Studio® are registered trademarks of Omnis Software Ltd.

Microsoft, MS, MS-DOS, Visual Basic, Windows, Windows 95, Win32, Win32s are registered trademarks, and Windows NT, Visual C++ are trademarks of Microsoft Corporation in the US and other countries.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

IBM, DB2, and INFORMIX are registered trademarks of International Business Machines Corporation.

ICU is Copyright © 1995-2003 International Business Machines Corporation and others.

UNIX is a registered trademark in the US and other countries exclusively licensed by X/Open Company Ltd.

Sun, Sun Microsystems, the Sun Logo, Solaris, Java, and Catalyst are trademarks or registered trademarks of Sun Microsystems Inc.

J2SE is Copyright (c) 2003 Sun Microsystems Inc under a licence agreement to be found at:
http://java.sun.com/j2se/1.4.2/docs/relnotes/license.html

MySQL is a registered trademark of MySQL AB in the United States, the European Union and other countries (www.mysql.com).

ORACLE is a registered trademark and SQL*NET is a trademark of Oracle Corporation.

SYBASE, Net-Library, Open client, DB-Library and CT-Library are registered trademarks of Sybase Inc.

Acrobat, Flash, Flex are trademarks or registered trademarks of Adobe Systems, Inc.

Apple, the Apple logo, AppleTalk, and Macintosh are registered trademarks and MacOS, Power Macintosh and PowerPC are trademarks of Apple Computer, Inc.

HP-UX is a trademark of Hewlett Packard.

OSF/Motif is a trademark of the Open Software Foundation.

CodeWarrior is a trademark of Metrowerks, Inc.

Omnis is based in part on ChartDirector, copyright Advanced Software Engineering (www.advsofteng.com).

Omnis is based in part on the work of the Independent JPEG Group.

Omnis is based in part of the work of the FreeType Team.

Other products mentioned are trademarks or registered trademarks of their corporations.

Table of Contents

Introduction 5

Migration or re-design? 5

Why can't I just migrate the existing windows automatically? 5

The rapid way to get a web and mobile app 5

Web app vs. desktop application 6

Modeless enter data 6

Scope of variables 6

Database session 7

There is no menu bar 7

Avoid too many fields in a form 7

Moving your Omnis App to the Web & Mobile 8

Database connection 8

Creating a session pool 8

Closing the session pool 9

Getting a session from the pool 9

What scope should I use for the session object? 9

How do I call a method in the Startup_Task from within the remote task? 9

Database Layer 10

How to use the session object variable? 10

Why use table classes? 10

Using a table class to manually execute a select statement 11

Using table classes with schema classes 12

SQL error handling 13

SQL error handling in custom methods 13

Business Rules 14

Login form 15

Create a user object 15

Avoid task variables! 15

The login form 16

Remote forms 17

How to implement a menu system? 17

Use inheritance to make your UI design easier 17

Simplify your UI design 18

Use Subform Sets to support popup forms 18

Make your UI design look great 18

© Omnis Software, and its licensors 2019. All rights reserved.

More information 19
Training 19
Consulting 19

© Omnis Software, and its licensors 2019. All rights reserved.

Introduction

Migration or re-design?

This white paper is designed for existing Omnis Studio developers who want to move their applications to an Omnis JavaScript application.

If you have already separated your business rules into object and/or table classes - then that's a great start! All you need to do is start with section "Login form" to find out how to handle a login remote form and how to build a menu system in order to switch between different remote forms.

If you **have not** separated your business rules from your window classes - if you have your SQL code distributed in all of your windows – or if you are still using DML to access your Omnis datafile – then **don't panic**!

In this case, start with the section "Database connection" to understand how easy it is to implement the business rules and your SQL in table classes. Did you know that you can access your Omnis datafile using SQL? If not, then check out the section “Database Connection” and choose the OMSQLDAM to connect to your Omnis Datafile.

Why can't I just migrate the existing windows automatically?

… into Omnis JavaScript forms?

Well - theoretically it might work - but only to a certain level. But then you would still need to do a lot of things manually.

Web applications are not desktop applications! For example, there is no menu bar available, like most desktop apps have, so you need an alternative navigation system. Instead you could have a main remote form that hosts a subform where the classname is assigned dynamically. Please see section "How to implement a menu system" for this.

