Omnis Academy

Documentation to accompany the creation of the Cucina Piccola application in the Omnis Academy.

Revision 1.1

Credits and Copyrights

Revision Record

Revision	Description
1.0, Feb 2018	Initial Release by Andreas Pfeiffer
1.1, Jan 2019	Update for Omnis Studio 10 by Andreas Pfeiffer

Table of Contents

Part 1: First steps and creating the service staff remote form

Installing Omnis Studio Development Version

Omnis Studio is a tool for the development of Web- and Mobile applications: no other tool is necessary. You can download Omnis Studio Development version, including the latest version for Windows and Mac OS, at http://www.omnis.net/developers/downloads

Download the installer and install Omnis Studio. You can get a trial evaluation serial number at: https://omnis.net/developers/downloads/free-trial/

When you launch Omnis the first time you will be prompted to enter the serial number.

Omnis Studio folder structure

The installation on both platforms (Windows and Mac OS) is divided into two parts. The **program** directory and the **data** directory. The latter one contains all files where Omnis needs to have write permission.

The data directory can be found at:

- **Windows:** c:\users\%USERNAME%\AppData\Local\Omnis Software\ and then the Omnis Studio installation.
  Please note: You might need to enter the "AppData" string because the folder might be hidden.
- **Mac OS:** This folder is also hidden. Open the finder and click on the "Goto" menu. Then hit the "option" key. This should make the "Library" item appear in the Goto menu. Click on Library - Application Support - Omnis and then on the Omnis Studio installation.

There are a number of supporting folders there. Here are just the most important:

- **Studio** - contains libraries for the Omnis Studio IDE and other files to support the IDE. i.e. config.json contains some basic settings of Omnis Studio. The omnis.cfg contains the serial number that you entered as well as the position of the IDE windows.
- **Startup** - contains libraries that will automatically start when you launch Omnis. It contains a number of additional libraries for the IDE. You can use this folder in the Omnis Studio Application Server to start your own library automatically.
- **Html** - the most important folder for the web- and mobile development in Omnis Studio. You would need to put this folder and the subfolders into the htdocs (Apache) or inetpub (IIS) folder when you want to deploy your app using a web server later on. It also contains the images folder where you can put your own images for the application.

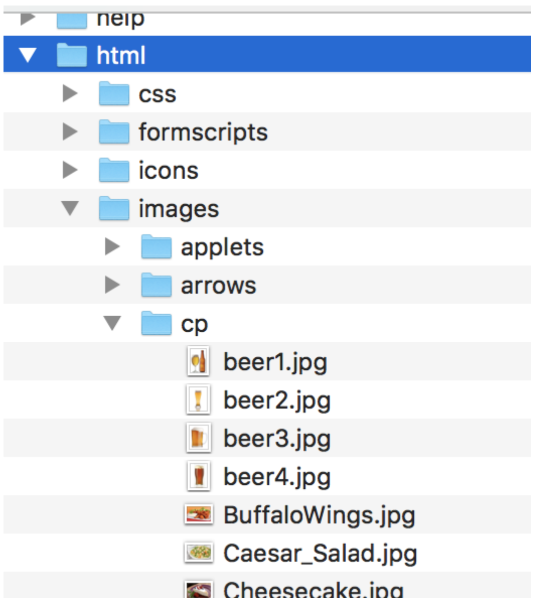
Example files

Unpack the cucina_piccola.zip file. It contains the following:

- Cp folder - contains all pictures needed for the app
- Cp_final.lbs - final version of the project
- Resource.lbs - Omnis library that contains example code to be copied during this course.
- Cucina_piccola.db and Cucina_piccola.db-journal - SQLite database with example data

Exercise 1 - Copy picture resources

- Copy the CP folder that contains all the pictures for Cucina Piccola into the /html/images folder of your Omnis installation in the data directory.



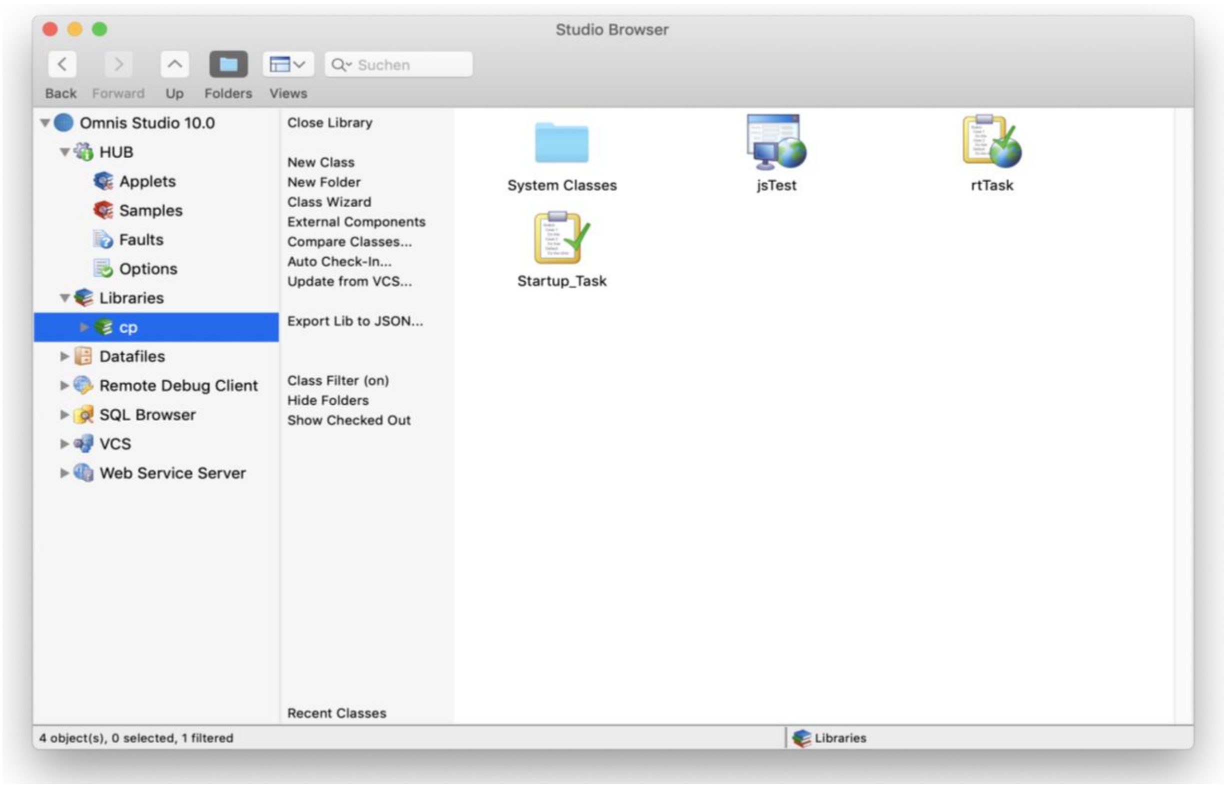
Folder structure

Exercise 2 - Start Omnis and create project

1. Launch the Omnis Studio Developer Version. Please enter your company name, name and serial number when prompted.
2. You should then see the **Studio Browser** window (in some versions of Omnis you may see a welcome screen – just close this).
3. Click on **Libraries** option in the tree list and then on **New Library** next to the tree list - then on **Blank Library** in the wizard window.
4. A file dialog will appear and you should navigate to your Cucina Piccola exercise folder (where the cucina_piccola.db files are) and name the new library: **cp.lbs**
5. This will create a new Omnis project – a so-called “library“ and also open the library so that you can see the library in the **Studio Browser** window on the left side in the tree list.

Studio Browser

❖ You can use F2 (Windows) or Cmd-2 (Mac OS) to quickly open the Studio Browser if it is closed for some reason.

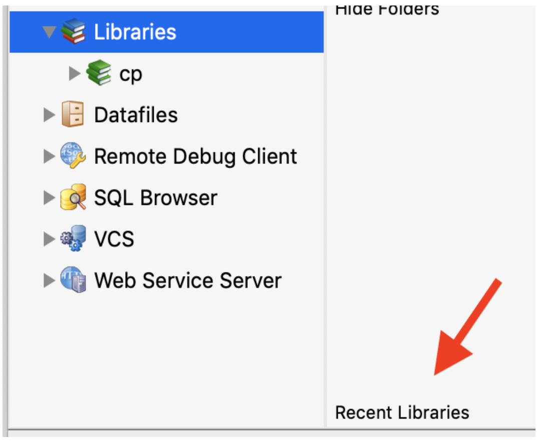


The Omnis Studio Browser (F2)

The Studio Browser allows you to maintain your libraries and classes inside the libraries. The middle section (next to the tree) allows you to create new classes and folders when a library is selected in the tree.

This middle section changes its links depending on the object that is selected in the tree. For example when you select the **Libraries** node in the tree you are able to create a new or open an existing library.

❖ There is a link on the button to open the recently used libraries:

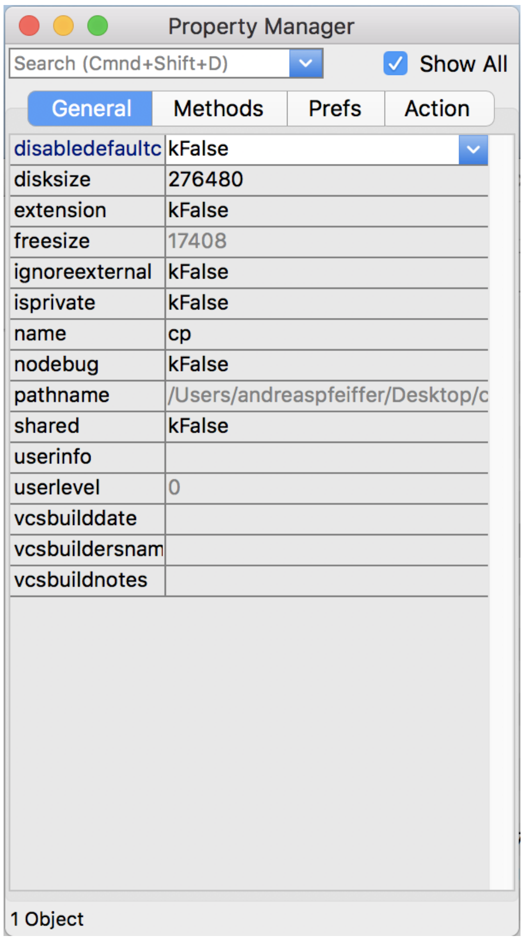


Recent Libraries

Property Manager

The **Property Manager** is context sensitive and shows the available properties of a selected library, class or component etc. You can either use the context menu on the item or use F6 (Windows) or Cmd-6 (Mac OS) to open it.

Make sure you have the checkbox **Show All** checked in order to see all properties. They are distributed into logical pages of a tab pane. (i.e. General, Methods etc.)



disabledefaultc	kFalse
disksize	276480
extension	kFalse
freesize	17408
ignoreexternal	kFalse
isprivate	kFalse
name	cp
nodebug	kFalse
pathname	/Users/andreaspfeiffer/Desktop/c
shared	kFalse
userinfo	
userlevel	0
vcsbuilddate	
vcsbuildersnam	
vcsbuildnotes	

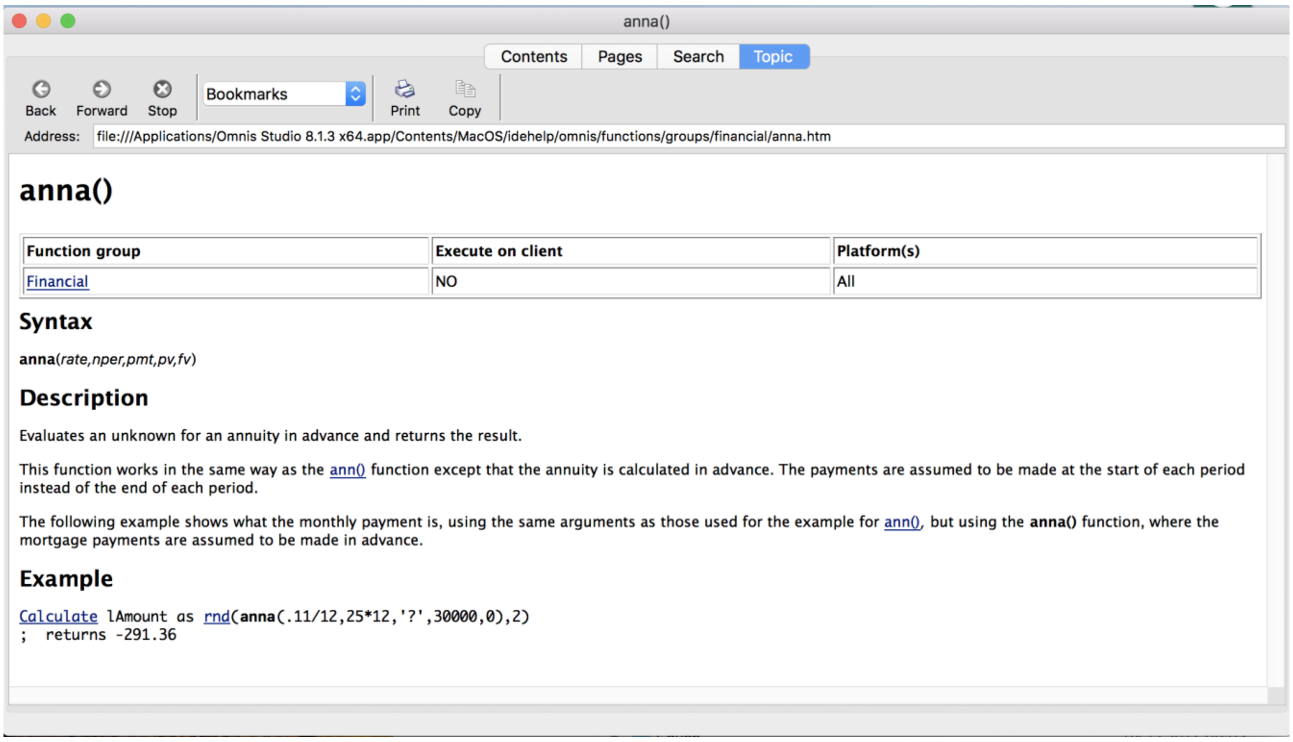
Property Manager (F6)

- ❖ It has a Search Box that allows you to quickly find the property on all panes if you know the name of it. Try it out!

Help

The Help Window can be opened with F1 (Windows) or Cmd-1 (Mac OS). It allows you to search for a keyword and explains all functions and commands.

❖ You can use Shift-F1 (Windows) or Shift-Cmd-1 (Mac OS). This will turn your cursor into a question mark. Then click on any item i.e. in the Catalog or in the Studio Browser to get the help for this item.


anna()
Contents
Pages
Search
Topic
Back
Forward
Stop
Bookmarks
Print
Copy
Address:
file:///Applications/Omnis Studio 8.1.3 x64.app/Contents/MacOS/idehelp/omnis/functions/groups/financial/anna.htm

anna()

Function group	Execute on client	Platform(s)
Financial	NO	All

Syntax

anna(*rate,nper,pmt,pv,fv*)

Description

Evaluates an unknown for an annuity in advance and returns the result.

