FrontBase®
An Omnis Studio Application

Because of last-minute changes to FrontBase, some
of the information in this manual may be inaccurate.
Please read the Release Notes on the FrontBase
distribution for the latest up-to-date information.

Revised: 02/04/07 cr

How to Contact FrontBase:

U.S.A. and International	FrontBase Inc. 26741 Portola Pkwy. Suite 1E #414 Foothill Ranch, CA 92610
	Frontline Software Aps Blokken 15 DK-3460 Birkerød Denmark
Ordering	Voice: +1 949 636 8026 Fax: +1 949 330 6371
	Voice: +45 4582 6262 Fax: +45 4582 0816
World Wide Web	http://www.frontbase.com
Technical Support	support@frontbase.com
Information	info@frontbase.com
Sales & Marketing	sales@frontbase.com
Licensing	license@frontbase.com
Document Feedback	doc-feedback@frontbase.com

Table of Contents

1 An Omnis Studio Application

To demonstrate how to use Omnis Studio as a front end to a FrontBase database we will create a simple call tracking application. This application will track information about calls between Customers and Employees. The example screenshots show Omnis Studio 4.2.0.3, FrontBase 4.2.6 and FrontBaseJManager 0.5082 running on an Intel Mac using OS X 10.4.8.

This chapter will guide you through the following:

It is assumed that you have downloaded and installed FrontBase, FrontBaseJManager and Omnis Studio. To download the latest FrontBase, FrontBaseJManager and Omnis versions please visit http://www.frontbase.com/ and http://www.omnis.net/download/studiodownload.html.

Note that the FrontBaseJManager application requires the latest Java Runtime Environment. This can be downloaded from http://www.sun.com/.

1.1 FrontBaseJManager

This is the Java management tool provided by FrontBase. Follow the steps in this section to create the FrontBase database for the demonstration application.

1. First launch the FrontBaseJManager application. The Monitored Databases panel should appear as in **Figure 1**.

2. To create a new database click on the **New** icon. This will open the New Database dialog window shown in **Figure 2**.

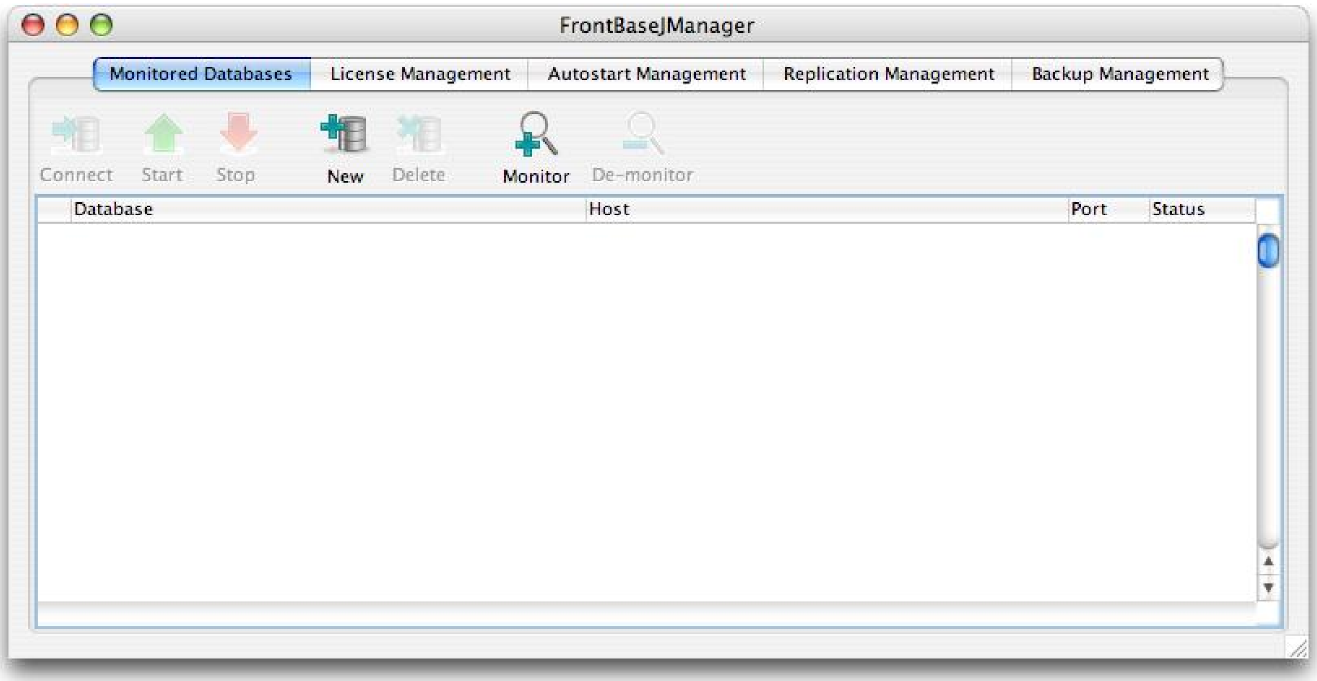
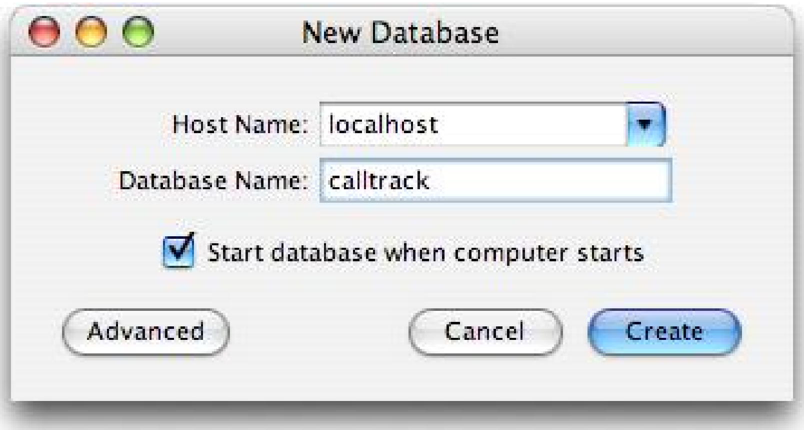


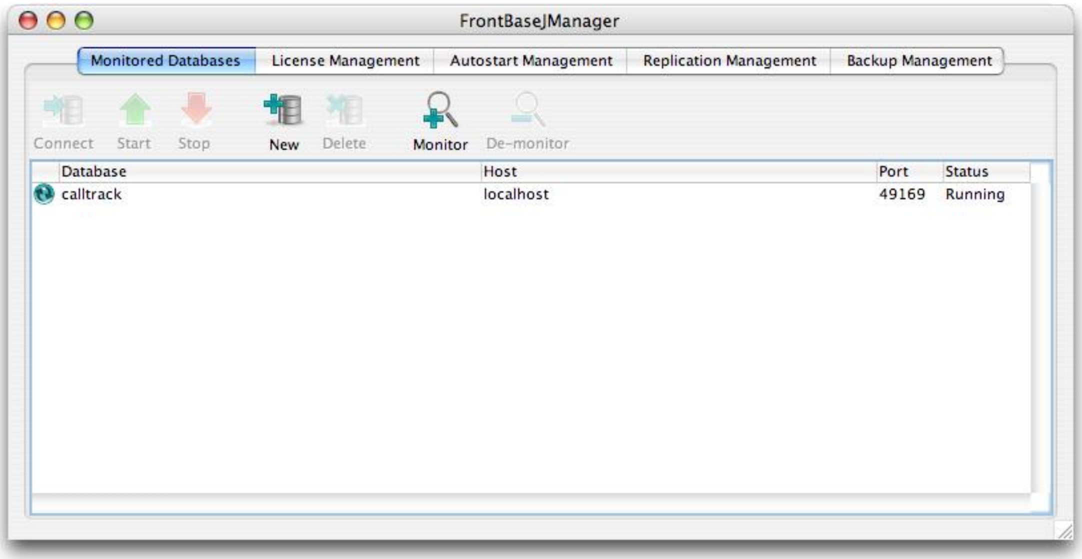
Figure 1. The Monitored Databases Panel.



Host Name: localhost
Database Name: calltrack
☑ Start database when computer starts

Figure 2. The New Database Dialog.

3. The default Host Name is already displayed, although you can change this if necessary. Enter **calltrack** for the Database Name and click the **Create** button. The calltrack database will be created and started in a second or two and displayed in the Monitored Databases window as in **Figure 3**.



Database	Host	Port	Status
calltrack	localhost	49169	Running

Figure 3. The Calltrack Database.

Now that we have a database defined in FrontBase, we can connect to it from Omnis Studio and create the rest of our application.

1.2 Connecting from Omnis Studio

We will now define a session template in Omnis Studio and use it to test the connection to the database which was created in the previous section. This assumes you have the correct FrontBase DAM installed in the Studio XCOMP folder.

1. Open Omnis Studio. Close the Welcome window if it is displayed and from the Studio Browser window select the SQL Browser from the **Folders** tree list as shown in **Figure 4**. Note that the examples use detailed view mode in the browser. To alter your view mode to detailed please select View->Details from the toolbar.

2. From the SQL Browser option list, on the left side of the browser window, select the **Session Manager** option. This will display all SQL session templates as in **Figure 5**.