In addition, web applications should not use any code that stops code execution. One example would be the command "Enter data" – if this command is used and the connection to the end user interrupts - which can happen in a web app - then Omnis will wait for ever if you do not have a time out set in the remote task. Modern web applications are always in modeless data entry mode for this reason and you should conform to this standard practice.

The rapid way to get a web and mobile app

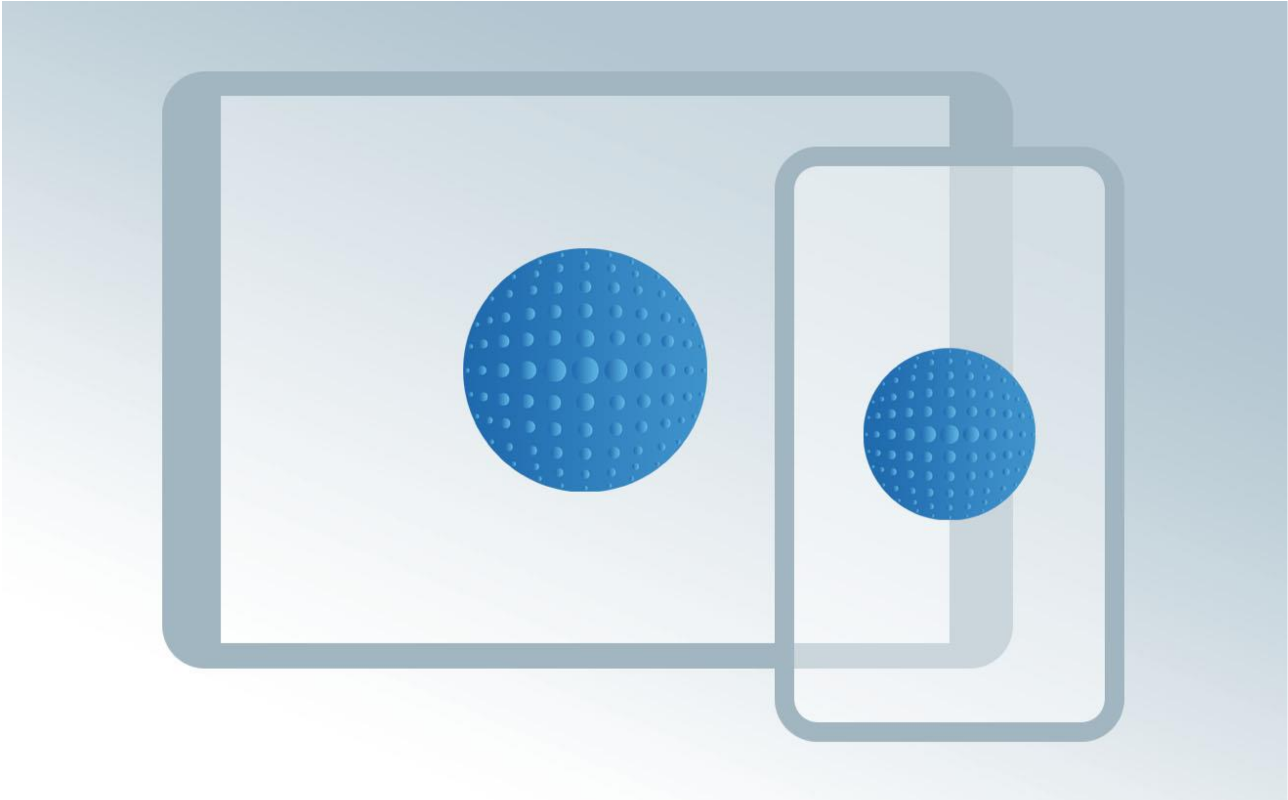
… is using Omnis Studio, because you will be able to use all your table classes, object classes and any other code that is not in your window classes.

Even if you need to re-design your user interface, you will find out that it is easy for you because you understand how Omnis works and you are familiar with debugging Omnis code, which is a very helpful feature when developing (as you know). There is no need to learn any other development language or tool to create web and mobile apps.

© Omnis Software, and its licensors 2019. All rights reserved.