This function works in the same way as the ann() function except that the annuity is calculated in advance. The payments are assumed to be made at the start of each period instead of the end of each period.

The following example shows what the monthly payment is, using the same arguments as those used for the example for ann(), but using the **anna()** function, where the mortgage payments are assumed to be made in advance.

Example

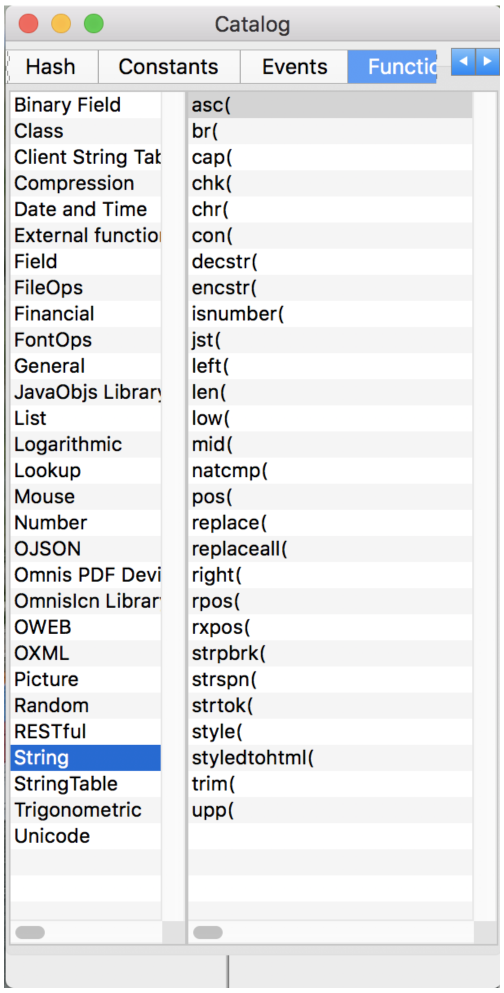
```
Calculate lAmount as rnd(anna(.11/12,25*12,'?',30000,0),2)
;  returns -291.36
```

Help (F1)

Catalog

The Catalog can be opened using F9 (Windows) or Cmd-9 (Mac OS). It is used to list variables, schemas, functions, constants and much more. You can either drag&drop the item from the Catalog into the calculation box of your code or you can double click on it when the cursor is in the calculation box in order to get the item into the code line.

❖ Use Shift-F1 (Windows) or Shift-Cmd-1 (Mac OS) and click on the function to get detailed help on it.



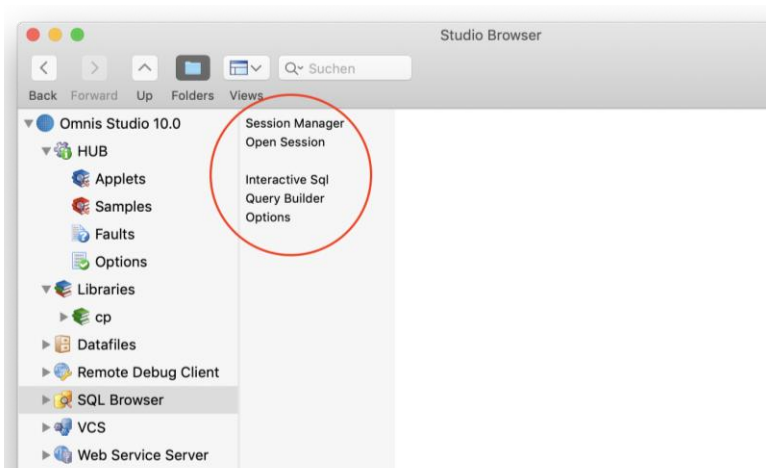
Binary Field
Class
Client String Tat
Compression
Date and Time
External functio
Field
FileOps
Financial
FontOps
General
JavaObjs Library
List
Logarithmic
Lookup
Mouse
Number
OJSON
Omnis PDF Devi
OmnisIcn Librar
OWEB
OXML
Picture
Random
RESTful
String
StringTable
Trigonometric
Unicode

asc(
br(
cap(
chk(
chr(
con(
decstr(
encstr(
isnumber(
jst(
left(
len(
low(
mid(
natcmp(
pos(
replace(
replaceall(
right(
rpos(
rxpos(
strpbrk(
strspn(
strtok(
style(
styledtohtml(
trim(
upp(

Catalog

SQL Browser

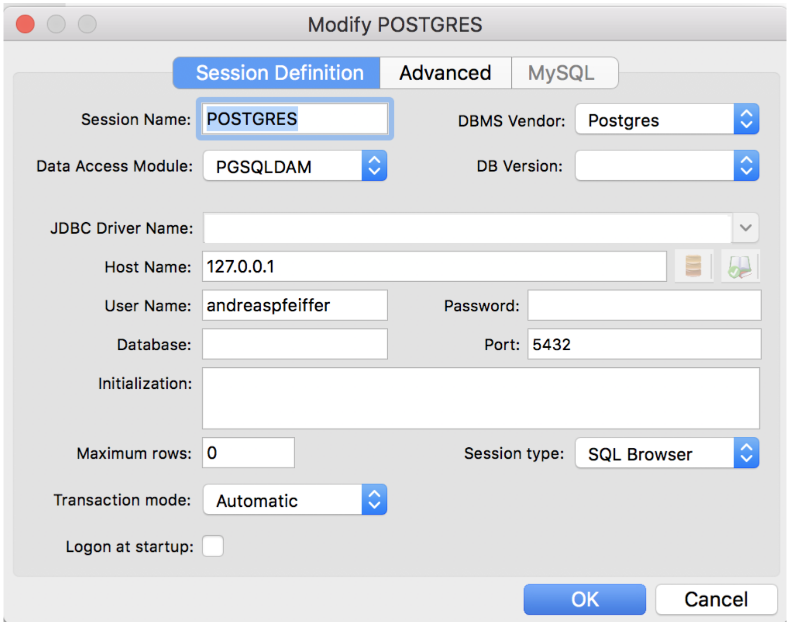
The Omnis Studio Browser has a node in the tree list “SQL Browser” that lets you open connections to your database.



SQL Browser Links

.

You will find links to the Session Manager that lets you define a new database session or modify existing session definitions.



Modify POSTGRES

Session Definition | Advanced | MySQL

Session Name:	POSTGRES	DBMS Vendor:	Postgres
Data Access Module:	PGSQLDAM	DB Version:	
JDBC Driver Name:			
Host Name:	127.0.0.1		
User Name:	andreaspfeiffer	Password:	
Database:		Port:	5432
Initialization:			
Maximum rows:	0	Session type:	SQL Browser
Transaction mode:	Automatic		
Logon at startup:	☐		

Session

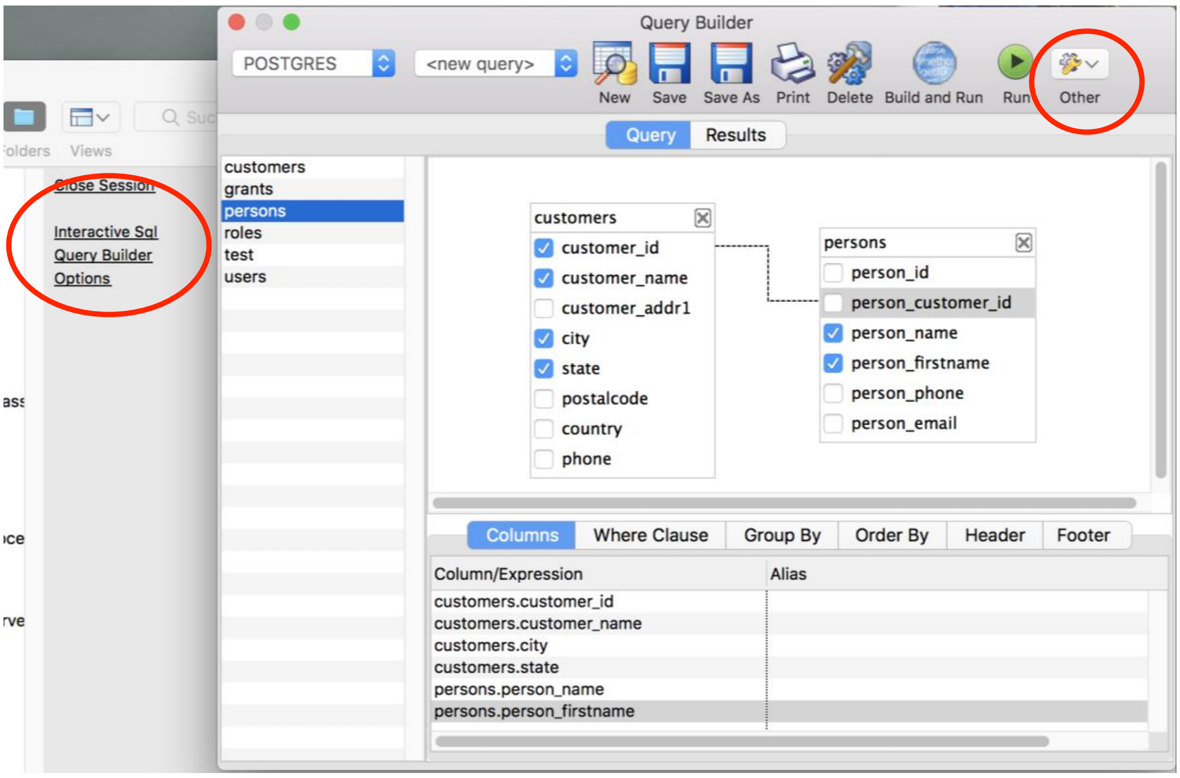
Once you have entered the session definition you can open the database session (click on the back to see the open link and select your defined session)

You will then be able to browse all database tables, views, triggers (whatever the database supports) in the SQL Browser.

You can use the context menu on the database table to run a SELECT statement or you can use the Interactive SQL window to run any SQL statement manually.

Query Builder

The Query Builder allows you to build select statements using one or several database tables. It lists all tables of the database session and you can drag tables from there to the design area on the right side. You can then check the columns you want to receive and link tables using a drag from the foreign key of the daughter table to the primary key of the parent table.



customers
- [x] customer_id
- [x] customer_name
- [ ] customer_addr1
- [x] city
- [x] state
- [ ] postalcode
- [ ] country
- [ ] phone

persons
- [ ] person_id
- [ ] person_customer_id
- [x] person_name
- [x] person_firstname
- [ ] person_phone
- [ ] person_email

Column/Expression	Alias
customers.customer_id	
customers.customer_name	
customers.city	
customers.state	
persons.person_name	
persons.person_firstname	

Query Builder

❖ You can change the join type using the context menu onto the link line.

When you hit “Build and Run” Omnis generates and fires up the SQL statement. You will then see the result on the “Results” pane.

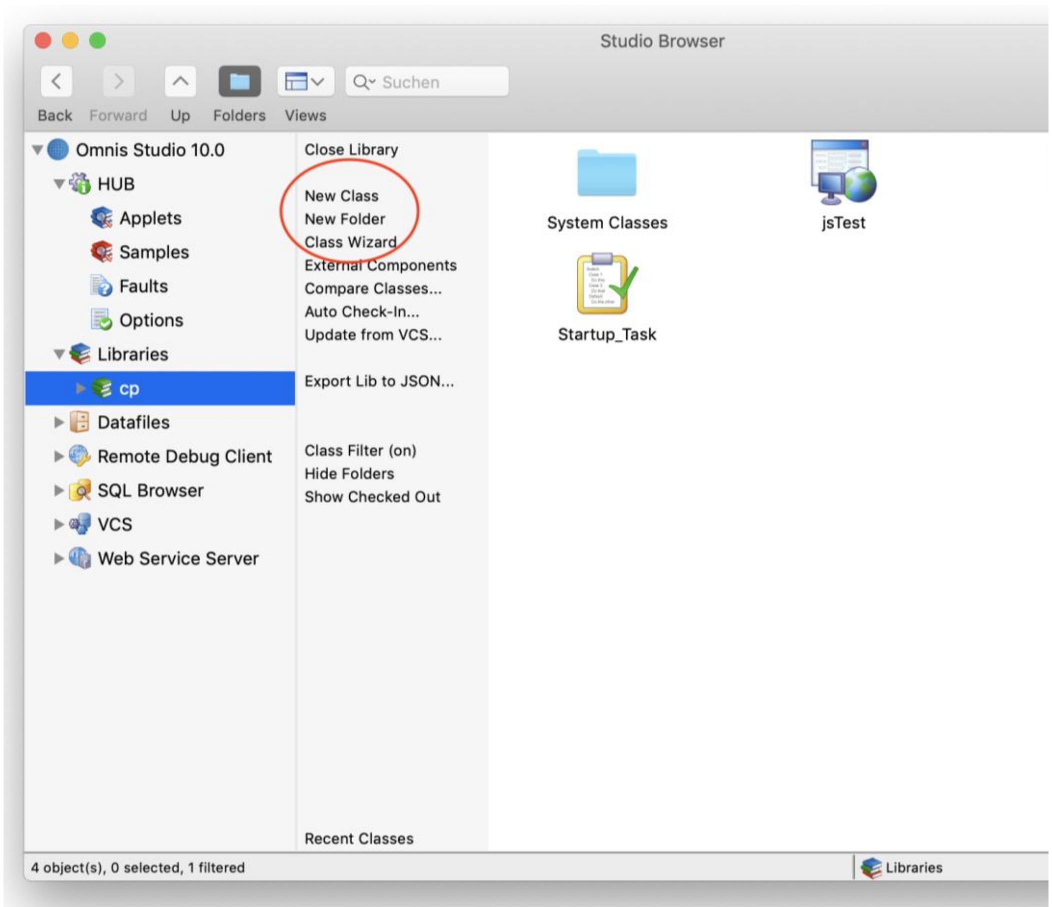
❖ The “Other” menu allows to export the data or to generate Omnis code in the clipboard in order to paste this code or opens a wizard (“Create table class..”) that allows to generate code into a new table class.

Exercise 3: Setting up the Library Properties

1. Select the library “CP” in the Omnis Studio Browser and either use the context menu or F6 (Windows) or Cmd-6 (Mac OS) to open the Property Manager.
2. Make sure you uncheck the “Show All” checkbox in the Property Manager.
3. When the library is selected the property manager shows all properties
4. In the PREFS pane you will find the “defaultname” property.
5. Set the “defaultname” to “CP”. This will insure that the library’s internal name is always “CP” even when the library is renamed on the hard disc.

Creating a new class

In order to create a new class or a folder inside your library you need to select the library in the Studio Browser first. You can then use the links next to the tree list or the context menu on the class area on the right to add a new folder or class.