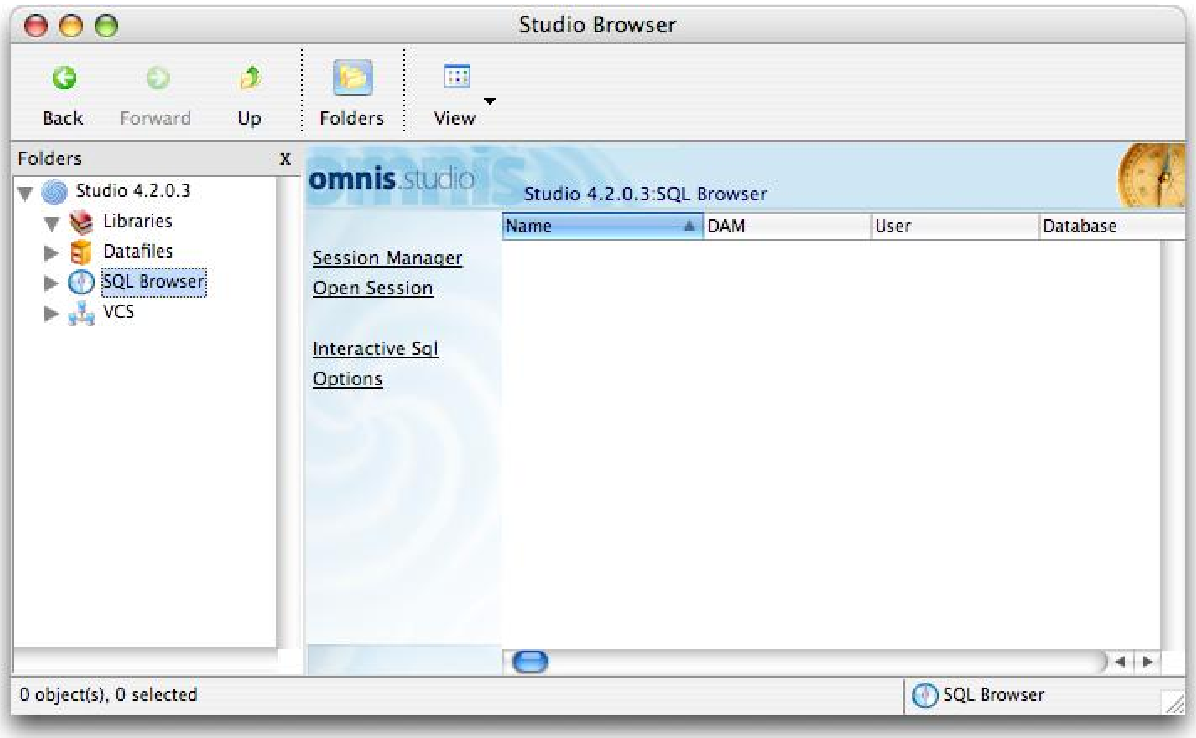
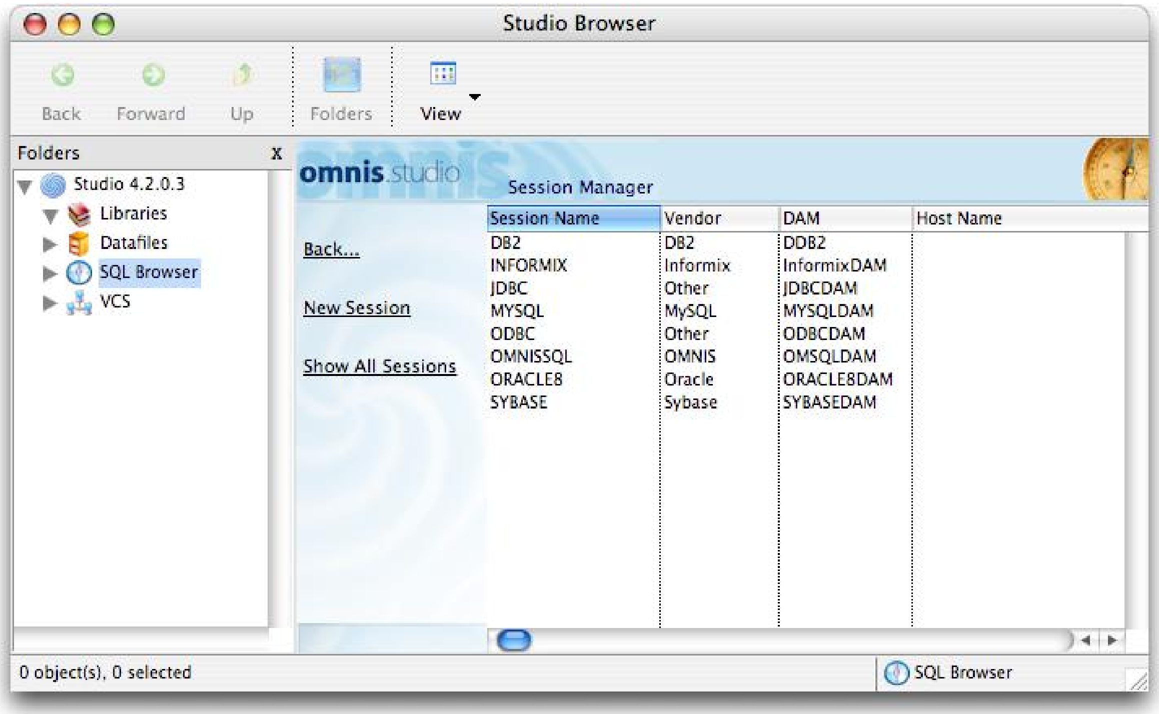


Figure 4. The SQL Browser Window.



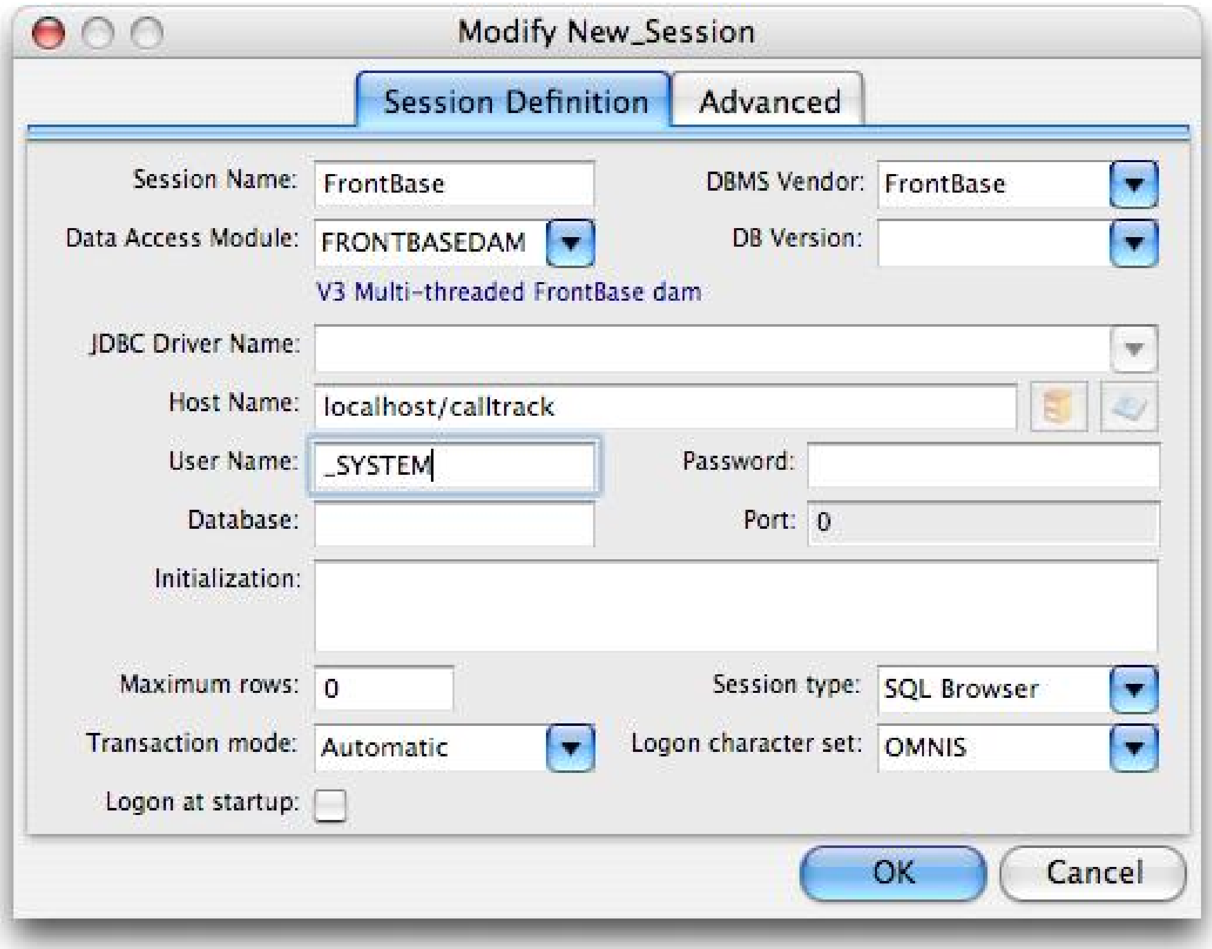
Session Name	Vendor	DAM	Host Name
DB2	DB2	DDB2	
INFORMIX	Informix	InformixDAM	
JDBC	Other	JDBCDAM	
MYSQL	MySQL	MYSQLDAM	
ODBC	Other	ODBCDAM	
OMNISSQL	OMNIS	OMSQLDAM	
ORACLE8	Oracle	ORACLE8DAM	
SYBASE	Sybase	SYBASEDAM	

Figure 5. The Session Manager.

3. Select **New Session** to open the New Session dialog and update the following fields on this window.

```
Session Name:FrontBase
DBMS Vendor:FrontBase
Data Access Module:FRONTBASEDAM
Host Name:<IP address of server>/calltrack
User Name:_SYSTEM
```

The Host Name is a slash-delimited string made up of the IP Address of the FrontBase server, the name of the FrontBase database for this connection, and an optional password for the database. Since our database does not currently have a password, this has been omitted here. The User Name can be any valid user name but we are making _SYSTEM the owner of the tables of our database. The finished window should look like the one in **Figure 6**.