Web app vs. desktop application

Desktop applications are powerful, but web and mobile applications have some significant advantages. The main difference is that web and mobile apps are universal – you can access them from anywhere – so you can run your application on the Omnis server and the end user just needs a browser to connect to your app. Any updates are easy to distribute because you just need to replace the library on the server. All clients will work with the latest version.



But this advantage also comes with a responsibility. There are some issues that you need to take into account.

Modeless enter data

As already mentioned, all forms must be in “modeless enter data” mode. Web apps do not have buttons that switch the form into the enter data mode. In contrast all forms are always in enter data mode. Check out this technical note https://developer.omnis.net/technotes/tngi0016.jsp describing how you can develop without the "enter data" command.

Scope of variables

As for any development, your variables should have a *minimum scope*. That said it is not a good idea to use class variables since they will be shared for all instances of a class. Imagine you would use a class variable in a remote form. All of the users will share the same content.

Instead you should use *instance variables,* such as row and list variables in order to use them with entry fields and grid fields.

© Omnis Software, and its licensors 2019. All rights reserved.

Database session

In a desktop app, only one user uses the database session at a time, while in a web app many users could be connecting to the database. Because of this, you are advised to use a session object variable that is taken from a session pool. This is described in the next section.

There is no menu bar

As already mentioned, a typical web app does not have a menu bar. Nevertheless, Omnis has a Navigation object that allows similar functionality but it is part of the form. See the section "How to implement a menu system".

Avoid too many fields in a form

Classical desktop apps can have hundreds of fields on one window. This will never work for mobile users, and is not good practice for desktop browser based web apps. For this reason, it makes sense to split windows into several subforms and only load those forms when they are really needed. This will enhance the readability (since only the important information is shown) and speed up the application. (See also the section "Simplify your UI design")

© Omnis Software, and its licensors 2019. All rights reserved.

Moving your Omnis App to the Web & Mobile

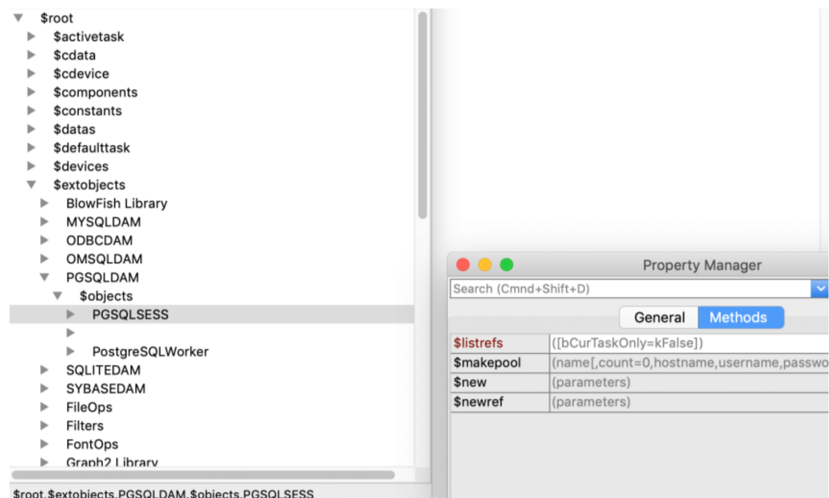
The remainder of this white paper shows you how to enhance your app to support web and mobile, quickly and easily, including how to create your database layer, login form, and main forms.

Database connection

A database session in Omnis Studio is handled as a session object. For this you can either subclass one of the Database Access Modules (DAMs) by assigning the superclass property of an object class to the External Object of your chosen database vendor or you might want to do this by making a *session pool*. The latter might be better suited for a web and mobile app because you can create a pool with a number of database sessions in it that can be re-used from different clients. This avoids a login for each client and enhances the speed of your application.

Creating a session pool

The Database Access Modules (DAMs) can be found in the External Objects group. If you do not know the path it might be easiest to open the Notation Inspector (F4/Cmd-4) and drill down to the desired DAM external - there you will find a $objects group that contains the session object that you are looking for. Then open the Property Manager (F6/Cmd-6) to see the properties and methods of this session object. You will find a method $makepool that you can use in order to generate the session pool.