Create class or folder

Class Types

There are a number of different class types in Omnis Studio. The most important for developing a Web- or Mobile App are:

- **Remote Task**
  Used to encapsulate the connection to the client. Normally you just need one Remote Task only. The Remote Task has properties that allow you to find out the IP address (clientaddress) or to set up an SSL connection (issecure) if your Web Server has a valid SSL certificate. The Remote Task stays open until the client closes the browser instance or if it times out (timeout). You can read all or set some of those properties. For example, if you set timeout to 30, the task is killed if the user stays idle for 30 minutes.

- **Remote Form**
  This is the class that allows to draw the UI for the application. It is a good practice to use several of those classes with not to many fields on it. You can use subform fields to nest remote forms. In addition you can use a subform field and dynamically change the classname in order to build a menu that controls different remote forms to be shown in one subform field.

- **Table Class**
  This class has inbuilt methods such like $select, $update, $insert etc. If internal methods are used you should bind the table class to a schema (or query) class. You can put your own select statements and business rules into this class. It allows to separate any complex SQL from the UI so that the methods can be reused in different context. A table class will be instantiated using a list or a row variable. That way a list or a row does not only contain data but also methods. It becomes the instance of the table class.

- **Object Class**
  This is another class that allows to encapsulate methods. You can use this class to implement objects that may contain more globally used functions. For example reading files, sending e-mails, parsing JSON, starting a timer etc. Object classes are similar to table classes but they don’t come with inbuilt methods. To instantiate an object class you would need to declare an object variable. You can assign its subtype to the object class. That way the object variable is used as the instance of the object class.

- **Report Classes**
  This class allows to print out your data to a printer. You can also determine different outputs like the PDF device. This is useful for web applications when the PDF shall be provided to the client.

- **Startup_Task**
  This class is global to the server side of a web- or mobile app. It controls any instance that will run on the server such as window classes. We could open a monitor window that shows all connections or add some code to the Startup_Task $construct method to build a session pool when the library and therefore the Startup_Task is opened.

Exercise 4: Adding a remote task

1. Create a Remote Task and call it rtTask.
2. Find the properties “clientaddress”, “issecure” and “timeout” of the remote task in the property manager.
3. Change the property “timout” to 30 minutes.

Exercise 5: Logon to the Cucina Piccola demo database

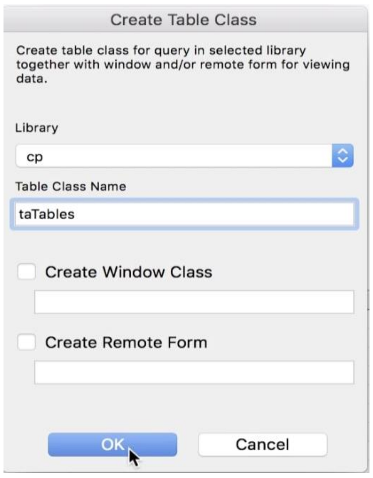
1. In the SQL Browser click on Session Manager and then onto New Session in order to define a new database Session.
2. Enter a name for the new session “CP”.
3. Choose SQLite as the database vendor and SQLITEDAM as the Data Access Modul.
4. There are two buttons next to the host name. Use the one to the right where the tooltip is “Select Datafile” and navigate to the Cucina Piccola training folder. Then choose the Cucina_Piccola.db file.
5. Confirm with OK to save these settings.
6. Click on “Back” and then onto “Open Session”. This should list the new CP database session.
7. Click on CP and the database session should open and become visible in the tree list.
8. You should then be able to double click onto “Tables” and find all database tables listed there. You can then use the context menu on any of the tables to show its data.

Exercise 6: Creating a table class using the Query Builder

1. Open the Query Builder
2. Drag “freeTables” server table to the design area.
3. Select both columns with the check box.
4. Click on “Build and Run”
5. Modify the generated SQL statement in the following form (Note: the two pipe symbols concatenate the string “table” with the column “table_id”):

```
SELECT
    'table '||table_id as tablename,
    CASE table_taken
      WHEN 1 THEN 'guests on table'
      ELSE  'table free'
    END as status,
    1 as icon
FROM freeTables
```

6. Test your statement using the button “Run” and correct any type errors.
7. If your result is satisfying then use the “Other” button to create a new table class “taTables” into your CP library.


Create Table Class

Create table class for query in selected library together with window and/or remote form for viewing data.

Library: cp

Table Class Name: taTables

☐ Create Window Class

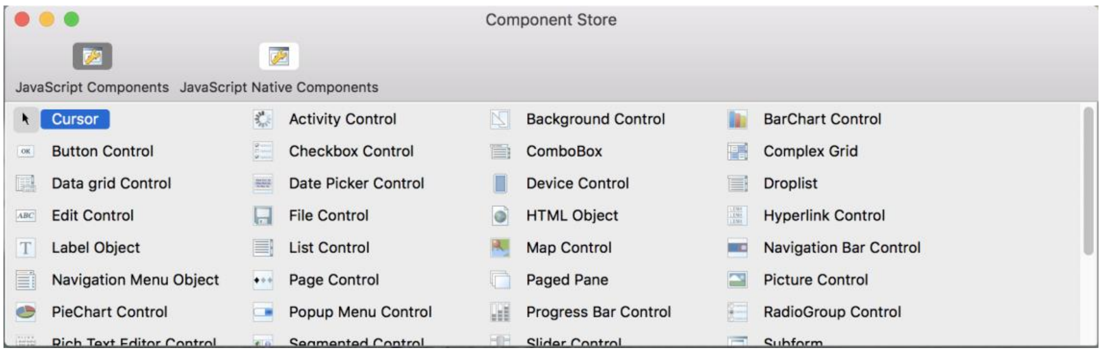
☐ Create Remote Form

Create a table class from the query

8. You can then close the Query Builder. You will be prompted if you want to save the query. Press “No”.
9. Go to your CP library and inspect the new table class taTables. The $construct method of this class contains a line of code that ensures that the instance of this class will use the database session “CP” that we have opened in the Omnis Studio developer version. In addition there is a method $load that contains the generated SQL code. It also has a line that executes the statement against the database session object and fetches the data into the current instance.

Designing the Remote Form

Double click on a Remote Form to open its design window. You will see tab panes on top of the design window that represents the so called “Layout Breakpoints”. You can switch between them in order to show different screen resolutions. You can add or delete those Layout Breakpoints using the the plus or x sign on it but we do not recommend to have more than 4. The fields you place on the remote form can have different size and position for each of the Layout Breakpoint. There is a property “animatelayouttransitions” in the “action” group of the remote form that allows to animate the fields when the screen size changed; i.e. the user tilts the device.



Component Store

You can add fields to the remote form from the **Component Store** that automatically pops up when you open the window. Alternatively you can use F3 (Windows) or Cmd-3 (Mac OS) to open it. Just drag&drop the required field from the Component Store onto your Remote Form. Alternatively you can select the field in the Component Store and then draw a rectangle on the form. This will create the component in the drawn size. After adding fields you would then typically change the fields properties as the field is already selected.

❖ You can customize the component store using the context menu on it. For example to switch on the text. Make sure you save the settings using another context click: Save Window Setup. This ensures that the changes are permanent.

Here are the most important properties of the fields:

- **name:** please put in a meaningful name, i.e. “orderBtn”, “street” etc. This property might be used to address the field later on.
- **dataname:** For all data driven fields such as entry fields, radio buttons, data grids etc. this property is used to bind the field to an instance variable.
- **edgefloat:** Allows the field to automatically resize or float when the screen size changes. This allows to use one Layout Breakpoint for mobile phones of different size. For example iPhone and iPhone Plus and also different Android phones would use the same Layout Breakpoint. With this property you can determine that the border of a grid becomes floating and therefore it will resize the grid for the larger phone size automatically.

  ❖ When you open the droplist for the edgefloat property there is a checkbox “Set for all layout breakpoints” at the bottom of the list that allows to set the edgefloat for all Layout Breakpoints in one go.

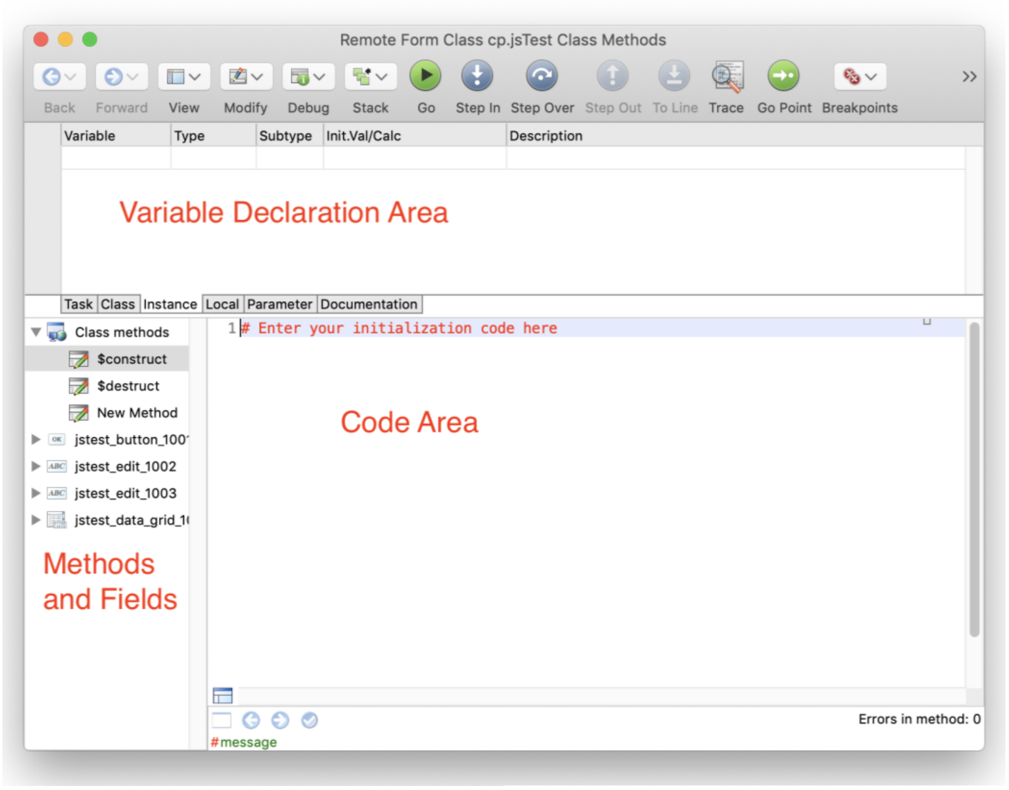
- **text:** Some fields such as buttons or labels have a text property that allows to enter text to be displayed on that field.
- **visible:** If you need to hide the field you can set this to kFalse.
- **visibleinbreakpoint:** Allows to hide a field for a specific Layout Breakpoint.

- **left, top, height and width** are the size and position of the field.
- **event:** Allows to receive any of the listed events. You would need to add code to the field event method to do any action on the received event.

  - ❖ You can test at any time using Ctr-T (Windows) or Cmd-T (Mac OS) or use the context menu on the background of the form. The web browser should open and display the designed form.

The Method Editor

You can open the Method Editor (Omnis code editor) for the Remote Form either using the context menu (choosing “Methods” or “Class Methods”) on the class, or double click on the background or on one of the fields of the Remote Form. In the latter case you would land in the $event method of the field. Otherwise you would be in the $construct method of the class.



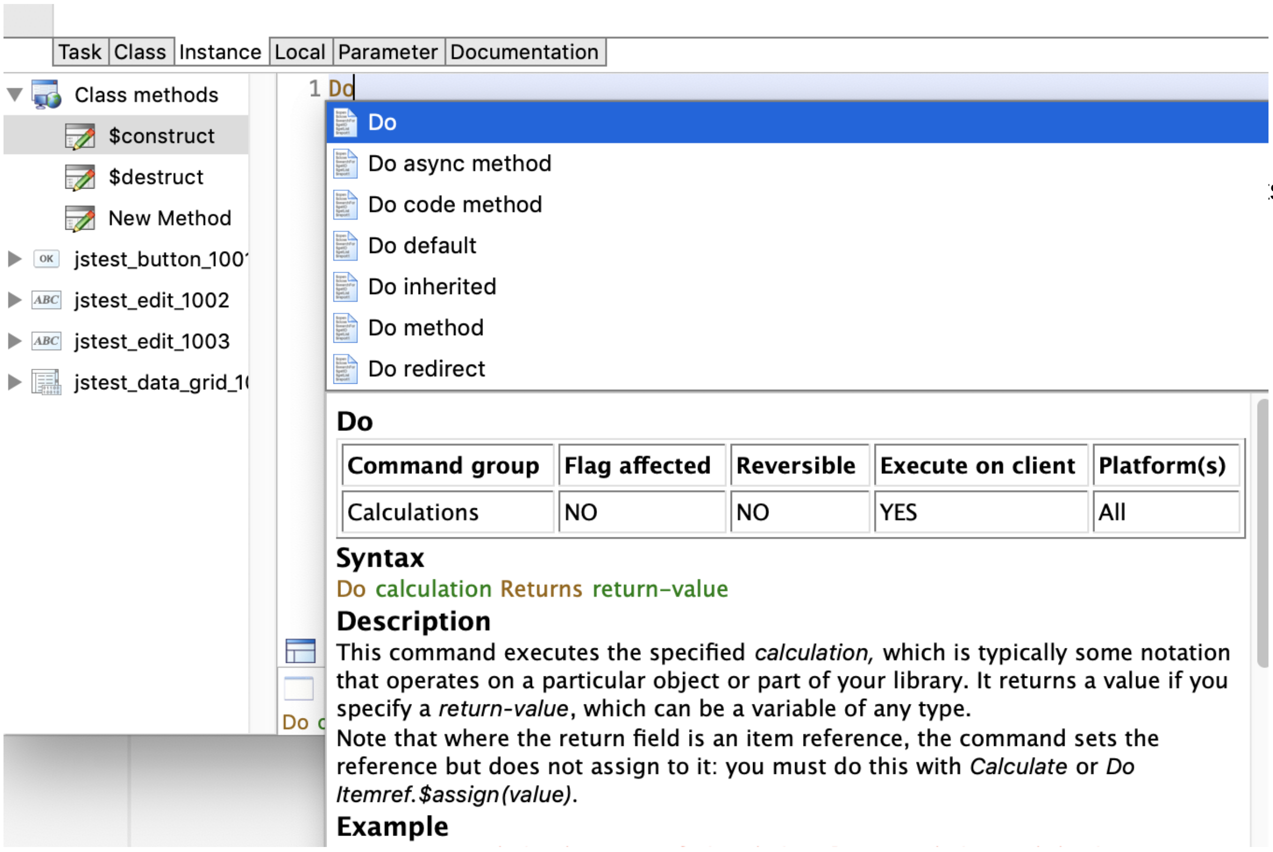
```
# Enter your initialization code here
```

Method Editor

If you have named your fields you will find them again in the Method and Fields area. Each field normally has a $event method by its own but can have additional methods.