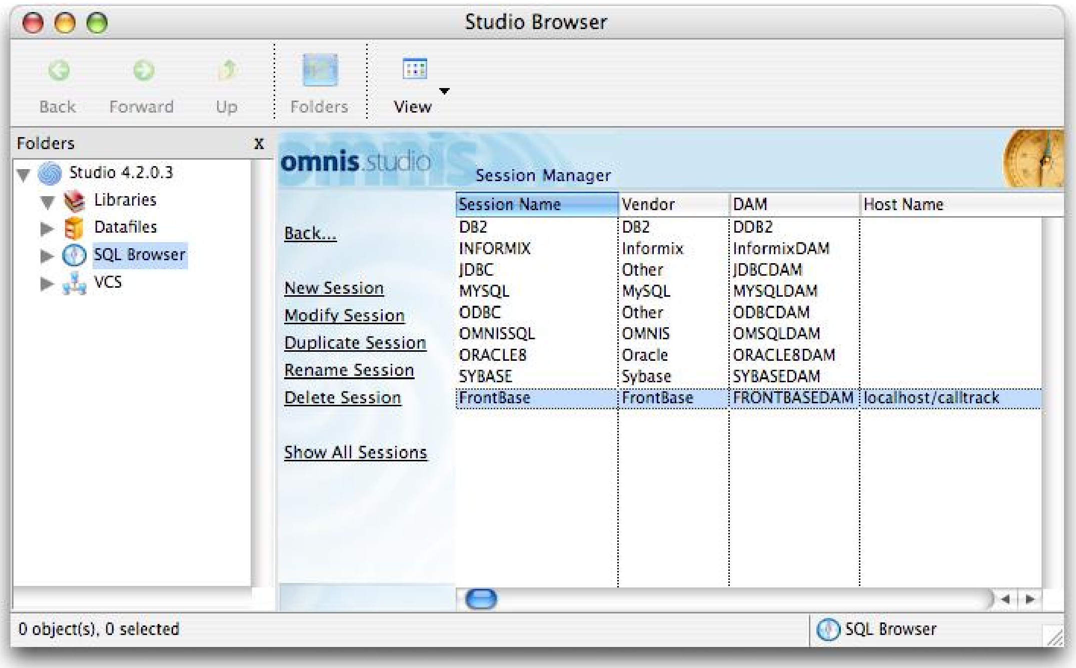


Session Name: FrontBase
DBMS Vendor: FrontBase
Data Access Module: FRONTBASEDAM
V3 Multi-threaded FrontBase dam
DB Version:
JDBC Driver Name:
Host Name: localhost/calltrack
User Name: _SYSTEM
Password:
Database:
Port: 0
Initialization:
Maximum rows: 0
Session type: SQL Browser
Transaction mode: Automatic
Logon character set: OMNIS
Logon at startup: ☐

Figure 6. A FrontBase Session Template.

4. To complete the definition of our FrontBase session template, click the **OK** button. This closes the session editor window. The Session Manager window should now look like the one in **Figure 7**.

5. Select **<u>Back…</u>** to return to the initial SQL Browser window as previously shown in **Figure 4**. Choose **<u>Open Session</u>** and the new **FrontBase** connection will be listed as in **Figure 8**.



Session Name	Vendor	DAM	Host Name
DB2	DB2	DDB2	
INFORMIX	Informix	InformixDAM	
JDBC	Other	JDBCDAM	
MYSQL	MySQL	MYSQLDAM	
ODBC	Other	ODBCDAM	
OMNISSQL	OMNIS	OMSQLDAM	
ORACLE8	Oracle	ORACLE8DAM	
SYBASE	Sybase	SYBASEDAM	
FrontBase	FrontBase	FRONTBASEDAM	localhost/calltrack

Figure 7. The New FrontBase Session Template.

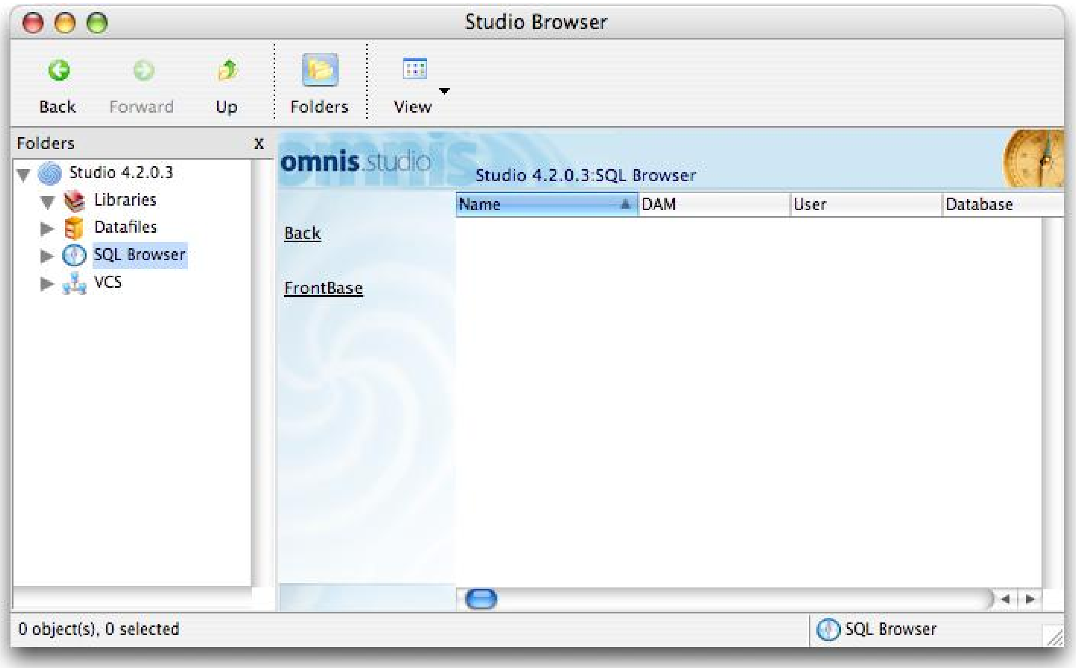


Figure 8. A Session to Connect to FrontBase.

6. Click <u>**FrontBase**</u> and if the connection is successful a **FRONTBASE** session will be shown under the SQL Browser branch in the Folders tree list as shown in **Figure 9**. The main browser window will list the type of available session objects. For FrontBase these are **Tables** and **Views**.

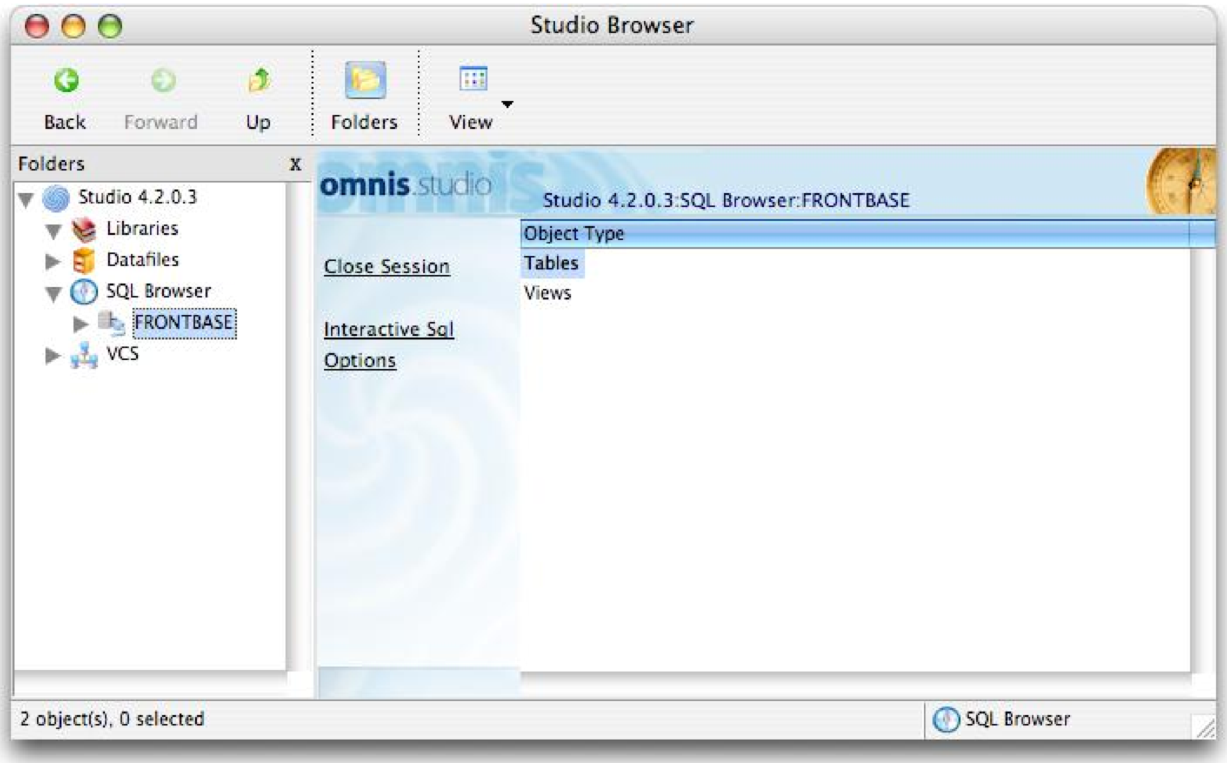


Figure 9. A Connected FrontBase Session.

If there is a problem connecting to the database then an error dialog will display the reason. This is commonly due to the DAM not being installed, the database not running or the connection information not being entered correctly.

7. Double-click on **Tables** in the SQL Browser window. This will list all available tables for our FrontBase session. Since we have not defined any tables in our database, this window is empty as shown in **Figure 10**.