DAM external, session object and $makepool method

Since the session pool will become global, it is probably best to make a $logon method within your Startup_Task so that you can call this method from its $construct. That ensures that every time you open the library it will generate the pool automatically. So the code will be something like:

```
Do $extobjects.PGSQLDAM.$objects.PGSQLSESS.$makepool(
    'myPool',1,'192.168.10.10','myUser','myPassword')
    Returns iMyPoolRef
```

The second parameter (1 in this example) is the number of initial sessions in the pool. But the pool also has properties that allow us to see the number of sessions already used in the pool and we can increase the number of sessions by changing another property if all sessions are in use.

Please note: The first parameter of the $makepool method is the name of the session pool. This must be a unique name. Since the pool is global you should also close the pool when you destruct your Startup_Task. If you do not close Omnis and just close and re-open your Startup_Task (for debugging reasons) it will fail to open a session pool with the same name. You can either check the variable iMyPoolRef (item reference) that you can use as a return or you can look in the Notation Inspector if the session pool becomes visible in $sessionpools.

© Omnis Software, and its licensors 2019. All rights reserved.

Closing the session pool

To close the session pool, you can use the following code in the $destruct of your Startup_Task:

```
Do $sessionpools.$remove(iMyPoolRef)
```

Please note: iMyPoolRef is an instance variable of type Item Reference that has been used as the return value of the $makepool method.

Getting a session from the pool

Once a pool is valid you can use the $new method in order to return a session from the pool. To do this it makes sense to do this in a separate method, i.e. in Startup_Task:

```
$getSessionFromPool:
If iSessionPoolRef.$poolsize=iSessionPoolRef.$inuse
  Do iSessionPoolRef.$poolsize.$assign(iSessionPoolRef.$inuse+1)
End If
Quit method iSessionPoolRef.$new()
```

Please note: iMyPoolRef is an instance variable of type item reference that has been used as the return value of the $makepool method. The session pools $poolsize can be increased when all sessions are busy ($poolsize=$inuse). The method will return a new object using the $new method of the pool.

What scope should I use for the session object?

Now that we have a method (see above) that will return a session from the pool, the question is, how global should the scope of the session object variable be? You might think that the smaller the scope, the better the design - which is generally the case. But if you use a remote task (for the JavaScript client), each user will have a separate instance of the remote task and might use the session globally until the application is closed and the remote task is destroyed. For this reason, I tend to use a task variable of type *object* or *object reference* within the remote task.

Please note: Make sure you use type *object references* when you plan to pass the object as a parameter to another method. If using an object (rather an object reference) you will end up with a copy of the object.

How do I call a method in the Startup_Task from within the remote task?

Above we made a $getSessionFromPool method and now it is time to make use of it. If using a task variable (tSessionObj) in the remote task it would make sense to do this from the $construct of the remote task:

```
Do $itasks.myLib.$getSessionFromPool() Returns tSessionObj
```

Please note: . Replace "myLib" with the name of your library. The name of the instance of the Startup_Task usually takes the name of your library. Therefore, it makes sense to use the $defaultname property of your library. This ensures that even when renaming the physical library name the internal name will remain the same.

Alternatively, you could use

```
Do $itasks.[$clib().$name].$getSessionFromPool()
Returns tSessionObj
# $clib().$name will return the name of your library.
```

© Omnis Software, and its licensors 2019. All rights reserved.

Database Layer

How to use the session object variable?

How you use the session object variable depends on your class design. I recommend using table classes to handle any SQL related code. Table classes can be used either with an accompanying schema class or without. The latter case can be used to execute manual select statements such as complex joins etc. The binding to a schema class using the $sqlclassname property of the table class is perfect if you want to work with row variables to represent a single row within your remote form.

Anyhow, in both cases, it is very simple to make use of the session object and if you have a single table class ("taSuper") as a superclass assigned for all of your other table classes, it is just a single line. Make sure that the $designclassname property of your superclass and all other classes is set to your remote task (not the Startup_Task). You can also do this by inheriting the $designclassname property of the inherited classes and only set it in the superclass. This will ensure that the task variable tSessionObj becomes visible when you code in your table classes.

Now all you need is a code line in $construct of your superclass taSuper:

```
Do $cinst.$sessionobject.$assign(tSessionObj)
```

Now any code that your table class executes internally will use the session object.

Please note: If your inherited classes already have their own $construct you either need to inherit them using the context menu on the method name (if there is no other code) or you can also use the command "Do inherited" to ensure that the $construct of the super class is executed.