The Variable Declaration Area allows to declare variables. You would first choose the scope using the tab pane underneath this area and then type in the variable name, type and subtype. For data that you want to bind to the dataname property of a field you must use an instance variable.

To enter code first select a line in the Code Area. Then start typing the command you want to enter: e.g. “d”. The **Code Assistant** will appear and show you the available commands that start with “D“. You may continue typing to specify a smaller find result in the code assistant. When you use the cursor keys to navigate to a command in the Code Assistant you will find more help on the specific command.



Do

Command group	Flag affected	Reversible	Execute on client	Platform(s)
Calculations	NO	NO	YES	All

Syntax

Do calculation Returns return-value

Description

This command executes the specified *calculation,* which is typically some notation that operates on a particular object or part of your library. It returns a value if you specify a *return-value*, which can be a variable of any type.

Note that where the return field is an item reference, the command sets the reference but does not assign to it: you must do this with *Calculate* or *Do Itemref.$assign(value)*.

Example

Code Assistant

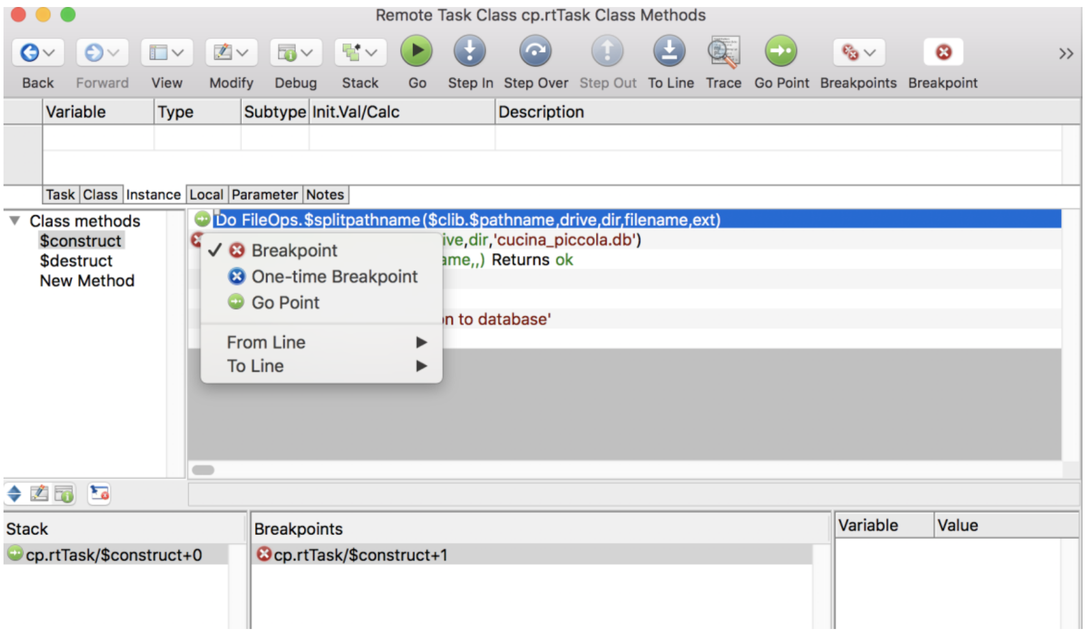
Exercise 7: Building the Service Staff Remote Form

1. Create a new Remote Form and name it “jsService”
2. Open the Methods by using the context menu on the new Remote Form.
3. Add an instance variable “iTableList” by typing this into the first box of the variable declaration area. Then set the type to become “List” and then set the subtype to become the new table class taTables.
4. Now enter the “Do” command in the code are of the $construct method.
5. After the „Do“ command please enter the name of the variable „iTableList“. While you start typing the variable it will be listed in the Code Assistant. You may want to use the tab key to quickly take it from the list.
6. Then add a dot and a dollar sign. You should now be able to see and pick the $load method that from the list.
7. Use Shift-8(Windows) or Shift-Cmd-3 (Mac OS) or double click on the remote form in the Studio browser to open its design window.
8. In the Component Store go to JavaScript Native components and drag the Native List Control onto the remote form.
9. Find the “dataname” property for this field and type in “iTableList” which should appear when you start typing.
10. Change the name property to “grid”.
11. Go to the “Data” pane in the Property Manager and assign the following values:
    accessorytypecol = 3
    text1col = 1
    text2col = 2
12. Test the Form (either using Ctr-T (Windows) or Cmd-T (Mac OS) or the context menu on the background - Test Form. The Form should appear in the Web Browser and display the data. The 3rd column (accessorytypecol) uses column 3 of the list which is always 1 and therefore shows the arrow icon. This icon indicates that the user will be able to navigate to the next level.

Part 2: Enhancing the service remote form and showing the pictures

Debugging in Omnis Studio

The Debugger



```
Do FileOps.$splitpathname($clib.$pathname,drive,dir,filename,ext)
ive,dir,'cucina_piccola.db')
ame,,) Returns ok
n to database'
```

cp.rtTask/$construct+0

cp.rtTask/$construct+1

Debugger

The **Debugger** in Omnis Studio is very powerful. You can set Breakpoints in order to stop the code execution when you click on the free space on the left side of the code line, the buttons in the ribbon bar or using the ribbon bar “Debug” menu. There are also keyboard shortcuts available. For reference look in the Debug menu.

There are different types of breakpoints available. The red one will stay in the code as long as you do not remove it. It will disappear when you close the library. The blue breakpoint is a one time breakpoint - meaning it will automatically disappear after the first use. This one can be set using a Cmd/Cntrl-Click left to the code line. In addition you can hardcode a breakpoint using the command “Breakpoint”. That one never disappears as long as you do not delete it in your code, but it will only work in the Omnis Studio development version. Finally there are breakpoints on variables that you can enable using the context menu on a variable. For example you can make the code stop when the content of the variable changes or reaches a certain limit.

If you want to debug your code you would need to place one of the breakpoints (i.e. the red one) into your code and then start your application by testing your remote form. When the code reaches the breakpoint it will bring Omnis to front and you can then step through your code using on of the “Step …” buttons or short cut keys for stepping (see the Debug menu).

❖ You can start to debug and open an execution stack by using shift-click on the left side of the code line. This will work as long as you are not using any instance or task variable in this code. The reason for this is that it does not create an instance of your class.

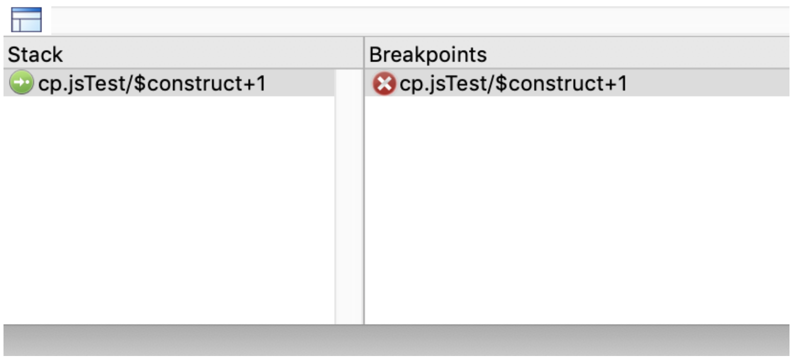
You will see a green ball with a white arrow when your Debugger hits the breakpoint the first time. This is what we call the Go point. That line will be executed the next time you hit one of the Step buttons or the Go button.

So how to debug the code from the first breakpoint?

- Go: This button executes all the next code lines until it hits another breakpoint or reaches the end of the method stack. It is recommended to use this button as the last action in order to resolve the stack.
- Step In: Steps and executes every single line. If a method call is reached Step In will step into the subroutine.
- Step Over: Steps and executes every line. It executes subroutines too but does not step into it.
- Step Out: Steps out of a subroutine and stops on the next line of the outer method.
- To Line: You can select a line underneath, execute all lines in-between the Go point and the selected line and make the Debugger stop at the selected line.

Once you start debugging you will see the method stack on the bottom of your editor. For each subroutine it will add another line in the method stack and you can navigate between them by clicking on a stack line. You can clear the method stack using the context menu.

❖ You can set the Go point to another code line if you want. For example if you make changes to your code while you debug you can then set the Go point to the line that you have changed and then you can re-execute this line again. For editing the code while debugging there are small buttons above the stack list that allows you to switch between edit mode and stack mode.



Stack	Breakpoints
cp.jsTest/$construct+1	cp.jsTest/$construct+1

Stack List

Automating the Logon

Currently we are using the SQL Browser to do the database logon. However this is part of the Omnis Studio Development version and we do not have this available in the application when it is distributed. For this reason we would need to develop a logon method that needs to be executed when a new user is connecting.

When the user logs into the first remote form instance Omnis will automatically encapsulate this instance within the Remote Task instance. When an instance is about to be instantiated its $construct method is automatically executed. Therefore the $construct method of the Remote Task is the very first method that is executed.

For the new logon code we would need the host name which is for the SQLite database just the pathname to the Cucina_Piccola.db file. Because this file is in the same folder as the library we can write a little bit of code to find out the pathname of the current library. We can use the shortcut “$clib” which refers to the the current library and add the property $pathname. Hence $clib.$pathname would return the complete pathname to the library.

However we would need to split this into the pathname to the folder and adding “Cucina_Piccola.db” to it. For this we can use the function “FileOps.$splitpathname” and the string function “con” which you both find in the Catalog.

Exercise 8: Writing the Logon method

1. Double click onto the Remote Task rtTask to open its method editor.
2. Make sure you are in its $construct method and add the command “Do”.
3. We need to split the pathname of the current library. Therefore please enter the following after the Do command. You will get help from the Notation Helper while you type:

   ```
   FileOps.$splitpathname($clib.$pathname,drive,dir,filename,ext)
   ```

   Note: You will notice that the variable names that you entered have a red wavy underline. This shows that there might be a type error or in this case the variables are not declared yet. You can fix this using the little check button at the bottom of your editor. This will open a prompt that allows you to declare the variable on the fly. Make sure you do this for all variables that you added in the code: “drive”, “dir”, “filename” and “ext”. Please confirm each of them to become local character variables.

   Note: You will find the new variables inside the Variable Declaration Area in the Local section.
4. Add another local variable “hostname” of type character by adding another line in the Variable Declaration area.
5. Now shift-click on the left side of this line in the code area to start the Debugger. You should now see the Go point at the code line.
6. Click on Step In or Step Over in order to execute this line. You should then see the path when you hover the mouse over the “dir” variable in the code.
7. Add another line by using Ctr.-N (Windows) or Cmd-N (Mac OS).
8. Type “C” in order to get the “Calculate” command. This is typically used to do an assignment. You can use the tab key to pick the „Calculate“ command as it is the first in the list.
9. Then type „h“ and the Code Assistant will show the new „hostname“ variable that you can choose with the tab key, too.
10. An additional tab will accept the „as“ and set the cursor to the end of the command so that you can enter the following function: con(drive,dir,’cucina_piccola.db’)

    You should now have this code in that line:
    ```
    Calculate hostname as con(drive,dir,'cucina_piccola.db')
    ```
11. Now Step over this line and you should see the calculated hostname when you hover over the “hostname” variable.
12. Either click Go or clear the method stack using the context menu on the stack list.
13. Declare a Task variable “tSessionObj” of type object. As a subtype select the droplist in the subtype column and navigate to the SQLITESESS inside the External Objects group.

    Note: This will be the session object variable that we are going to use to connect to the database.

14. Now add the following into the next line:
    ```
    Do tSessionObj.$logon(hostname,,) Returns ok
    ```

Note: The two commas represent the parameters for user name and password. They are mandatory for the $logon method but should be empty because SQLite does not support users.

15. The return of the “Do” command will be a new local variable: “ok”. You are supposed to declare this variable using the little check button at the bottom. Please select “boolean” as type.
16. Finally enter the following code lines:

```
If ok
Else
  Quit method 'could not logon to database'
End If
```

17. Close the editor for the remote task.

Exercise 9: Allow the data object to use the new DB session

1. Select the table class “taTables”
2. Find the property “designtaskname” and assign the remote task there by using the droplist next to the property.

   Note: this ensures that the task variable becomes visible at design time.

3. Double click on “taTables” to open its editor.
4. You will find code in its $construct method that assigns the session object to the current instance. It uses the global session that we opened in the IDE. Replace this session using the task variable instead. Your code should then look like this:

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

5. Test your “jsService" Remote Form. If you do not see any data or if you receive the error message use a (red) breakpoint to debug your code.

The advantages of using a super table class

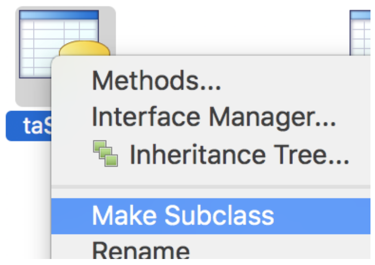
The construct of our table class uses code to assign the session object to the current instance which is in case of a table class either a list or a row variable - actually a data object. Assuming that all of our data objects shall make use of the same database session we would need to repeat this for each new table class that we add to our project. Also what if we want to implement common behavior for all data objects such as writing into a “insert_date” or “user_name” column of the table?

To avoid duplicate code we can simply use a super class that inherits all methods to the other table classes.

How to inherit a class?

There are two ways to do this. If you have a super class and you want to to create a new inherited class from it you can simply right mouse click on that class in the Class Browser and select “Make Subclass” in its context menu. In this case Omnis tries to inherit as much methods and properties as it can.

Alternatively you can just assign the $superclass property of the subclass to the name of the superclass.

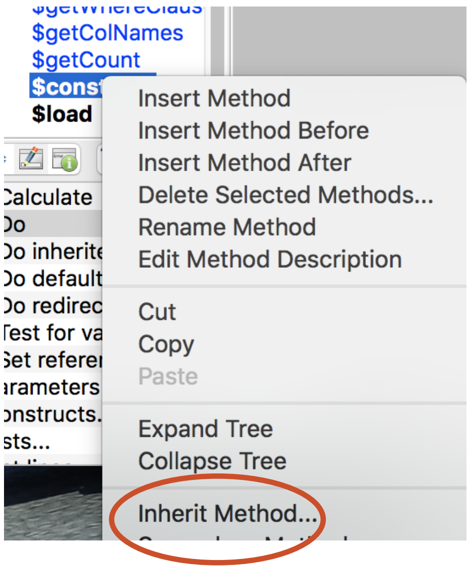


Make

How to override or inherit methods and properties?

The inherited methods and properties are displayed in blue color to indicate that they are determined from the super class. You can inherit using the context menu onto the method name or the property name. Vice versa you can override/overload a method or a property using the context menu on it.

❖ You can double click on a blue inherited method in order to jump to the super class method.



Inherit Method

Exercise 10: Introducing the super table class

We need to tidy up our library a little bit. For this reason we will make a folder that is going to contain data bound classes.