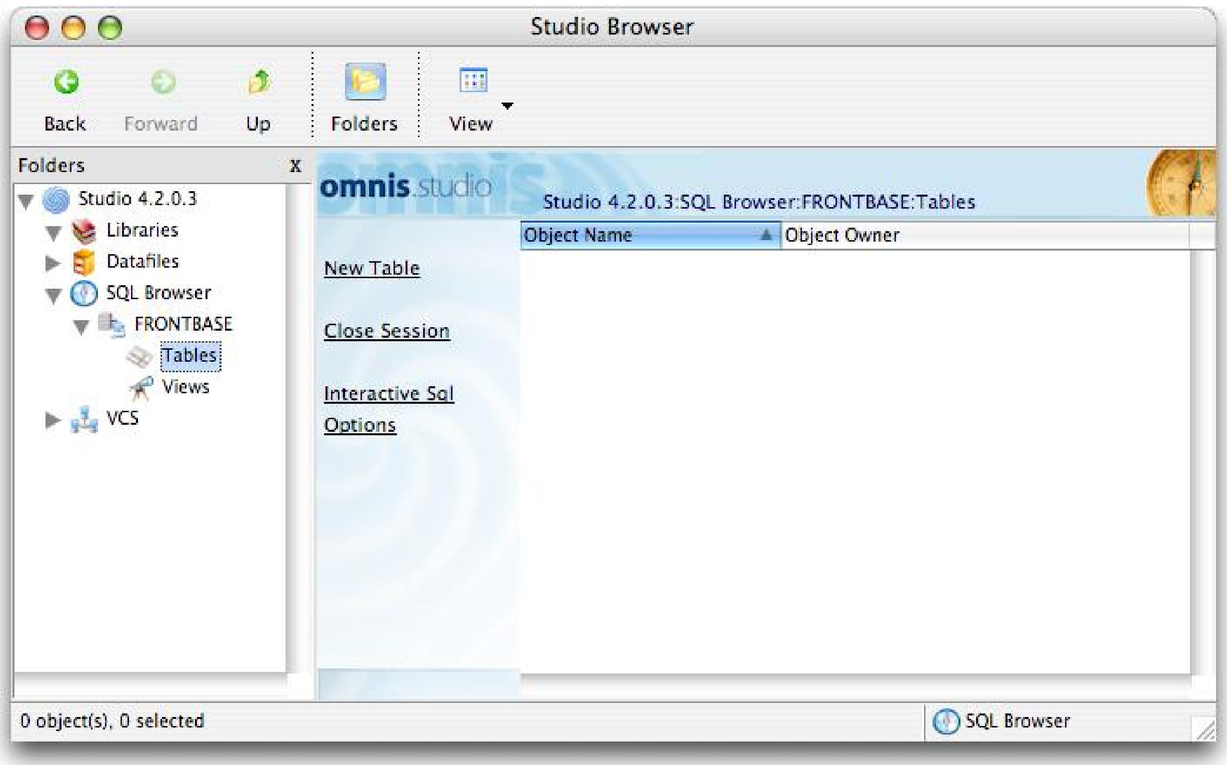


Figure 10. The Table List.

We will now use the tools provided in Omnis Studio to create the necessary tables for our project, but first we must create an Omnis Studio library to house the components of our application.

1.3 Library Creation

The following section details the steps to create our example library.

1. Select the **Libraries** branch from the Folders tree list and the browser will list all the available libraries as in **Figure 11**. This will initially be empty since there are no open libraries.

2. Select **New Library** and this will open a standard file dialog for naming and saving our new library as shown in **Figure 12**.

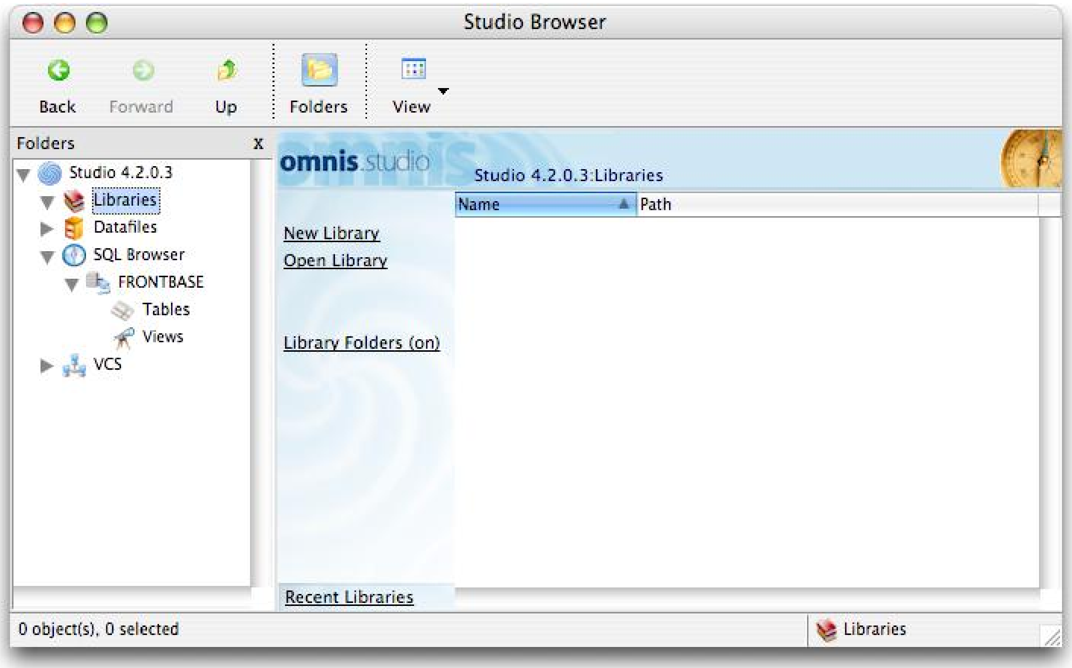


Figure 11. The Libraries Folder.

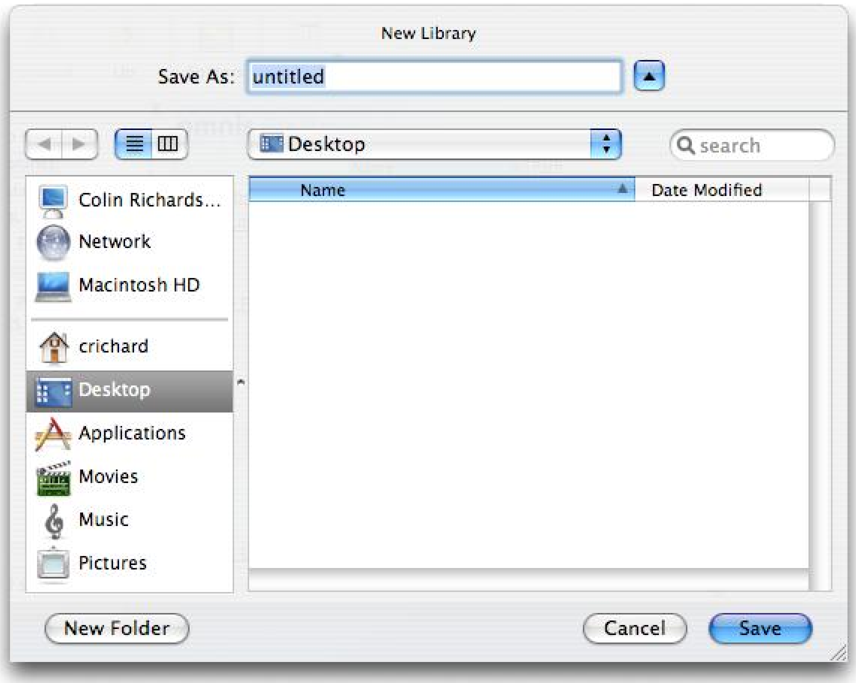
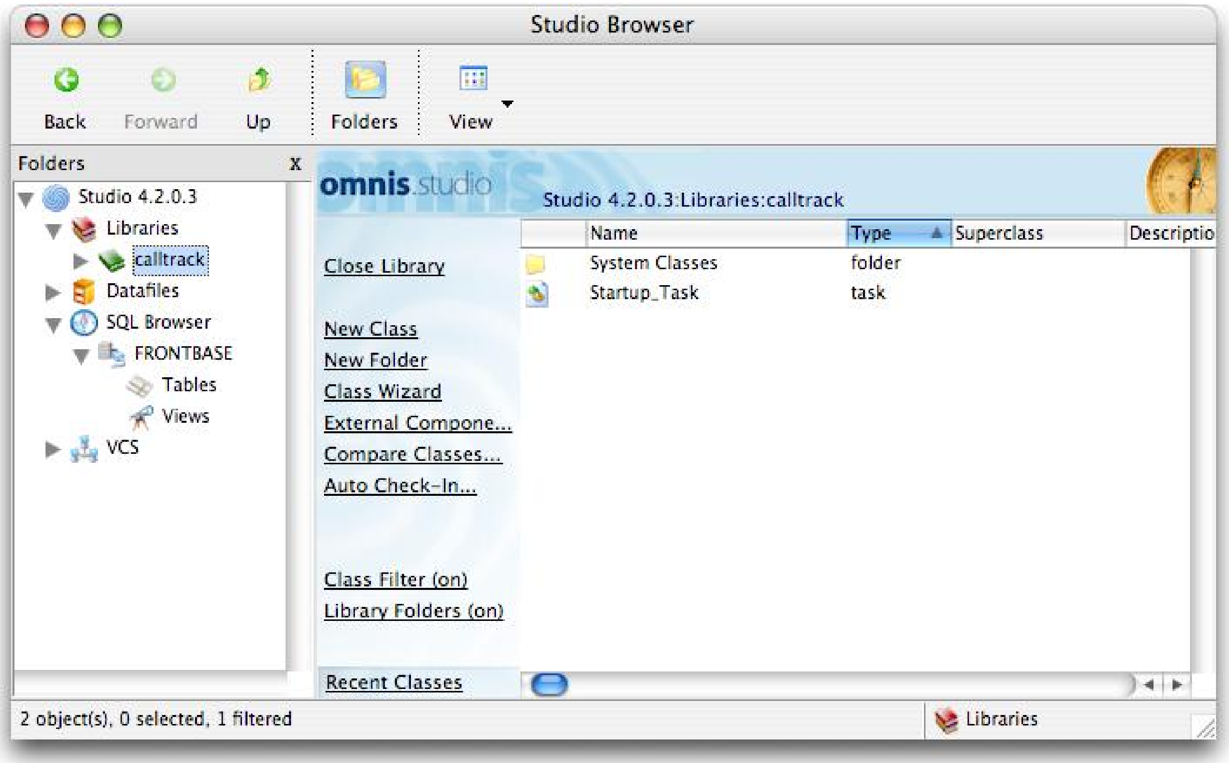


Figure 12. The New Library Dialog.