Why use table classes?

There might be reasons why you would use object classes rather table classes. However, table classes can do exactly what object classes can, but have more inbuilt methods and - for me, the important point - do not require an extra object variable in the remote form. Instead the list or row variable becomes the instance of a table class.

© Omnis Software, and its licensors 2019. All rights reserved.

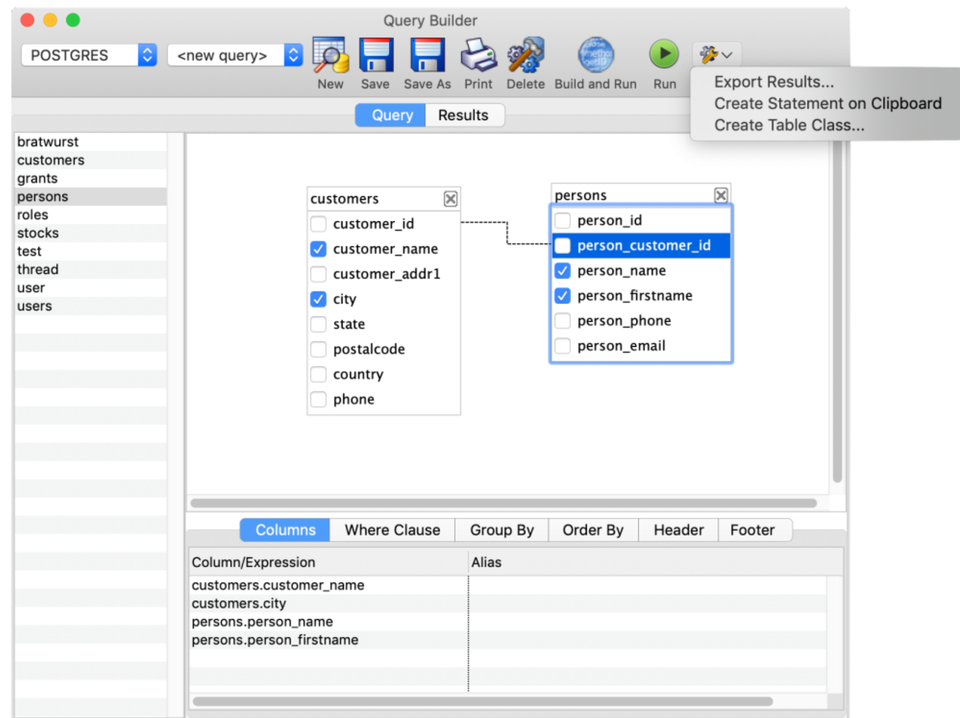
Using a table class to manually execute a select statement

A table class that is not bound to any schema or query class can be used to execute a manual select statement. The list that is used as the instance of the table class will then automatically be defined with the columns that are returned by the SQL statement. When binding the $sessionobject (see above) to a table class each table class has its own $statementobject too. Therefore you can use

```
$cinst.$statementobject.$execdirect()
or $cinst.$statementobject.$prepare()
and $cinst.$statementobject.$execute
```

within the table class to execute any SQL against your database session.

The easiest way to test this is to use the Query Builder. When you have a SQL session open in the SQL Browser of the Omnis IDE you can use the Query Builder and drag the tables that you need for the query onto the white space. Use the mouse to drag a line between the foreign key of the child table to the primary key of the parent to determine the relation. Select the columns that you want to be fetched and click on "Build and Run". You can also manually change the query and click the "Run" button instead. Once you are satisfied with the query, go to the "Other" button and click "Create Table Class …". A wizard will appear and ask for the table class name. In addition, you can choose to create a remote form automatically.



Query Builder - Other button - Create Table Class…

© Omnis Software, and its licensors 2019. All rights reserved.

Therefore, the result of this will be that Omnis generates a new table class in your library that contains a Statement Block in your code. It will contain the same SQL query as created in the Query Builder in a method called $load

```
Class methods
  $construct
  $load

 1 Begin statement
 2 Sta: SELECT
 3 Sta:       "customers"."customer_name",
 4 Sta:       "customers"."city",
 5 Sta:       "persons"."person_name",
 6 Sta:       "persons"."person_firstname"
 7 Sta: FROM
 8 Sta:       "persons",
 9 Sta:       "customers"
10 Sta: WHERE
11 Sta:       "persons"."person_customer_id"="customers"."customer_id"
12 End statement
13
14 If $cinst.$statementobject.$execdirect()
15   Do $cinst.$fetch(kFetchAll)
16   Quit method kTrue
17 Else
18   Quit method kFalse
19 End If
```

$load method of the new table class created by the Query Builder

You can now use this table class in your remote form (or window) as follows.