1. Create a folder in your library called “DatabaseLayer”.
2. Drag the table class taTables into the new DatabaseLayer folder.
3. Open the resource.lbs
4. Move the table class taSuper into the DatabaseLayer folder of your own library. This will copy the class into your library.
5. Now select taTables within your cp.lbs library and find the $superclass property.
6. Assign the name of the super class “taSuper“ to this property. (This makes taSuper the super class of taTables.)
7. Open the class editor for taTables (double click in the Browser on taTables).

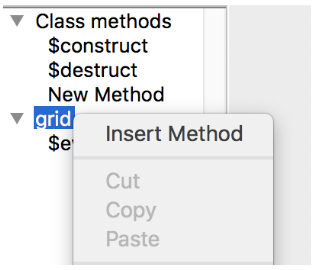
   Note: you will now see all public methods of the super class in blue. The $construct method is not inherited since it has had already code.
8. Inherit the method $construct: Right mouse click onto the method name and chose “Inherit Method”. You will be prompted if you want to delete the existing code. Confirm with “Yes”.
9. Inherit the property “designtaskname” of taTables.
10. Now copy all other table classes, the two schema classes “invoices” and “orders” as well as oNavigation from the resource.lbs into the DatabaseLayer folder of your own library.
11. Copy the system class #ICONS from the “System Classes" folder into the “System Classes” folder of your own library. (You will find some icons that we use)
12. Close the resource.lbs library.

Exercise 11: oNavigation object class

1. Check out the oNavigation object class. Double click to see its methods.
2. Select $getGroupList. You will see that it returns a nested list since the variable “mainList” is using “subList” as a column.
3. Check out $getProductList. This one returns a list of products.
4. Check out the declaration of the local list variables. You will see that it uses methods of the imported table classes. This method loads products for each product group and returns a nested list too.

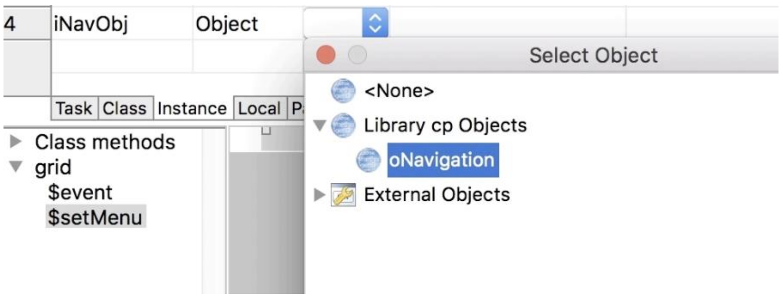
Exercise 12: Writing code for the grid component

1. Open the method editor of jsService.
2. Find the “grid” component in the method editor and add a new method to the grid component using the context menu on the name of the component within the method list. Name it “$setMenu”.



Adding a method to a field

3. Declare a parameter “pOffset” in the declaration pane of the method $setMenu. The Type should be Integer, and Subtype 32-bit Int. This will be used to navigate to the next level (+1) when the user clicks on an item in the grid or one level back (-1) when the user clicks on the “back” button.
4. Add an instance variable “iMenuLevel” - Short Integer. This will be used to hold the current menu level. Set the initial value to “1”.
5. Add “iHeader“ instance variable of type “character”. This will be used to display a header text. For example the table name that has been chosen in the highest level.
6. Add “iNavObj“ instance variable of type “object”. As a subtype select oNavigation within your library.



Assigning the subtype of an object variable

7. Add another instance variable “iCurrentListName” of type character.
8. Add an instance variable “iGroupList” of type list - no subtype needed.
9. Add an instance variable “iProductList” of type list - no subtype needed.
10. Add an instance variable “iOrderRow” of type row based on the table class (subtype) taOrders.
11. Add an instance variable “iSummaryList” of type List - no subtype needed.
12. Set the focus into the code area of the method editor and add the following code lines:

```
Calculate iMenuLevel as iMenuLevel+pOffset
Switch iMenuLevel
Case 1
  Calculate iHeader as ''
  Calculate iCurrentListName as 'iTableList'
```

13. Next line: Case 2

14. Next line: Calculate iHeader as con(’Table ’,iTableList.$line)
15. Now calculate iCurrentListName as ’iGroupList’ using quotes around “iGroupList”.
16. Next line: Do iNavObj.$getGroupList(iTableList.$line) Returns iGroupList

    Note: iTableList.$line is the current line number of the iTableList.
17. Next line: Case 3
18. Next line we need to find out if we are in the first group. Enter the following:
    If iGroupList.$line=1
19. Next line: Do iNavObj.$getProductList(iGroupList.subList.$line=2) Returns iProductList
20. Next line: Calculate iCurrentListName as “iProductList“
21. Next line: Else
22. Next line:
    If iGrouplist.subList.$line=1

    Note: This is to check if the “summary” button has been used
23. Next line: Do iOrderRow.$getGroupedList(iTableList.$line) Returns iSummaryList
24. Now calculate iCurrentListName as ’iSummaryList’ using quotes around “iSummaryList”.
25. Next line: End if
26. Next line: End if
27. Next line: Default
28. Next line: Calculate iMenuLevel as iMenuLevel-pOffset
29. Next line: End Switch
30. In the next line assign a new data name to the grid field:
    Do $cinst.$objs.grid.$dataname.$assign(iCurrentListName)

For your convenience here is the complete code that you should have entered by now:

```
Calculate iMenuLevel as iMenuLevel+pOffset
Switch iMenuLevel
  Case 1
    Calculate iHeader as ''
    Calculate iCurrentListName as 'iTableList'
  Case 2
    Calculate iHeader as con('Table ',iTableList.$line)
    Calculate iCurrentListName as 'iGroupList'
    Do iNavObj.$getGroupList(iTableList.$line) Returns iGroupList
  Case 3
    If iGroupList.$line=1
      Do iNavObj.$getProductList(iGroupList.subList.$line=2) Returns iProductList
      Calculate iCurrentListName as 'iProductList'
    Else
      If iGroupList.subList.$line=1
        Do iOrderRow.$getGroupedList(iTableList.$line) Returns iSummaryList
        Calculate iCurrentListName as 'iSummaryList'
      End If
    End If
  Default
    Calculate iMenuLevel as iMenuLevel-pOffset
End Switch
Do $cfield.$dataname.$assign(iCurrentListName)
```

Event Handler

You can implement event handler for every field. This needs to be a method $event that you will find right at the field name within your editor for the remote form.

Every $event method starts with the “On” command using the event constant, i.e. “evClick”, “evDrop” etc. The following code will be executed when the selected event happened.

- ❖ The field itself has an “event“ property. Remember to activate the event that you want to use. evClick for buttons is activated by default.

Exercise 13: Writing the event handler

Now we need to call this code when the user clicks on the grid.

1. Go to the $event method of the grid field.
2. There should already be an “On” “evClick” already in the first code line.
3. In the next line please enter:
   Do $cfield.$setMenu(1)

   Note: $cfield (current field) refers to the grid and thus it will find the $setMenu method. The parameter is the offset that tells the recipient method the offset 1: We want one level deeper.

4. Now we need to switch on the event for the grid field. Use F3 (Windows) or Cmd-3 (Mac OS) to open the design window of the remote form and go to the properties of the grid. There is an “event“ property where you need to switch on the evClick event.

   Note: By default events are mostly deactivated. This is to minimize the event traffic between the Browser and the Omnis App Server.

Exercise 14: Adding more fields to the service form

We can use a “paged pane“ component as a container.

1. Open the remote form jsService in design mode. If you are still in the method editor you can use Shift-F8 (Windows) or Cmd-Shift-8 (Mac OS) to open it.
2. Add a PagedPane control from the Component Store to the remote form.
3. Assign its name property to “top_container”.
4. Open the droplist of the “edgefloat” property in the Appearance group. Select the checkbox “set for all layout breakpoints” and chose kEFPosnTopToolbar. This will make the container stick to the top of your remote form.
5. Change the height of the container in every layout so that a button will easily fit in.
6. Select the grid component and assign kEFPosnClient for all layout breakpoints to the edge float of the grid.
7. Add an Edit control in the middle of the container. This field is going to display the iHeader variable. So please enter this into its dataname.
8. Center the field within the container on each layout.

   Note: If the field is not visible in a smaller layout you might want to use the field list (context menu on the form) to select the field and assign the “left” property using the Property Manager.
9. Add a button control on the left side of the edit field that will become the “back” button.
10. Let us call it “backBtn” and assign “back” as the button text.
11. Double click onto the button in order to open the Method Editor. This takes you to the $event method of the button.
12. After On evClick please enter the following in order to execute the $setMenu method of the grid:

    ```
    Do $cinst.$objs.grid.$setMenu(-1)
    ```

Exercise 15: Test your form

Now you should be able to test your form. You may want to add a Breakpoint to the $event method of the grid in order to debug the code and inspect the variable. Make sure you do not add the breakpoint at the “On” command but rather the next line after the “On” command.

Adding text labels and pictures to the grid

You might have noticed that there are no text labels on the buttons within the grid and also no product pictures yet.. The values for those are actually already loaded in the list but we cannot see them because we need to determine the columns of the list that have those information.

Exercise 16: Enabling the properties for the icons and text labels

1. In the remote form jsService select the grid field and change the following properties (in the Data pane):
   accessorycontent=5
   accessoryvalue=6
   imagecol=4

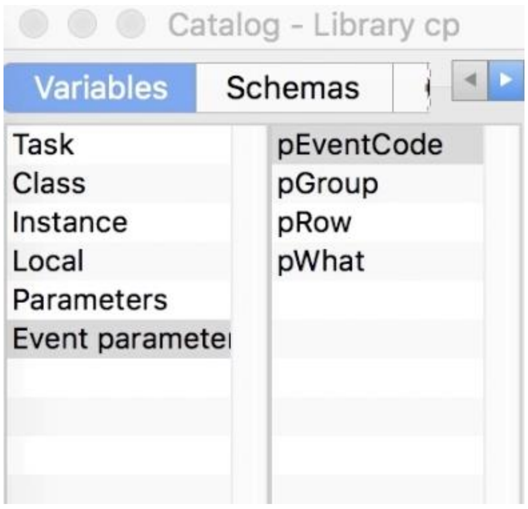
   Note: these values point to the corresponding columns in the list that is attached because of the $dataname property.

2. Test again. You should now see the pictures for the group in the 2nd level and text for the checkout button. And in the 3rd level - i.e. when you click on the “Drinks” group - you should be able to see pictures for each product and also text on each order button.

Event Parameters

Event methods can have its own event parameters. If you select a code line in a $event method of a field right after - not on the On evClick for example you can use the Catalog F9 (Windows) or Cmd-9 (Mac OS) and look at the variable pane. There you will find a group “Event Parameter”.

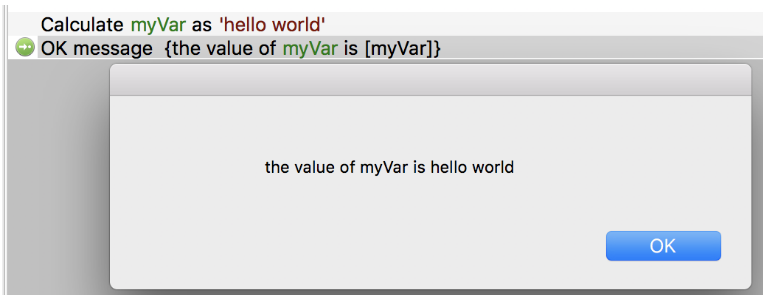
- ❖ When debugging you can inspect the contents of those event parameters either hover over that parameter or using the variable context menu item on it.



Event Parameters in Catalog

Square Brackets

Square brackets allow you to force evaluation of a variable. For example when Omnis expects to receive text only you can mix this text using a variable in square brackets.



```
Calculate myVar as 'hello world'
OK message  {the value of myVar is [myVar]}
```

the value of myVar is hello world

Square brackets in Ok message

jst() Function

The jst() function can be used to format a text string. Please refer to the manual http://developer.omnis.net/documentation/functionref/index.jsp?detail=jst.html#jst or the Help Window to see its rich functionality.

For the use of this course we would need the “P” argument that will fill the rest of the string with the following argument. Also we will use the minus in order to get this filled on the left side.

For example:

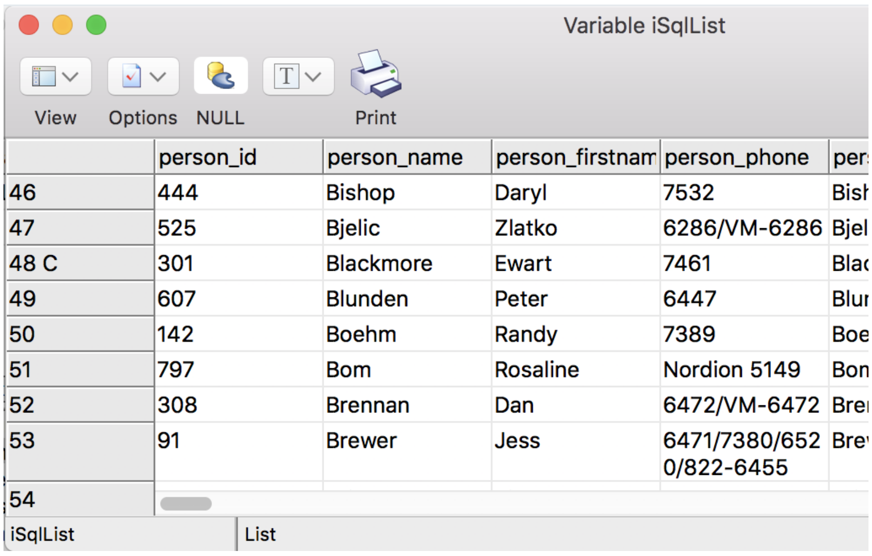
jst(999,’-5P0’)   will be converted into: 00999

List Notation

List variables are structured variables using columns and list lines. The counting of the list lines start with 1.

You can access the content of a cell within a list variable using the name of the list followed by the line number and the column name (in this order) separated by a dot.

For example the following picture shows the contents of the list variable iSqlList.



	person_id	person_name	person_firstnam	person_phone	per
46	444	Bishop	Daryl	7532	Bish
47	525	Bjelic	Zlatko	6286/VM-6286	Bjel
48 C	301	Blackmore	Ewart	7461	Blac
49	607	Blunden	Peter	6447	Blur
50	142	Boehm	Randy	7389	Boe
51	797	Bom	Rosaline	Nordion 5149	Bon
52	308	Brennan	Dan	6472/VM-6472	Brei
53	91	Brewer	Jess	6471/7380/6520/822-6455	Brev
54					

iSqlList List

List variable

iSqlList.52.person_name would return “Brennan” since this is in line 52 in column person_name.

In some cases a line can become the current line because the user might have clicked on a line of a list component where the list is assigned as its dataname or because you did a search on the list or simply assign $line of the list: Do iSqlList.$assign(48)

As you can see there is a “C” right at the line number 48 of the example list. This tells you that this is the current line. Or in other words: iSqlList.$line=48

If you need to refer to a cell of the list within the current line then you can simply omit the line number: iSqlList.person_name would return “Blackmore” in this example.