3. Name the new library **calltrack.lbs***, determine a location for the library file and click **Save**. The new calltrack library will be created and then listed under the **Libraries** branch in the Folders tree list. The main browser window will list the top-level components of the new library. This is shown in **Figure 13.**



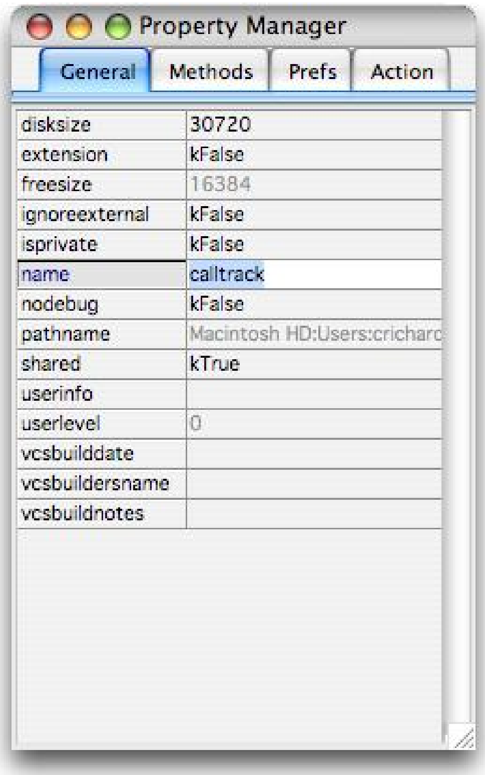
Name	Type	Superclass	Descriptio
System Classes	folder		
Startup_Task	task		

2 object(s), 0 selected, 1 filtered

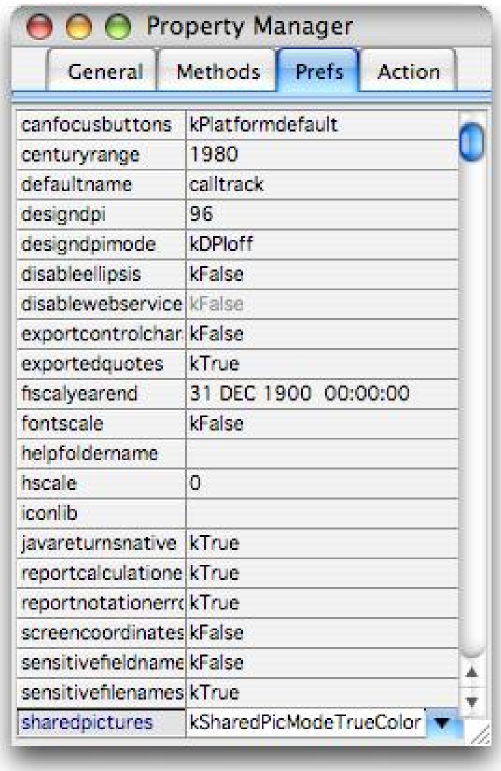
Figure 13. The New Calltrack Library.

4. It is good practice to set the internal "default" name for the library at this time. We do this by using the **Property Manager**, which we can open using the **View** menu on the main Omnis Studio menu bar or by control-clicking on the library name in the Folders tree list and selecting the **Properties** item from the context menu that appears. The Property Manager is shown in **Figure 14**.

* The ".lbs" extension is required for cross-platform compatibility with Windows systems. However, if we were using a Windows version of Omnis Studio for our example, then the extension would automatically be supplied.



disksize	30720
extension	kFalse
freesize	16384
ignoreexternal	kFalse
isprivate	kFalse
name	calltrack
nodebug	kFalse
pathname	Macintosh HD:Users:crichard
shared	kTrue
userinfo	
userlevel	0
vcsbuilddate	
vcsbuildersname	
vcsbuildnotes	



canfocusbuttons	kPlatformdefault
centuryrange	1980
defaultname	calltrack
designdpi	96
designdpimode	kDPIoff
disableellipsis	kFalse
disablewebservice	kFalse
exportcontrolchar	kFalse
exportedquotes	kTrue
fiscalyearend	31 DEC 1900 00:00:00
fontscale	kFalse
helpfoldername	
hscale	0
iconlib	
javareturnsnative	kTrue
reportcalculatione	kTrue
reportnotationerrc	kTrue
screencoordinates	kFalse
sensitivefieldname	kFalse
sensitivefilenames	kTrue
sharedpictures	kSharedPicModeTrueColor

Figure 14. Property Manager (General). **Figure 15. Property Manager (Prefs).**

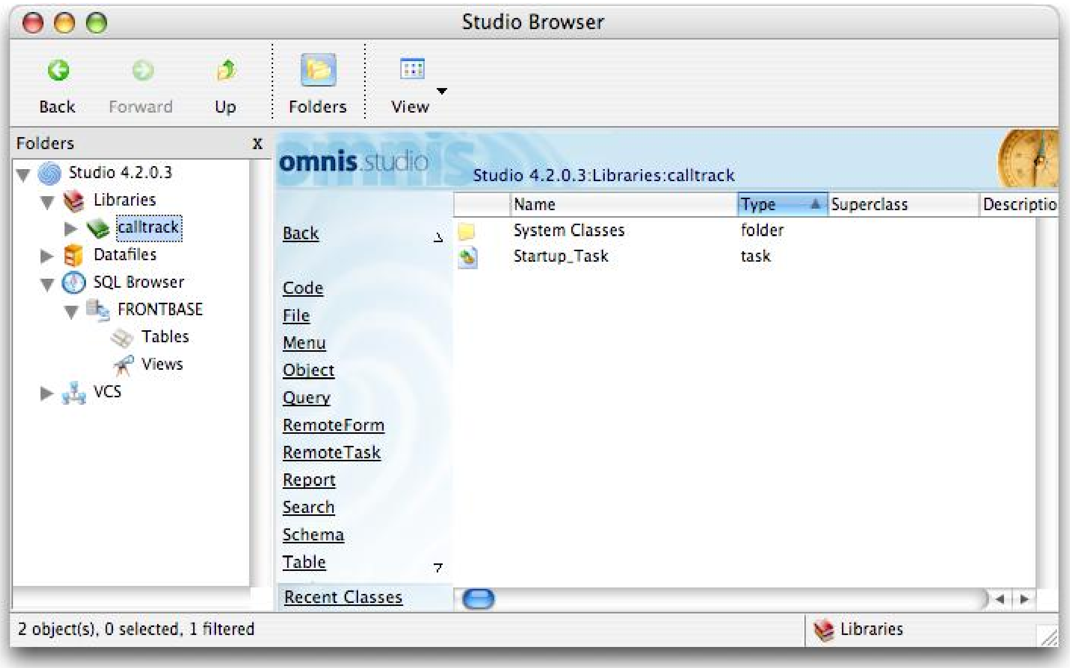
5. To set the default name copy the **name** property value from the **General** tab and paste it into the **defaultname** property under the **Prefs** tab. While in the Prefs tab, it is also a good idea to set the **sharedpictures** property to **kSharedPicModeTrueColor**. The resulting contents of the Prefs tab should look like **Figure 15**.

1.4 Table Creation

The definition used to map variables in Omnis Studio to the columns of a FrontBase database table is called a Schema class. To create tables in our database, we first create schema classes in Omnis Studio and then move them into FrontBase through our SQL Browser session.

1. Click **New Class** from the main Studio Browser option list. A listing of all standard class types will be displayed, as shown in **Figure 16**.

2. Click **Schema**. This places a schema class named **New Schema** into the browser window and highlights its name so we can immediately type one that is more appropriate, as shown in **Figure 17**.

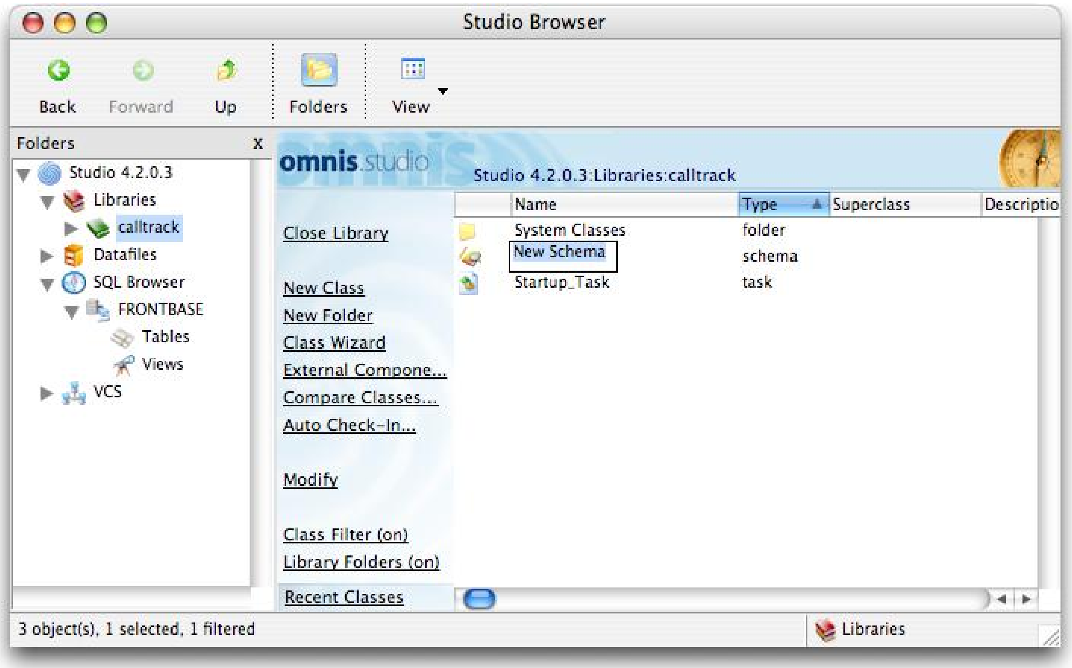


Studio 4.2.0.3:Libraries:calltrack

Name	Type	Superclass	Descriptio
System Classes	folder		
Startup_Task	task		

2 object(s), 0 selected, 1 filtered

Figure 16. The Class Types.