1. Declare an instance variable of type list (i.e. iDataList) with the subtype of the table class that you just created.
2. Call iDataList.$load() within the $construct of your remote form.
3. Add a Data Grid to your remote form and type in the variable name into its $dataname property.
4. Assign the number of columns that you expect to the $designcols property of the grid.
5. Test your remote form. You should now see the expected result list of the query.

Please note: As you can see the $construct is now using the SQL session that you use within the Omnis IDE and this is fine for testing your form. However, this session will not be available during runtime and you should inherit the table class from your taSuper table class and also inherit the $construct method as described earlier.

Using table classes with schema classes

Alternatively, you can use a table class that you can bind to a schema class. This approach is recommended to load, update, insert or delete a single row in a specified table.

1. To create a schema class, you can drag a server table from the SQL browser to your library.
2. To create a new table class, you can make a subclass from your super table class using its context menu "make subclass"
3. The table class has a $sqlclassname which cannot be assigned if this property is colored blue – this means it is inherited from the super class. But you can override this using the context menu on the property name. You can then choose the schema class to bind it to the table class in $sqlclassname.

Now when using a row variable in the remote form, you can specify the table class as a subtype of the row variable. It will then use the column specification of schema class that is attached to the table class and provide all public methods that are in the table class, including any of your own methods if you want. For example. you can do the following to load a customer:

© Omnis Software, and its licensors 2019. All rights reserved.

```
Do iDataRow.$select("Where customer_id = ",iDataRow.customer_id)
Do iDataRow.$fetch()
```

This code will use the customer_id column in your row variable and fire the select statement where the customer_id is equal to the value someone typed into the customer_id column of the row.

SQL error handling

It is easy to implement an automated error handler since all inbuilt methods such as $select, $insert etc. will automatically call a predefined method named $sqlerror.

If you are using a super table class, you can override the $sqlerror method just by creating one in taSuper:

```
$sqlerror:
Set reference lvStatementObjRef to $cinst.$statementobject
Calculate iScript as lvStatementObjRef.$sqltext
Calculate iErrorText as lvStatementObjRef.$nativeerrortext
```

Please note: The local variable lvStatementObjRef is of type *item reference*. You can use it to point to the internal statement object of your table class instance (list or row variable defined on the table class). The statement object has a number of properties that allow you to find out what is going wrong.

The method above uses instance variables that can be returned from within a separate method:

```
$getErrorText:
Quit method iErrorText
```

That way you can ask for the last error text from within your remote form. For example, if you want to do an insert of the data that are in your data row (iDataRow) you can use the following code behind the insert button:

```
On evClick
  If iDataRow.$insert()
  Else
    Do $cinst.$showmessage(iDataRow.$getErrorText())
  End if
```

Please note: Never use ok messages or $showmessage from within a data object directly. The UI should be responsible for displaying any ok message, but the datalayer object (the row or list variable) is responsible for providing the SQL error text.

SQL error handling in custom methods

Do you remember the $load method of the table class that we received from the Query Builder? There we would need to manually call the $sqlerror method to trap any SQL error:

```
If $cinst.$statementobject.$execdirect()
  Do $cinst.$fetch(kFetchAll)
  Quit method kTrue
Else
  Do $cinst.$sqlerror()
  Quit method kFalse
End If
```

© Omnis Software, and its licensors 2019. All rights reserved.

Business Rules

Depending on the design of your fat client application, you might be able to use your existing business rules as long as you have them in an object or table class already, and not in the window itself. But even then, it might be easy enough to move them from the UI into your table classes.