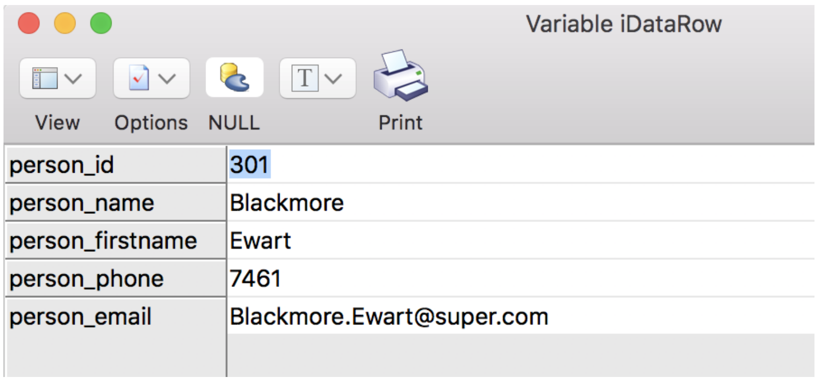
You can also use square brackets if you have either the list number or the column name within a variable or parameter. For example if the row number is in a parameter “pRow” you can use: iSqlList.[pRow].person_name or alternatively assign $line (the current line) using the value of pRow.

You can also access a cell without knowing the column name if you know the column number. In this case you can use a “c” as a prefix to the column number: iSQLList.c2 would refer to column two of the current line in iSQList. Note: The “c” is mandatory in order to distinguish from the line number.

Row Notation

A row variable is simply a list with only one line and the line is always current. Therefore you should omit the line number: Do iDataRow.person_name.$assign('Duck') would assign “Duck” into the column “person_name” of the row variable “iDataRow”.

Please note that when debugging, the row is displayed vertically.



person_id	301
person_name	Blackmore
person_firstname	Ewart
person_phone	7461
person_email	Blackmore.Ewart@super.com

Row Variable

So in this example iDataRow.person_name would return “Blackmore”.

Exercise 17: React on the order button

1. Go to the $event method of the grid.
2. Add some lines between “On evClick” and “Do $cfield.$setMenu(1)”
3. Right under “On evClick” enter the following:

   ```
   If pWhat=kJSNativeListPartAccessory
   ```

   Note: this is to find out if one of the order buttons in the list are clicked on
4. Add an empty line below.
5. Add an “Else” command.
6. After the “Do $cfield.$setMenu(1)” add an “End if” command.
7. Between “If” and “Else” please enter the following:

   ```
   Switch iCurrentListName
     Case ’iProductList’
       Do iProductList.$line.$assign(pGroup)
       Do iProductList.subList.[pRow].amount.$assign(iProductList.subList.[pRow].amount+1)
     Case ’iGroupList’
       # Checkout
   End Switch
   ```

   Note: If one of the buttons in the list is clicked we need to find out wether it is the “checkout” button in the iGroupList or if it is the “order” button in the iProductList. For the latter one we are going to count the amount in the nested sublist.

8. In order to get feedback in the form when the user clicks on one of the order buttons we should also display the amount on the order button. For this please add a local variable “myString” of type character.
9. Then add a line of code bevor the Case ‚iGroupList‘:

   ```
   Calculate myString as iProductList.subList.[pRow].amount
   Do iProductList.subList.[pRow].buttontext.$assign(myString)
   ```

10. Test your code. When clicking on an order button the button text should display how often the button has been clicked on. Unfortunately the width of the button changes due to the length of the string.
11. In order to fix this we will use the jst() function. Please change the last line we entered in the following way (changes are in bold):

    Do iProductList.subList.[pRow].buttontext.$assign(**jst(**myString**,’-4P0’)**)

12. Test again to see the changes.

For your convenience here is the complete code that you should have entered by now in the $event method of the grid:

```
On evClick
  If pWhat=kJSNativeListPartAccessory
    Switch iCurrentListName
      Case 'iProductList'
        Do iProductList.$line.$assign(pGroup)
        Do iProductList.subList.[pRow].amount.$assign(iProductList.subList.[pRow].amount+1)
        Calculate string as iProductList.[pGroup].subList.[pRow].amount
        Do iProductList.[pGroup].subList.[pRow].buttontext.$assign(jst(string,'-4P0'))
        Do $cinst.$objs.orderBtn.$visible.$assign(kTrue)
      Case 'iGroupList'     ## click on the checkout button
    End Switch
  Else
    Do $cfield.$setMenu(1)
  End If
```

Exercise 18: Hide the header if not needed

We do not need to display the back button or the header if we are in the first table showing the table list. Therefore we are going to hide the top_container when we are in the first level:

1. Go to the $setMenu method of the grid.
2. At the very end of the method enter:

   ```
   Do $cinst.$objs.top_container.$visible.$assign(iMenuLevel>1)
   ```

3. Now also set the $visible property of the top_container to kFalse because we don’t want it to be shown initially when the form first opens.
4. Test your form.

Part 3: Making the order button

Writing the record into the database

Exercise 19: Writing an event handler for updating the order

Now we need a button to send the orders that have been marked with the order buttons in the list into the kitchen. For this we are going to save this information into the database.

1. Add a button onto the top_container right next to the header field.
2. Assign “orderBtn” as the name and assign “order” as the text to be displayed.
3. Make the button initially invisible by setting the “visible” property to kFalse.
4. Make sure the positioning is correct in all Layout Breakpoints.
5. Double click on the button in design mode in order to open the $event method of this button.
6. Add a local variable “ok” of type boolean.
7. After the on evClick please enter a For Loop; we use iProductList.$line as the loop counter that will count up for every iteration of the loop. The start value is going to be “1” and the end value will be the total number of lines in the list: iProductList.$linecount:

   ```
   For iProductList.$line from 1 to iProductList.$linecount step 1
     Do iOrderRow.$save(iProductList.sublist,iTableList.$line) Returns ok
   End For
   If ok
     Do $cfield.$visible.$assign(kFalse)
     Do iTableList.$updateStatus(iTableList.$line,kTrue)
     Do $cinst.$objs.grid.$setMenu(-1)
   Else
     Do $cinst.$showmessage(iOrderRow.$getErrorText())
   End If
   ```

   Note: The $save method of the iOrderRow receives the nested sublist in the iProductList that contains the ordered amount. As a 2nd parameter it receives the table number that happens to be the line number of iTableList.

   If the order was successful we are making the button itself ($cfield) invisible to avoid to press this button again and we are going to update the table status using a new method $updateStatus which is not yet implemented. Hence it will not be shown in the Code Assistant when we start typing the code.

   $cinst.$showmessage is a way to popup a message in the browser. It uses the method $getErrorText

8. In the $event method of the grid we would then need to make the order button visible when one of the order buttons of the list is clicked on. So please add a line of code after assigning the button text (and before the next “Case” command):

   ```
     Do $cinst.$objs.orderBtn.$visible.$assign(kTrue)
   ```

Bind Variables

You can use Bind Variables in order to let the Data Access Object evaluate the content for your variable. For example when sending a SELECT statement in the WHERE clause you would need to put any text into single quotes:

```
SELECT * FROM CUSTOMERS WHERE NAME = ’Scotty’
```

So when having the string “Scotty” within a variable “MyVar” you would use square brackets for example to evaluate the variable:

```
SELECT * FROM CUSTOMERS WHERE NAME = [MyVar]
```

Unfortunately you would miss the single quotes as they are not part of the variable contents. Hence you would need to do something like this:

```
SELECT * FROM CUSTOMERS WHERE NAME = ’[MyVar]’
```

The better way is using a bind variable which is done adding a prefixed at sign right before the first square bracket:

```
SELECT * FROM CUSTOMERS WHERE NAME = @[MyVar]
```

This tells Omnis **not** to evaluate the variable but transfers this information to the Omnis Data Access Modul (DAM) you are using. That way the DAM decides how to format the variable content properly for the specific database.

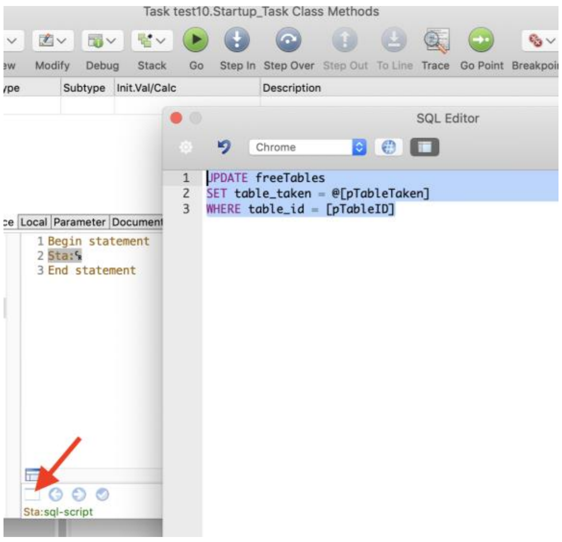
- ❖ This is especially useful when using date variables as the DAM uses the database specific data formation.

SQL Editor

In Omnis one can enter SQL code into a so called statement block. There are commands like „Begin statement“ and „End statement“. Between those two commands you have typically one or more lines that begin with „Sta:“

To make it easier to enter this code you can open the SQL editor when the cursor is at a line that starts with „Sta:“. Just click on the little button on the left side and on the bottom of the editor.

When you then close the editor it will transfer the SQL into Omnis code and vice versa when you reopen the editor.



```
UPDATE freeTables
SET table_taken = @[pTableTaken]
WHERE table_id = [pTableID]
```

```
Begin statement
Sta:
End statement
```

SQL

Exercise 20: Writing code to update the status of the table

We need to develop the $updateStatus method that we called in the event handler method of the order button.

1. Create the method “$updateStatus” editor for the table class taTables. It should be in the DataLayer folder. You can double click on it to open the editor.
2. Add a new method “$updateStatus” underneath the $load method. You can use the context menu onto the method list for this.
3. In this method please enter two parameters:
   pTableID - Integer - 32bit
   pTableTaken - Boolean
4. Add a command line: “Begin Statement”
5. In the next line add the “Sta:” command. Note: You can use the little button on the bottom edge of the code window to open the SQL editor in order to enter the following SQL code. When you close the editor the SQL code will be automatically transferred into Statement commands in Omnis:

   ```
   UPDATE freeTables
   SET table_taken = @[pTableTaken]
   WHERE table_id = [pTableID]
   ```

   Note: Integer variables like pTableID do not need the usage of a bind variable because the server accepts numeric values anyway.

6. If the „End Statement“ is not already set please use the mouse pointer to add another line to avoid an automated “Sta:” command:

   ```
   End statement
   ```

7. The next line will check the SQL code and execute. If the preparation fails it will call the $sqlerror method. In addition it will switch the text in the list depending on the value of the parameter „pTableTaken“:

   ```
   If $cinst.$statementobject().$prepare()
       Do $cinst.$statementobject().$execute()
       Calculate $cinst.c2 as pick(pTableTaken,'table free','guests on table')
       Quit method kTrue
   Else
       Do $cinst.$sqlerror()
       Quit method kFalse
   End If
   ```

Note: the instance of the table class has an inbuilt $statementobject. Therefore we can refer to it using $cinst.$statementobject. The parentheses after the statementobject is used to resolve this part of the notation first. Then it uses a method there: $prepare().

$prepare uses the Statement block in order to check the syntax and it returns false if an error is detected. Therefore in case of an error we call the $sqlerror method that we have in the super table class taSuper.

If $prepare is successful the $execute of the statement object then sends the statement to the database server.

For your convenience here is the complete code that you should have entered by now in the $updateStatus method of the table class taTables:

```
Begin statement
Sta: UPDATE freeTables
Sta: SET table_taken = @[pTableTaken]
Sta: WHERE table_id = [pTableID]
End statement
If $cinst.$statementobject().$prepare()
  Do $cinst.$statementobject().$execute()
  Calculate $cinst.c2 as pick(pTableTaken,'table free','guests on table')
  Quit method kTrue
Else
  Do $cinst.$sqlerror()
  Quit method kFalse
End If
```

Exercise 21: Test your order button

You should now be able to write orders into the database. If one of the following steps fail you would need to place a breakpoint in your code and find out what is going wrong there.

1. Open the remote form in your browser
2. Select one of the tables.
3. Choose “Drinks” or “Food” and use the order button in the list to order items. This button supposed to show the number of orders. The main order button in the header should appear.
4. Then click the order button in the header. This should write the orders into the server table. The order button should disappear and the grid should show the the previous menu level.
5. Click on “Summary”. Now you should see the order that has been done for this table.

Part 4: Implementing the Corporate Identity and developing of a second remote form for the kitchen and bar staff

Implementing the Corporate Identity using a remote form as super class

As we are going to use more than one remote form we can use a super class remote form in order to implement the Corporate Identity (CI). All methods and fields that we implement in that class can than be inherited into other remote forms. That way any functionality and UI will be available in all inherited classes and we just need to implement this once.

Exercise 22: Making a form that displays the Corporate ID

We are now going to add a remote form that will display a picture within the /html/images path of the AppData (Windows) or Application Support (Mac OS) folder.

1. Create a now Remote Form “jsSuper” in your library.
2. Open jsSuper in design mode and add a Paged Pane control. This is used to allocate some space in the upper part of the form.
3. Call it “top_pane” in the name property.
4. Assign kEFposnMenuBar in the “edgefloat” property. Make sure you have selected “Set for all layout breakpoints”.
5. Add a picture control into the top_container at the left corner. Call it “picture”.
6. In the 320 layout breakpoint please choose a width and a hight of 70 for the picture component. Make sure that the top_container’s height is minimum 70.
7. For the larger layout breakpoint you can make the picture and the top_container a little bit larger since there is more space available.
8. Go to the “dataname” property of the picture component. Enter iPictureURL. When you hit enter you will be prompted if you want to declare this variable. The scope is fixed to “instance” because this is the only scope that can be used within the “dataname”. The type “character” is fine as well as the subtype which is the length. Don’t change anything there. But add the following into the Init Val/Calc field:

   ’images/cp/cucina_form_top.png’

   Note: This points to the relative path of your “html” folder. Remember you have copied the folder “cp” that contains also the picture “cucina_form_top.png” into the /html/images folder. You can replace this picture with your own png or jpeg file if you want.
9. Find the “noscale” property for the picture field and set it to kFalse. This way the picture will be scaled to the size of the picture field.
10. Check if “keepaspectratio” is set to kTrue. This will ensure that the aspect ratio of the picture is correct.
11. Set “disablesystemfocus” to kFalse. Otherwise it would put a frame around the picture when the user selects it.
12. Test the form. It should display the picture (if the path is correct) in different sizes when you resize the browser window.

Exercise 23: Making jsSuper become a super class

We will now make jsSuper become the super class of the remote form jsService that we worked on.

1. Close the design window and the method editor of jsService if it is still open.
2. Select jsService and go to the “superclass” property.
3. Assign “jsSuper” in the “superclass” property.
4. Open jsService in design mode. You will see that it contains the “top_container” including the “picture” field from the super class.
5. Test jsService. It should now display the CI there too.