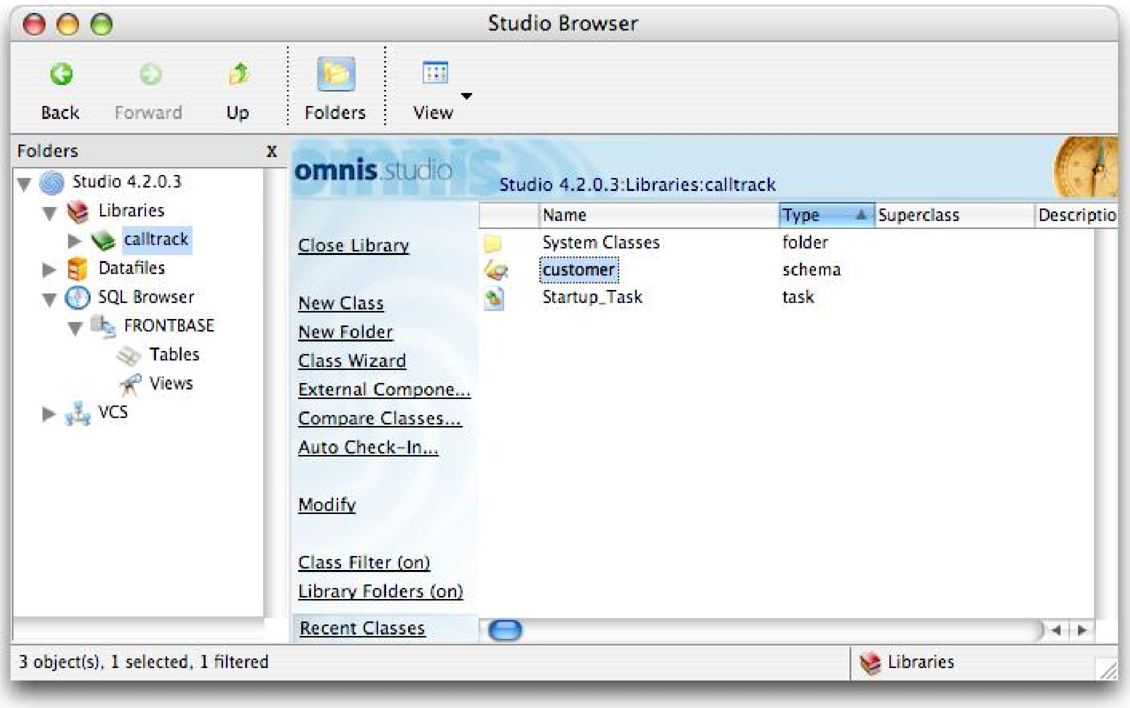
Studio 4.2.0.3:Libraries:calltrack

Name	Type	Superclass	Descriptio
System Classes	folder		
New Schema	schema		
Startup_Task	task		

3 object(s), 1 selected, 1 filtered

Figure 17. Editing the New Schema Name.

3. For our first schema class we will use the name **customer**. Typing that name and pressing the **Enter/Return** key will give us the result shown in **Figure 18**.



Studio 4.2.0.3:Libraries:calltrack

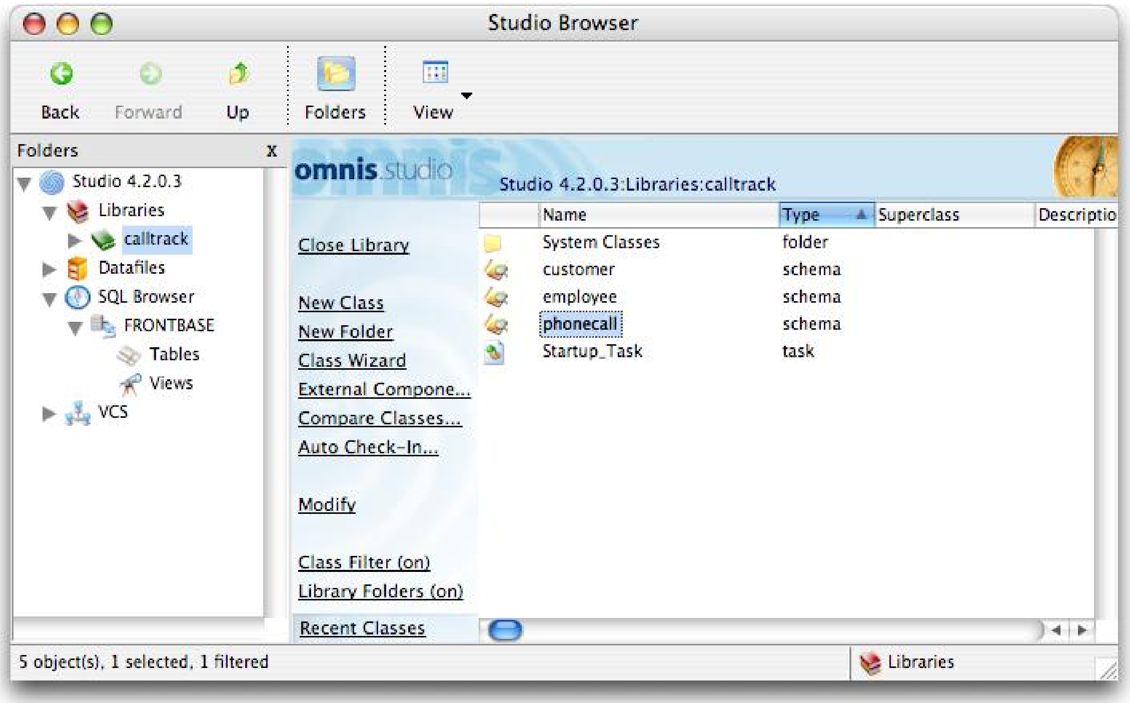
Name	Type	Superclass	Descriptio
System Classes	folder		
customer	schema		
Startup_Task	task		

3 object(s), 1 selected, 1 filtered

Figure 18. The Customer Schema.

4. Repeat the previous steps to add two more schema classes, one called **employee** and the other called **phonecall**. The three schemas should be listed as in **Figure 19**.

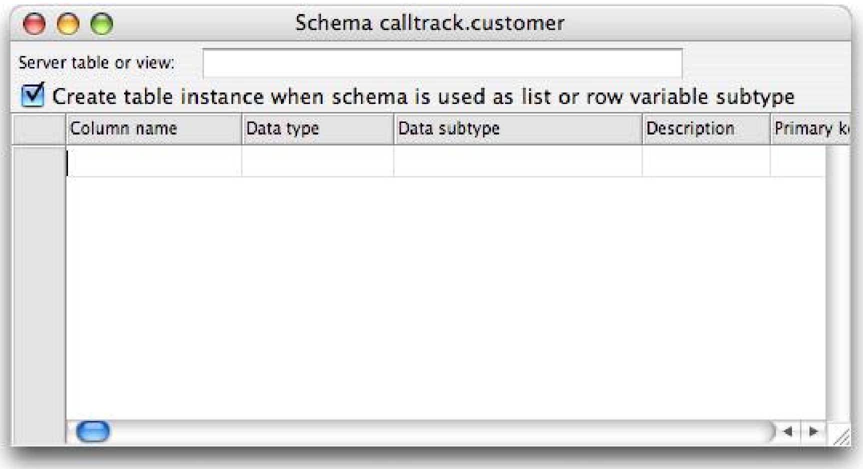
5. We now have three schema classes, but they are all empty. That is, they have no columns defined yet. To define columns for a schema class, we simply open it and fill in the necessary information. To open a schema class, just double-click on it in the browser window. If we do that for the customer schema class, we will see the Schema Class Editor for that class as shown in **Figure 20**.



Name	Type	Superclass	Descriptio
System Classes	folder		
customer	schema		
employee	schema		
phonecall	schema		
Startup_Task	task		

5 object(s), 1 selected, 1 filtered

Figure 19. The Schema Classes.



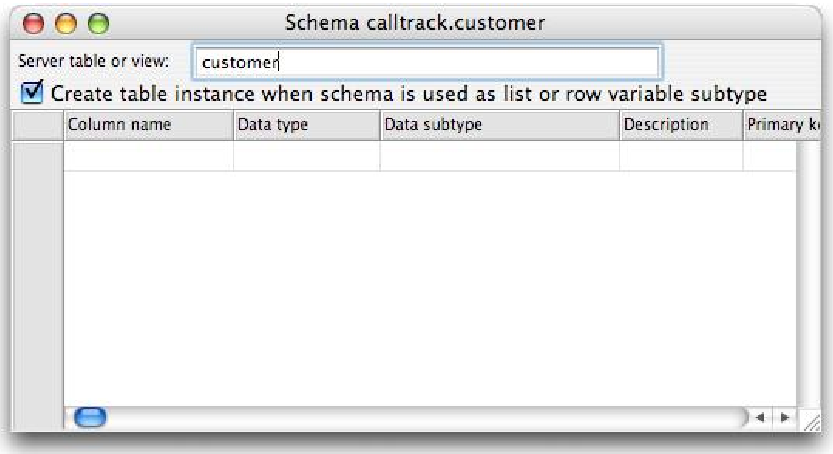
Server table or view:

☑ Create table instance when schema is used as list or row variable subtype

Column name	Data type	Data subtype	Description	Primary k

Figure 20. The Schema Class Editor.