Once you are using table classes, it is quite easy to implement your business rules. For example, if you want to check if columns of your insert statements are not null or have a certain value, you can implement a $check method in your table class. To make it easy can do do this in the super table class taSuper:

```
$check:
Quit method kTrue
```

Say we want to call the $check method every time we do an insert, we can then override the inbuilt method by creating a $insert method in the super table class:

```
$insert:
If $cinst.$check()
  Do default Returns ok
  Quit method ok
Else
  Quit method kFalse
End if
```

Please note: "ok" is a local variable of type boolean and is used to find out if the default insert was ok. If the $check method of the current instance returns kFalse for any reason the $insert will also return kFalse.

So far nothing really has changed because the $check method always returns kTrue.

But now we decide that one of the inherited table classes needs to make use of the $check method since we want to implement a business rule. We can now simply override the $check method in the inherited table class using the context menu on the method name in the inherited class and return kFalse if one of our rules are failing:

```
$check of the inherited class:
If isclear($cinst.name)
  Calculate iErrorText as "Name is missing"
  Quit method kFalse
End if
Quit method kTrue
```

Please note: When using $cinst inside the table class it means that you are referring to the row (or list) that is using this table class. Therefore, in the case of an insert it must be the row variable that contains the data. Since we already have the instance variable iErrorText in the superclass, we can use this to hold the error text.

Now the insert from the remote form would be something like this:

```
If iDataRow.$insert()
Else
  Do $cinst.$showmessage(iDataRow.$getErrorText())
End if
```

Please note: This is analogous to the section above. Now we either receive a SQL error or the error message from the business rule.

© Omnis Software, and its licensors 2019. All rights reserved.

Login form

Create a user object

Create an object class that contains an instance variable of type *row* where the subtype is set to the table class that is bound to the schema class that represents the user table.

This object class should then have a public method called $login that either returns true if the login was successful or false if the login failed. This method can then use the row variable to load the user record and compare the - hopefully hashed - password that is stored for this user in the database.

Omnis Studio 10 has an external worker object that can hash the password. Just compare the two hash values in the $login method.

In addition, you might want to load the access role grants in a list within that object for this user. This allows you to have a public method in the user object class, i.e. $grant, where you can parameterise a string which can be verified using a $search method of the grants list.

For example, this is the $grant method that I use in the user object class:

```
$grant (parameter pGrant of type character):
Do ivGrantList.$search(
  $ref.grant_name=pGrant,kTrue,kFalse,kFalse,kFalse)
Quit method ivGrantList.$line>0
```

Avoid task variables!

Variables of the scope task - in this case the remote task - are “evil” as long as the meaning of the variable is not really that global. Therefore, in most cases *you should have only two task variables* in your application:

1. The session object (see above) as the database session really is global to the user session.
2. The user object variable as the user really is global to the session.

So for this purpose, you can declare a user object variable as a task variable and access any of its public methods from anywhere in your application. For example, if a menu object is created any of the menu items can be granted using the $grant method of the user object. Or if you need the user name of the current user working with your application you could have a $getUserName method that returns the user name of your internal row variable and then you can use it in your forms or in your table classes to store the username in your tables.

© Omnis Software, and its licensors 2019. All rights reserved.

The login form

The login form is the first form that should be opened in your app. For example apart from any graphical elements, such as the customers corporate identity, you should have a field that allows the end user to enter the username and another field that takes the user password. The latter one should have the $ispassword set to kTrue, to hide the user input automatically.

Finally, you would then have a button element that asks the user object to proof the login and if this is successful change the remote form:

```
On evClick
  If tUserObj.$login(iUsername,iPassword)
    Do $ctask.$changeform("jsMain")
  Else
    Do $cinst.$showmessage("Access not allowed")
  End If
```

Please note: tUserObj is the remote task variable that has the subtype set to the user object class. iUsername and iPassword are instance variables of the login remote form. The $changeform method of the remote task changes the remote form.

© Omnis Software, and its licensors 2019. All rights reserved.

Remote forms

How to implement a menu system?

The method $ctask.$changeform is perfect to switch between different forms, such as from the login form to the main form, but it does not allow you to have an overlaying menu. Therefore it is easier to use a main remote form that has basically two elements on it:

1. A field that shows a menu.
    1. For larger Layout Breakpoints you can use a Navigation Menu Object, for example.
    2. For smaller Layout Breakpoints you could use a popup list or simulate a Hamburger Menu
2. A subform component that will be assigned dynamically. This one should then take the remainder of the space. You can use the $edgefloat properties in order to maximise the field according to the browser or device size.