Adding a remote form for the kitchen and bar staff

The order is now saved in the database from within jsService. Unfortunately, there is no remote form that lists all the open orders so that the kitchen staff can see them and provide the food or drinks.

Exercise 24: Adding the remote form for the kitchen and bar staff

We are now going to make a second remote form that will be inherited from jsSuper. It shall display all open orders and allow you to either display all, just the drinks or just the food orders. In addition it shall allow you to mark orders as “served”.

1. In the Omnis Studio Class Browser please right mouse click onto jsSuper and select “Make Subclass”. This creates a new remote form sub classed from jsSuper. Call it “jsOutlet”.
2. Add a “datagrid” component from the Component Store. It will be used to list the orders.
3. Name it “grid”
4. Type in “iDataList” into the “dataname” property. You will be prompted to declare this variable. The type shall be “List” and the subtype shall become: taOrderList
5. Double click onto the background of the form in order to open the class methods. You will notice that the inherited methods, i.e. $construct is shown in blue to indicate that it is part of the super class.
6. Add a new class method “$load”
7. Add a “Do” command to the $load method.
   Do iDataList.$load(iType)

   Note: You may notice that iType is underlined with a curly red line. That tells us that this expression is not valid - in that case the variable is not declared yet. You may want to set the cursor to this text either using the mouse or one of the arrow keys at the bottom and then use the little check button aside the arrow buttons. This opens a dialog that allows you to do the variable declaration. It shall be an instance variable of type Integer. You can choose “Short (0-255)”.
8. We want to execute the $load method of jsOutlet when the form opens. Therefore we would need to call this from the $construct method. Since that method is inherited we can override it. Therefore please select $construct of jsOutlet and open its context menu. There you will find the “Override Method”. Note: the method name becomes black and the method text is accessible.
9. Enter the command “Do inherited”. Note this ensures that the inherited code from the super class of this method is executed.
10. In the next line we enter:

    Do $cinst.$load()

    Note: This calls the $load method that we just wrote.
11. Now go to the design window, i.e. Shift F8 (Windows) or Cmd-Shift-8 (Mac OS)
12. Select the grid component and set the “designcols” property to 7. This will make the grid to display seven columns.
13. Set the “userdefined” property to kTrue. This way you have full control for each separate column.

    Note: One of the columns of the grid should now have a red selection frame in the design window. You can select either column to change properties for each one.

14. Select the first column and find the “column” pane in the Property Manager. Assign “iconpath” to the “columndatacol” property and “status” into “columnname”.
15. Now set the other columns accordingly:

Column	$columndatacol	$columnname
1	iconpath	status
2	order_table_num	table
3	order_date	order at
4	product_id	product
5	product_name	text
6	order_size	size
7	order_amount	amount

16. Test your form. You should now see the data in the grid but you might notice that the first column displays the pathname that refers to an icon but it does not display those yet. You would need to set the “columnstyledtext” property in the “column” pane of the first column in the grid to kTrue. Another test should now show red, green or grey bullets as the status icon.
17. Change the “height” property of the grid to 2500 for either Layout Breakpoint and resize the width and position so that it fits the size of the LayoutBreakpoint.
18. Set the “edgefloat” property to kEfRight. This will increase the grid automatically for intermediate sizes such as larger phones or if using a browser on larger screen sizes.
19. Change the “rowheight” property to 40. This allows easier usage if using touch screens.

Exercise 25: Setting a filter for the order list

In the last exercise we were adding an instance variable to determine whether all orders shall be listed or just drinks or just food.

1. Add a “radio group” component from the Component Store to the form between the top_container and the grid. You might want to move the grid a little bit down.
2. Name it “selector”
3. Put “iType” into the “dataname” property of the radio group.
4. In the “text” property please enter: “**All,Drinks only,Food only**”
5. Change “::horizontal” property in the Appearance pane to kTrue
6. Set “columncount” to 3
7. In the $event property enable the event evClick.
8. Double click onto the field and you should be in its $event method.
9. Below the line “On evClick” please enter:

   ```
   Do $cinst.$load()
   ```

10. When you now test it should be able to switch between all, only drink and only food orders.

Exercise 26: Marking orders as served

If an order is prepared by the staff it would be nice if they can mark individual orders as “served” so that the status is changed. For this we will implement an event handler on the grid.

1. Enable the evClick “event” for the grid in the Property Manager.
2. Double click onto the grid control in order to get to its $event method.
3. Delete all “On” commands but **not** the one with “On evClick”. We do not need any other event.
4. Open the Catalog to see the event parameter. There is the “pVertCell” event parameter that informs on which line the user has clicked on.
5. Below the “On evClick” we will enter:

   ```
   Do iDataList.$line.$assign(pVertCell)
   ```

   Note: this is to make the list in the current using the line number that we get from the event parameter.

6. We now need to find out that the user has clicked on the first column - the “status” column that contains the icon:

   ```
   If pHorzCell=1
     Do $cfield.$updateOrder()
   End if
   ```

   Note: $cfield is the reference to the same field - the grid where the $event method is too. Currently we do not have a method “$updateOrder” on that grid so you do not get help from the Notation Helper.

7. Add a method to the grid using the context menu on the “grid” in the method list. Name this method “$updateOrder”.
8. Add an instance variable “iOrder” of type “row” and the subtype “taOrders”.
9. Add the following line to the $updateOrder:

   ```
   If iOrder.$updateStatus(iDataList.order_id)
   ```

   Note: We can omit the line number of the list because we set the current line in the $event method of the grid.

10. Then add:

    ```
    Do $cinst.$load()
    ```

11. Add a comment. You can do this typing a „#“ sign. You may enter: “we will do a push here later”. This is to remind us that we will add code later on.
12. Add an “End if” command line.
13.

For your convenience here is the complete code of the $updateStatus of the grid field:

```
If iOrder.$updateStatus(iDataList.order_id)
  Do $cinst.$load()     ## reload this instance
  # we do a push here later
End If
```

Exercise 27: Fix a redraw issue

When testing the click on the icon you might notice that the grid scrolls to the end of the list. This is because the $load method of the class reloads the whole list and that makes the last line the current line. Hence the grid component tries to display the current line.

1. To fix this problem go to the $load method of jsOutlet.
2. Add a local variable “currentLine” of type Integer and subtype 32bit.
3. Add a new line above the “Do iDataList.$load(iType). You can do this when you select the line and press Ctr.-i (Windows) or Cmd-i (Mac OS).
4. Enter: Calculate currentLine as iDataList.$line

   Note: The local variable will be used to remember the line number before reloading the list.
5. After re-loading the list we can then assign the line number back:

   Do iDataList.$line.$assign(currentLine)
   or alternative:
   Calculate iDataList.$line as currentLine
6. Test your code. “Red” orders are not served yet. When you click on a red bullet it should change the status of your order in the database and reload the list and then display the order using a yellow bullet instead. If a yellow icon is clicked it will be reversed and the icon becomes red again. This allows to undo an action if the wrong order has been accidentally clicked on.

For your convenience here is the complete code of the $load method of the class jsOutlet:

```
Calculate currentLine as iDataList.$line
Do iDataList.$load(iType)
Calculate iDataList.$line as currentLine
```

Part 5: Adding a push service to synchronise all devices, printing a report and Installing the Omnis App Server

Push service

The push service allows to trigger a certain method $pushed of a remote form. To make an instance ready to receive any push you would need to enable this behaviour - typically you can do this within the $construct method of the remote form that wants to receive the push:

```
Do $cinst.$clientcommand('openpush',row())
```

- ❖ The 2nd parameter of the $clientcommand is mandatory but “openpush” does not need one. Hence you would pass in an empty row() function instead.

You can then reference the remote form instance and send a $pushdata(row()) message to this instance. Typically you would need to send a row variable as a parameter. This row variable can then be declared as a parameter of type row within the $pushed method of the recipient instance. The row variable is mandatory so if you do not need to send any parameters use an empty row() function.

You can use $iremotetasks.[nameOfTheTask].$iremoteforms.[nameOfTheForm] in order to reference to a specific remote form instance within a specific remote task instance.

- ❖ Alternatively if you want to send messages to all instances of a specific remote form you can use: $iremoteforms.$sendall($ref.$senddata(row))

Exercise 28: Enable the push service in jsSuper

In order to receive push notifications, we need to switch on both of our remote forms that either of them would be able to receive this push. Imagine we would automatically reload the order list in the jsOutlet form as well as reloading the summary list in the jsService form when an order status is changed in the outlet.

1. Add a new class method $load in the superclass jsSuper.

   Note: this will be used as a place holder in order to define our interface in the inherited classes.

2. Go to the $construct method of jsSuper and switch on the push service:

   ```
   Do $cinst.$clientcommand('openpush',row())
   ```

3. In the next line please enter:

   ```
   Do $cinst.$load()
   ```

4. Go to the $construct method of jsOutlet. We do not need any of this code there anymore. Hence please inherit the complete method using the context menu onto the method name and selecting “Inherit method”. You will be asked if you want to override this code. Please confirm. The method name of this $construct method appears to be blue now.
5. Open the method editor of the jsService remote form. Override the blue $load method there using the context menu on the method name. The method name should now be in black.
6. Go to the $construct of the jsService remote form. You will see that there is the command that calls the $load of the iTableList. Cut this line out of the $construct and paste this into the $load method that we have overridden.
7. Then inherit the $construct method as you have done before in jsOutlet.

Note: Both remote forms that are inherited from jsSuper now are ready to receive a push. In addition we have moved to load the initial data into the $load method of either class.

Exercise 29: Sending the push

Now we are ready to send the push.

1. Go to the “order” button of jsService. In the $event method right after we call the $setMenu(-1) of the grid component we add another line:

   Do $iremoteforms.$sendall($ref.$pushdata(row()))

   Note: $ref refers to each single instance that is covered by the $sendall method. So basically we are going to send a $pushdata to all of our remote form instances.

   Note2: Because we do not want to pass any information we can just send an empty row using the row() function.

2. Go to the $updateOrder method of the grid component within jsOutlet. Right after the “Do $cinst.$load()” command we add another line that also fires up the push:

   Do $iremoteforms.$sendall($ref.$pushdata(row()))

   Note: this will then update other outlet terminals as well as the remote forms for the service staff, i.e. the summary list.

3. Add a new class method in jsOutlet: $pushed

   Note: The method name has a pink color. This tells us that it is going to be a client executed method which is mandatory for the $pushed method.

4. Add a line of code:

   Do $cinst.$load()

   Note: So when a push is received the $load method is executed automatically and the data grid is re-loaded.
5. Now go back to the jsService remote form and add a new privat class method “loadSummary”.
6. At the grid component go to the $setMenu method and find the line “Do iOrderRow.$getGroupedList(iTableList.$line) Returns iSummaryList”. Copy this line into the “loadSummary” class method and replace it within the $setMenu with the command:

   Do method loadSummary

   ..in order to call this method.

   Note: The “Do method” command is separate from the “Do” command and is used to call private methods.
7. Now we can add a $pushed method as a class method of jsService and call the loadSummary from there too.
8. Refresh the browser for both remote forms in order to reload the client executed code.

Note: this is necessary because the client executed code is translated into JavaScript and therefore it would need to be reloaded from the browser when the code has been changed.

9. Test your work!

When you now do an order in the jsService remote form and press the main order button the order should magically appear in the jsOutlet remote form if this one is open.

Also when your service remote form displays the summary list for a table and in the outlet form one of the orders are marked as served. In this case the icon color of this order should change in the summary list of the other remote form.

Finally you could also open several instances of the remote forms and see those changes in either one.

Exercise 30: Marking an order as cancelled

You might have noticed that there is an undo button in the summary list of jsService. This button appears as long as the order has not been served. It allows the user to cancel an order but the outlet staff needs to confirm this.

1. Go to the $event method of the grid component within the jsService remote form.
2. We need to add another “case” for the “iSummaryList”:

   ```
   Case ’iSummaryList’
   ```

3. Then add a line to set the current line within the iSummaryList:

   ```
   Do iSummaryList.$line.$assign(pGroup)
   ```

4. Now we will use an indirect way to call the cancel request. Add the following code:

   ```
   If iOrderRow.$setCancelRequest(iSummaryList.sublist.[pRow].order_id ….
   ```

   Please note we get the product ID from the sublist that is nested in the summary list.

   Note the line number comes from the event parameter pRow and needs to be in square brackets to allow Omnis to evaluate this.

   The 2nd parameter is boolean and either sets the cancel request (0) or refuses the request (1). We can use the button text in order to find out the status:

   ```
   iSummaryList.sublist.[pRow].buttontext=‘redo’
   ```

   Here is the complete code line:

   ```
   If iOrderRow.$setCancelRequest(iSummaryList.sublist.[pRow].order_id,
   iSummaryList.sublist.[pRow].buttontext=‘redo’)
   ```

5. The next line is to assign a new value for the button text:

   ```
   Calculate iSummaryList.subList.[pRow].buttontext as
   pick(iSummaryList.subList.[pRow].buttontext=’undo’,’undo’,’redo’)
   ```

   Note: It uses the pick() function in order switch the value.

6. Now add a line that sends a push to all other clients:

   ```
   Do $iremoteforms.$sendall($ref.$pushdata(row()))
   ```

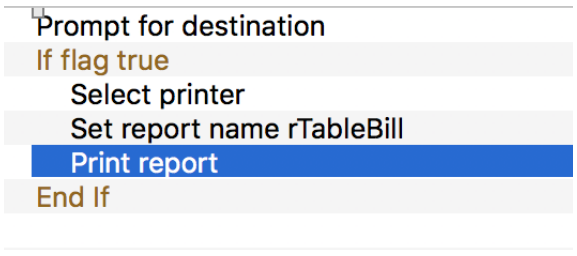
7. Add an “End If” command to close the If statement.
8. Reload the browser and test your form.

If all goes well you should now be able to make a cancel request in the summary list of the service remote form. The icon color of the order should become grey in both forms. The cancel request can now be confirmed from within the outlet form when clicking at the grey icon. The order will then be removed from all forms.

Adding a report

You can create report classes that allow you to print any data. In this example we are using a ready designed report but feel free to customise it if you want.