6. We should first name the table we want this schema class to map to in our FrontBase database. The name that we provide here will be the name given to the table when it is created in FrontBase. For consistency, let's use **customer**. The editor window will now look like the one in **Figure 21**.


Schema calltrack.customer

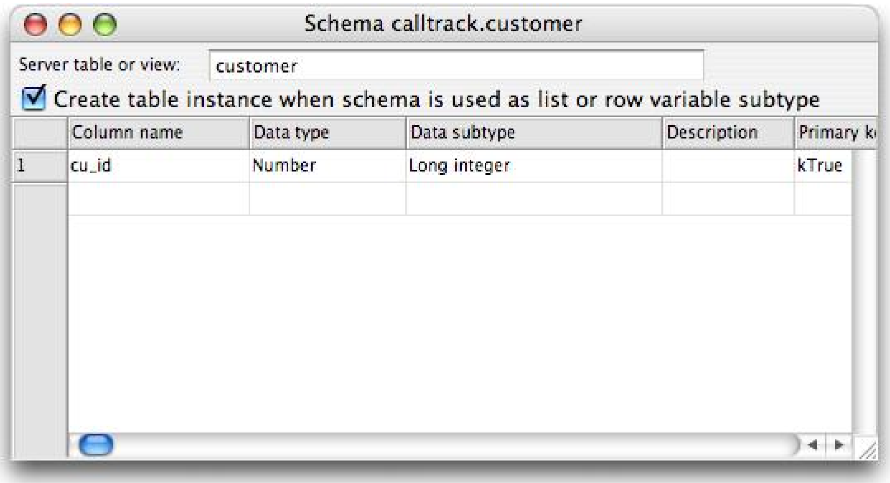
Server table or view: customer

☑ Create table instance when schema is used as list or row variable subtype

	Column name	Data type	Data subtype	Description	Primary k

Figure 21. Editing the Server Table Name.

7. Next we must define the columns for this table. To avoid unintentional conflicts with reserved words and to easily identify each column, it is a good idea to prefix each column name with all or part of the table name. Following this naming convention can save us many headaches as we work. For our first column, the customer id, enter **cu_id** under **Column name**. We will assign it a **Data type** of **Number** and a **Data subtype** of **Long integer**. The data type and subtype are selected from dropdown lists provided by Omnis Studio. We will also make this column the primary key of the schema by setting the **Primary key** column to **kTrue**. This means the value in this column will uniquely identify each entry in the database. When correctly done, the editor window will look like the one in **Figure 22**.


Schema calltrack.customer

Server table or view: customer

☑ Create table instance when schema is used as list or row variable subtype

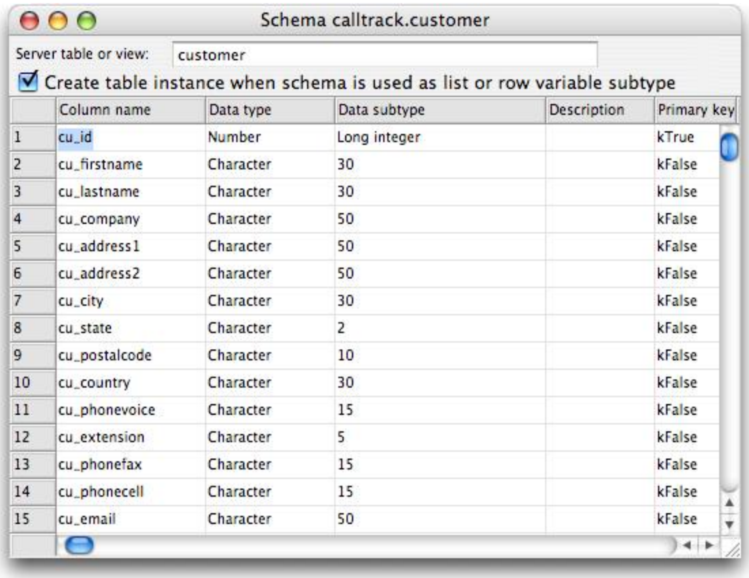
	Column name	Data type	Data subtype	Description	Primary k
1	cu_id	Number	Long integer		kTrue

Figure 22. The cu_id Column Definition.

8. Continue defining the columns for this schema class as shown in **Table 1**. Note, that when entering the subtype (length) for a Character variable, it must be manually typed rather than selected from a list. The finished schema class definition should look like the one in **Figure 23**.

Column name	Data type	Data subtype	Primary key
cu_id	Number	Long integer	kTrue
cu_firstname	Character	30	kFalse
cu_lastname	Character	30	kFalse
cu_company	Character	50	kFalse
cu_address1	Character	50	kFalse
cu_address2	Character	50	kFalse
cu_city	Character	30	kFalse
cu_state	Character	2	kFalse
cu_postalcode	Character	10	kFalse
cu_country	Character	30	kFalse
cu_phonevoice	Character	15	kFalse
cu_extension	Character	5	kFalse
cu_phonefax	Character	15	kFalse
cu_phonecell	Character	15	kFalse
cu_email	Character	50	kFalse

Table 1. The Customer Schema Column Definitions.


Schema calltrack.customer

Server table or view: customer

☑ Create table instance when schema is used as list or row variable subtype

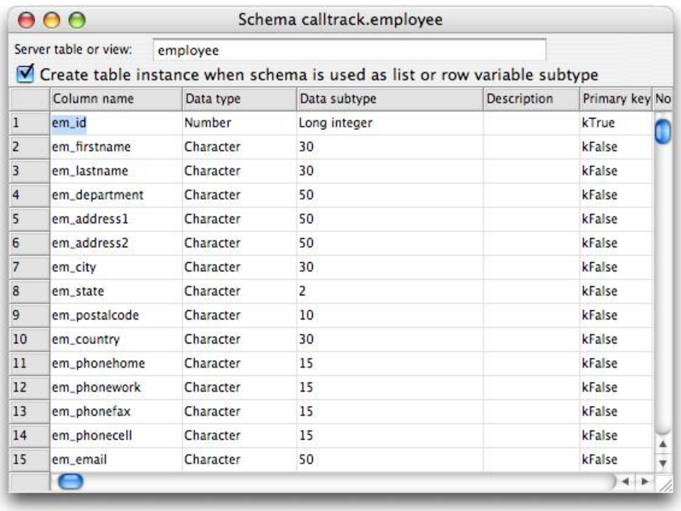
	Column name	Data type	Data subtype	Description	Primary key
1	cu_id	Number	Long integer		kTrue
2	cu_firstname	Character	30		kFalse
3	cu_lastname	Character	30		kFalse
4	cu_company	Character	50		kFalse
5	cu_address1	Character	50		kFalse
6	cu_address2	Character	50		kFalse
7	cu_city	Character	30		kFalse
8	cu_state	Character	2		kFalse
9	cu_postalcode	Character	10		kFalse
10	cu_country	Character	30		kFalse
11	cu_phonevoice	Character	15		kFalse
12	cu_extension	Character	5		kFalse
13	cu_phonefax	Character	15		kFalse
14	cu_phonecell	Character	15		kFalse
15	cu_email	Character	50		kFalse

Figure 23. The Completed Customer Schema.

9. Close this Schema Class Editor window and open one for the employee schema class. Enter **employee** in the field for **Server table or view** and create the column definitions for this schema class as shown in **Table 2**. The finished schema class definition should look like the one in **Figure 24**.

Column name	Data type	Data subtype	Primary key
em_id	Number	Long Integer	kTrue
em_firstname	Character	30	kFalse
em_lastname	Character	30	kFalse
em_department	Character	50	kFalse
em_address1	Character	50	kFalse
em_address2	Character	50	kFalse
em_city	Character	30	kFalse
em_state	Character	2	kFalse
em_postalcode	Character	10	kFalse
em_country	Character	30	kFalse
em_phonehome	Character	15	kFalse
em_phonework	Character	15	kFalse
em_phonefax	Character	15	kFalse
em_phonecell	Character	15	kFalse
em_email	Character	50	kFalse

Table 2. The Employee Schema Column Definitions.



Schema calltrack.employee

Server table or view: employee

☑ Create table instance when schema is used as list or row variable subtype

	Column name	Data type	Data subtype	Description	Primary key	No
1	em_id	Number	Long integer		kTrue	
2	em_firstname	Character	30		kFalse	
3	em_lastname	Character	30		kFalse	
4	em_department	Character	50		kFalse	
5	em_address1	Character	50		kFalse	
6	em_address2	Character	50		kFalse	
7	em_city	Character	30		kFalse	
8	em_state	Character	2		kFalse	
9	em_postalcode	Character	10		kFalse	
10	em_country	Character	30		kFalse	
11	em_phonehome	Character	15		kFalse	
12	em_phonework	Character	15		kFalse	
13	em_phonefax	Character	15		kFalse	
14	em_phonecell	Character	15		kFalse	
15	em_email	Character	50		kFalse	

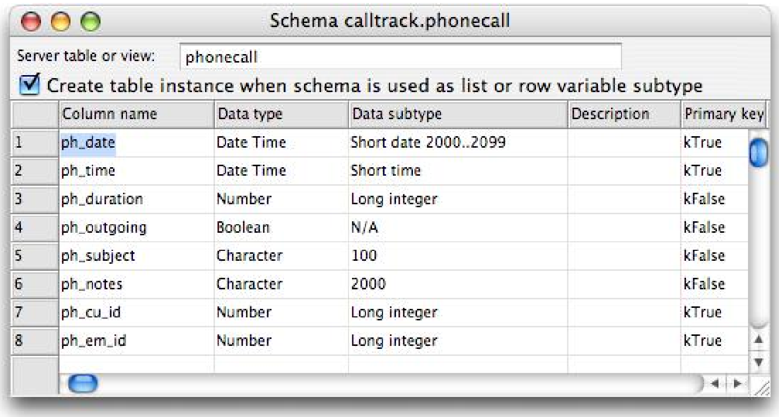
Figure 24. The Completed Employee Schema.