When the list is built for the Navigation Menu Object this method can use the $grant method of the user object to allow the classes to be listed in the menu or not depending on the access rights of this user.

When the user chooses one of the menu items you would then assign the $classname property of the subform component:

```
On evMenuEntryClicked
  If pLineIdent>0
    Do $cinst.$objs.subform.$classname.$assign(
        $classes.[pLineIdent].$name)
  End If
```

Please note: The list that is used for the Navigation Menu Object is designed to have a column that contains the ident of the class in the library. The event parameter pLineIdent contains this id and can therefore be used to identify the class using square brackets. The name of the subform component is "subform" in this case.

The user can now switch between different remote forms using the Navigation Menu Object.

Use inheritance to make your UI design easier

Some of your remote forms might have the same buttons, i.e. "insert", "update", "search" etc. So you can use a superclass that provides the buttons and the functionality for each of the inherited classes. When you implement a single row variable as an instance variable with no subtype you can then - instead of early binding using the subtype - use the $definefromsqlclass method to define the row in either of the inherited classes.

For example, you could have the following code in the inherited class:

```
Do iDataRow.$definefromsqlclass($tables.taCustomer)
```

Please note: iDataRow is declared in the superclass while the $definefromsqlclass is used in the $construct of the inherited class. This way each inherited class can refer to a different table class.

It is now easy to implement the code for the insert button in the superclass:

© Omnis Software, and its licensors 2019. All rights reserved.

```
On evClick
  If iDataRow.$insert()
  Else
    Do $cinst.$showmessage(iDataRow.$getErrorText())
  End if
```

Please note: The code is now in the superclass. This is just one method that will serve all of the inherited classes! The individual business rules are now in each table class.

Simplify your UI design

It is good advice to design your app for mobile devices first. Less information is better and clearer for the user: if you think about it, the more information you have got on one page the more difficult it will be to read and to understand.

You are advised to use two different Layout Breakpoints and use containers (paged pane with only one page) and the $edgefloat property to fill the rest of your screen. For this purpose, you need to use the kEFPosn.. constants. They allow you to stretch a paged pane component on one side that contains i.e. buttons to control the data.

Use Subform Sets to support popup forms

You can use subform sets to open other forms on top of the main form. These subforms can be dragged, resized and closed so that it allows a desktop like form handling. For further information please check out the JS Subform Set library in the sample section of the Omnis IDE Hub or this link: https://omnisservice.mh.omnis-software.com/jsgallery/jsgallery.htm (scroll down to subform sets).

This is not recommended for mobile applications. Instead you might want to use the Subform Set Panels. There is also example code for this available in the sample section.

Make your UI design look great

As application developers, you want to sell your web and mobile apps and you want to find many customers that are interested. And as we should all know, good UI design will help to sell your app. And this might be one of the big differences compared to legacy fat client applications. A modern web application has to have good design.

Here are some tips for it:

- “Less is more.” So do not overwhelm the user by adding more and more components to a remote form. Try to hide as much as you can so that the user can concentrate on the important part only. So make your design minimal.
- Compare your design with modern websites: use CSS classes to enhance your design.
- Make use of CSS classes to customise every single element if you want. Please refer to the documentation for this: https://omnis.net/developers/documentation/
- You can also use the animation feature but try not to be too intrusive. For example, if you hide or show elements, using fade out and fade in, the visibility change can take something like 300 Milliseconds, so do not overuse this feature.

Always remember, customers first check out the look of the application because they may not able to understand the underlying code. A full featured, powerful application may not sell if it does not look modern and well designed.

© Omnis Software, and its licensors 2019. All rights reserved.

More information

We would love to help you with more resources and would like to help you with your Omnis Studio migration project. So please check out the following items.

Training

Omnis Software offers online and onsite training. For example, there is a free course available at https://omnis.net/developers/academy/

But we do offer more enhanced courses if you like. Please get in contact with your local sales team: https://omnis.net/contact/ for more information.

Consulting

We can make the transition easy. If you need any help to get the first steps (or more) done, we can offer Omnis specialists to support your migration process. This is also a great way to learn about the techniques used for Omnis Studio JavaScript. Please get in contact with your local sales team: https://omnis.net/contact/

© Omnis Software, and its licensors 2019. All rights reserved.