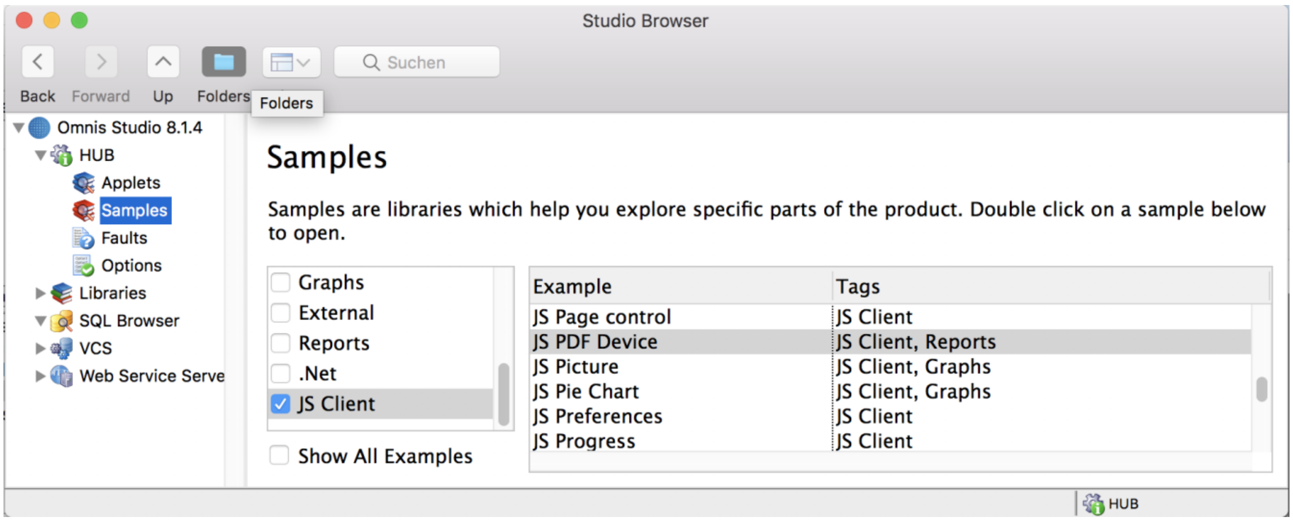
There are commands to set the destination where you want to print the report.



```
Prompt for destination
If flag true
    Select printer
    Set report name rTableBill
    Print report
End If
```

Print commands

Usually in a typical web application you would not print the report directly to the printer. Instead you would want to generate a PDF file and display this file within the application. This can be done using the JS PDF Device. There is also an example library available in the Samples section of the Omnis HUB.



Samples

Samples are libraries which help you explore specific parts of the product. Double click on a sample below to open.

- [ ] Graphs
- [ ] External
- [ ] Reports
- [ ] .Net
- [✓] JS Client
- [ ] Show All Examples

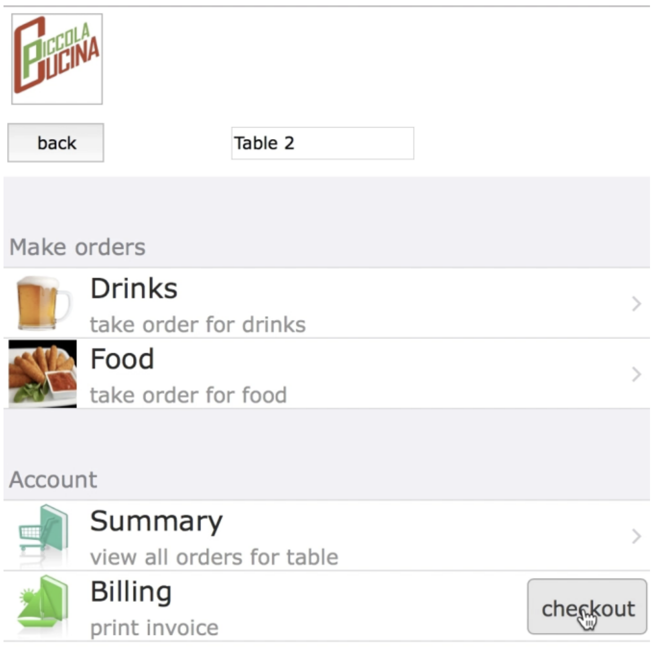
Example	Tags
JS Page control	JS Client
JS PDF Device	JS Client, Reports
JS Picture	JS Client, Graphs
JS Pie Chart	JS Client, Graphs
JS Preferences	JS Client
JS Progress	JS Client

JS PDF Device

For the Cucina Piccola app it is fine to print out the invoice directly on the server. We assume that the server is just a computer located in the restaurant that has a connected printer installed.

Exercise 31: Make the check out

Finally we would like to allow the customer to check out. The system shall automatically create the bill from the order list of the specific table. There is already a “checkout” button in the menu.



Menu with “checkout” button

1. Make sure you copy the report class “rTableBill” from the resource.lbs.
2. Add a new class method “checkOut” in jsService.
3. Add a new instance variable “iInvoiceRow” of type row and assign “taInvoices” as a subtype.

   Note: make sure you have this table class and the assistive schema class “invoices” copied from the resource library.

4. Add the following code:

   If iInvoiceRow.$insert(iTableList.$line)
5. The next line is going to update the table status:

   Do iTableList.$updateStatus(iTableList.$line,kFalse)

   Note: the 2nd parameter is to release the blocking status of the table.

6. Add another command “Set report name” and use rTableBill as the report name.
7. The next line is the command “Print report”. The asterisk tells Omnis to generate an instance name automatically. Then pass in invoice id as a parameter:
   Print report * (iInvoiceRow.$getIdent())

8. In the next line please execute a push:

   ```
   Do $iremoteforms.$sendall($ref.$pushdata(row()))
   ```

   Note: this is to see for other service forms that the order has been payed and the table is available again.

9. Now let’s add an “Else” command. This is to handle if the invoice row could not perform the $save for any reason.
10. Add a message to the client:

    ```
    Do $cinst.$showmessage(iInvoiceRow.$getErrorText())
    ```

11. Add an “End If” in order to close the statement.

Here is the complete code of the checkOut class method:

```
If iInvoiceRow.$save(iTableList.$line)
  Do iTableList.$updateStatus(iTableList.$line;kFalse)
  Set report name rTableBill
  Print report * (iInvoiceRow.$getIdent())
  Do $iremoteforms.$sendall($ref.$pushdata(row()))
Else
  Do $cinst.$showmessage(iInvoiceRow.$getErrorText())
End If
```

12. Go to the $event method of the grid within jsService.
13. If not yet done add another “case” for the checkout:

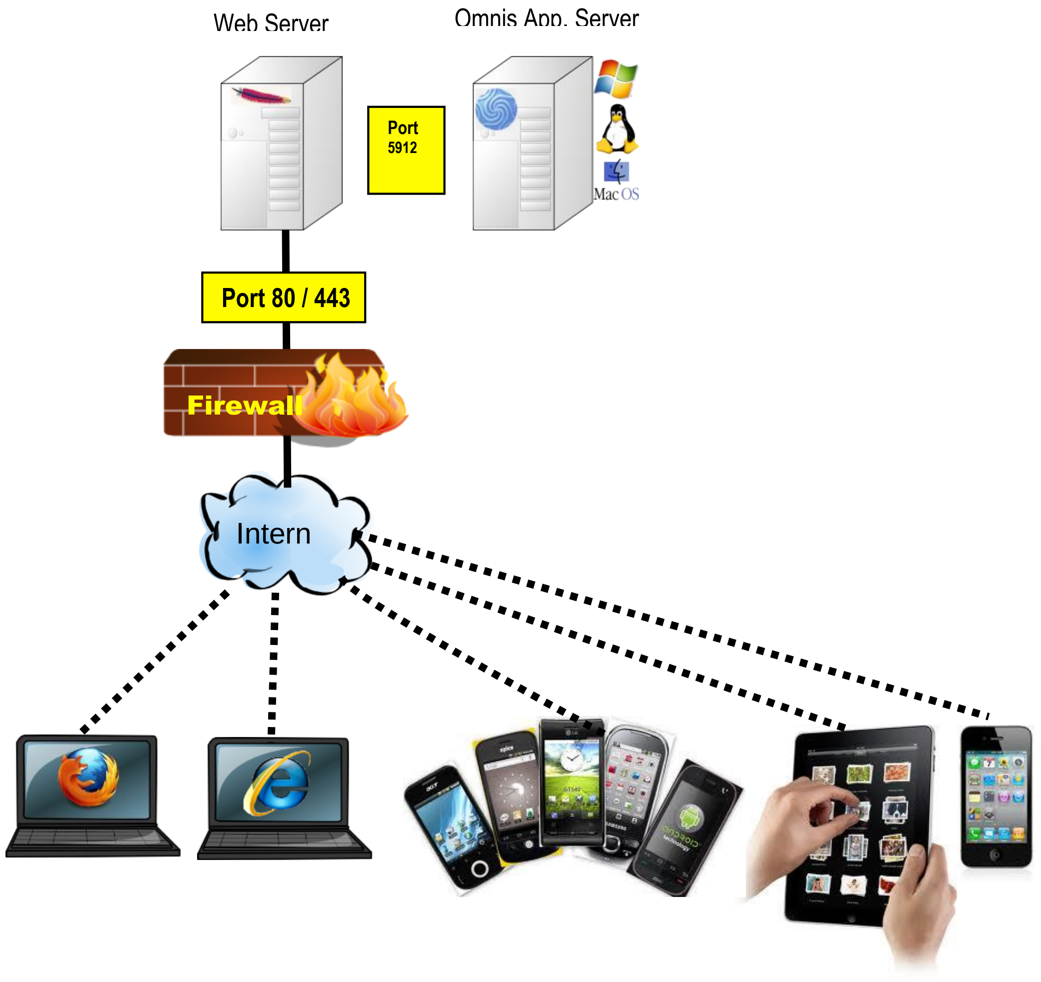
```
Case ‘iGroupList’
  Do method checkOut
```

14. Test your work now!

You should now be able to print invoices for a table that has orders. The icons for the orders in the outlet form should then change green and disappear from this list the next day. This table in the table list of the other service remote forms should then be shown as available.

Omnis Studio Web Architecture

The Omnis Studio Server Runtime (App Server) has an inbuilt HTTP server that lets Omnis communicate with a Web Server through an Apache or CGI module.



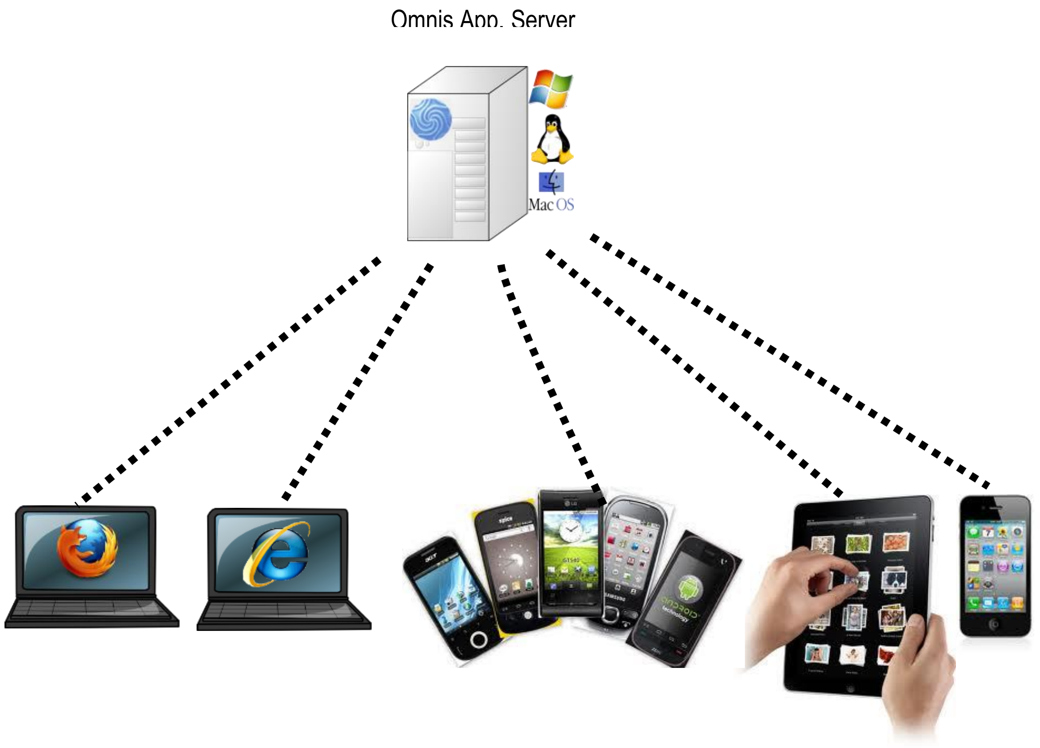
This architecture is best practice for web- and mobile apps that accesses the Omnis App Server through the internet.

You can scale the Omnis App Server using a load sharing process (LSP). In that case the LSP will receive the HTTP requests from the Web Server and distributes the request to the least busy Omnis App Server.

Note: Please follow the steps of the following technical note on our Developer Portal in order to configure the Omnis App Server alongside the web server:
http://developer.omnis.net/technotes/tnjs0003.jsp

Omnis Studio Intranet Architecture

For small in-house apps - just like the Cucina Piccola - we can use the Omnis App Server directly without using a “real“ web server. Omnis works then as an HTTP server and you can directly access it from within your local area network.



Exercise 32: Install and configure the Omnis App Server

If you want your Cucina Piccola app run within your restaurant without using the Omnis Studio *developer version* you would need to install the **Omnis App Server.**

Please note: For testing and debugging you can use the Omnis Studio development version, but for deployment you must use the Omnis App Server and purchase a different license.

1. Download and install the Omnis App Server from here: http://www.omnis.net/developers/downloads/
2. Remove any unnecessary library from the /Startup folder of the Omnis App Server and place your own library and the database (cp.lbs + cucina_piccola.db) into the /Startup folder. This ensures that your library is started when the Omnis App server is launched.

   Note: In order to run the Omnis App Server in multi threaded mode you would need to run the command "start server" for example in the $construct of the Startup_Task of your library. Please check out the "Web and Mobile Apps" manual on the Developer Portal for further information.

3. Starting up the Omnis App Server you will be prompted for your Omnis App Server serial number that you can purchase from Omnis Software.
4. Configure your server configuration. This can be done through the "File" menu or through the config.json file that is located inside the /Studio folder of the Omnis App Server. Make sure you use a port number that not used on the machine. (e.g. 5912)
5. Make sure you exchange all resources that are used within the /HTML tree of the Omnis App Server installation. Remember that we copied the necessary pictures into a subfolder of the /html/pictures folder of the development version in the beginning of this course. We actually need the same resources within the Omnis App Server.

   Note: the /html folder is located inside the hidden "\AppData" (Windows) or "/library/ApplicationSupport" (Mac OS) folder.

6. Finally copy the two HTML files (jsService.htm and jsOutlet.htm) from the /html folder of the Omnis Studio developer version to the /html folder of the Omnis App Server.
7. Test your work!

You should now be able to connect your Omnis App Server remote forms using the following format. Let us assume the IP address of the server is 192.168.0.10 and the port number you have chosen is 5912:

http://192.168.0.10:5912/jschtml/jsService.htm

and

http://192.168.0.10:5912/jschtml/jsOutlet.htm

Remarks

I hope you had fun developing the Cucina Piccola app in this course.

If you struggled at any point please let me know – I really would like to help you. Drop me an email at: andreas.pfeiffer@omnis.net

Also if you continue and develop your own applications let me know. I am always curious what other developers create using Omnis Studio.

All the best,

Andreas