10. Close the employee schema and add the column definitions to the phonecall schema as shown in **Table 3**. Note, the primary key is made up of more than one column. In this table, a call between a customer and employee at a particular date and time should be unique for each entry.

Column name	**Data type**	**Data subtype**	**Primary key**
ph_date	Date time	Short date 2000..2099	kTrue
ph_time	Date time	Short time	kTrue
ph_duration	Number	Long integer	kFalse
ph_outgoing	Boolean	N/A	kFalse
ph_subject	Character	100	kFalse
ph_notes	Character	2000	kFalse
ph_cu_id	Number	Long integer	kTrue
ph_em_id	Number	Long integer	kTrue

Table 3. The Phonecall Schema Column Definitions.

11. Now enter **phonecall** in the field for **Server table or view**. The finished schema class definition should look like the one in **Figure 25.**


Schema calltrack.phonecall

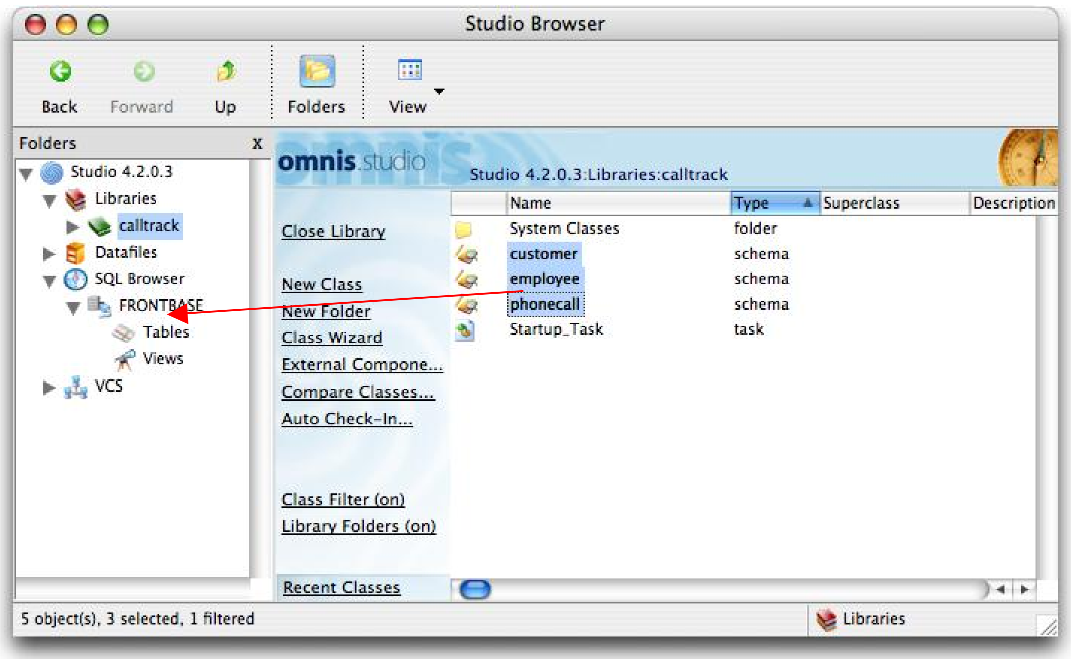
Server table or view: phonecall

☑ Create table instance when schema is used as list or row variable subtype

	Column name	Data type	Data subtype	Description	Primary key
1	ph_date	Date Time	Short date 2000..2099		kTrue
2	ph_time	Date Time	Short time		kTrue
3	ph_duration	Number	Long integer		kFalse
4	ph_outgoing	Boolean	N/A		kFalse
5	ph_subject	Character	100		kFalse
6	ph_notes	Character	2000		kFalse
7	ph_cu_id	Number	Long integer		kTrue
8	ph_em_id	Number	Long integer		kTrue

Figure 25. The Phonecall Schema.

12. Close the final Schema Class Editor window. Now we have three fully defined schema classes but we need to inform FrontBase of their contents. Omnis Studio makes this incredibly easy! Select the three schemas and drag them onto the **Tables** icon under the FRONTBASE session branch in the Folders tree list as shown in **Figure 26**. This will then automatically create the equivalent tables in FrontBase. If you select the Tables icon, the browser window will show the three tables in the FrontBase SQL Browser session as listed in **Figure 27**.

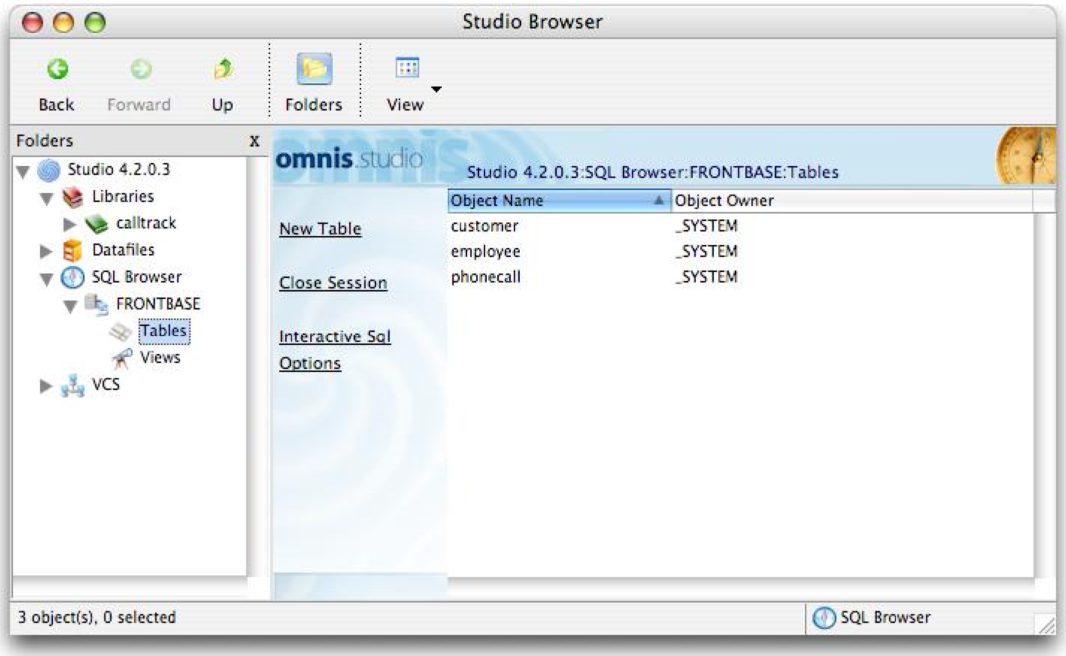


Studio 4.2.0.3:Libraries:calltrack

Name	Type	Superclass	Description
System Classes	folder		
customer	schema		
employee	schema		
phonecall	schema		
Startup_Task	task		

5 object(s), 3 selected, 1 filtered

Figure 26. Create the Tables in FrontBase.



Studio 4.2.0.3:SQL Browser:FRONTBASE:Tables

Object Name	Object Owner
customer	_SYSTEM
employee	_SYSTEM
phonecall	_SYSTEM

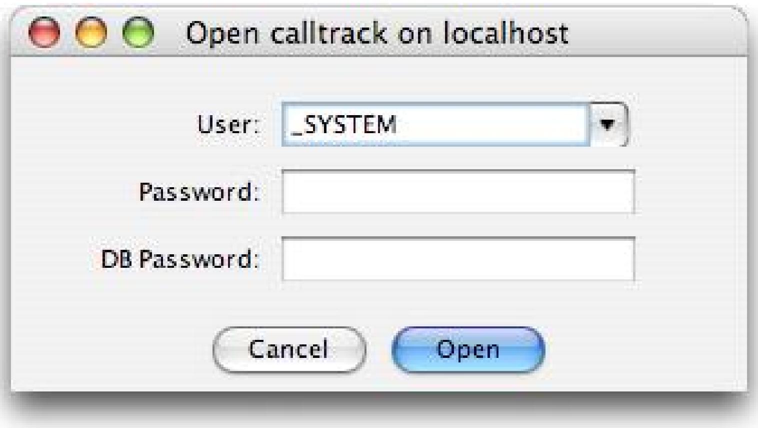
3 object(s), 0 selected

Figure 27. The FrontBase Tables.

1.5 Examining the Tables with FrontBaseJManager

Just to verify that we have successfully created the three tables in our FrontBase database, (named "calltrack"), we should examine the resulting table structures using FrontBaseJManager.

1. First we need to connect to our database in FrontBaseJManager to examine the tables we created with Omnis Studio. Double-click on the **calltrack** entry in the **Monitored Databases** window. This will open the connection window shown in **Figure 28**.



User: _SYSTEM

Figure 28. The Connection Dialog.

2. The default user should be set to **_SYSTEM**. Since no passwords have been set for either the database or any users, just click the **Open** button. The **Connection** window in **Figure 29** will be displayed. The default view shows the **SQL Interpreter** panel. This is used to execute SQL statements and log the statements that have been sent to the server. The **Connection Pane** list on the left side of the window gives access to the different panels which are available in order to perform tasks such as adding new users, viewing the database settings, etc.

3. To examine the tables that were created from Omnis we need to use the **Schema Objects** panel. Click on this in the Connection Pane to display the panel in **Figure 30**. This panel is split into three parts. The left column lists the available schemas, the middle column the type of object that can be contained in the selected schema and the right column will show the names of those objects for the selected object type.

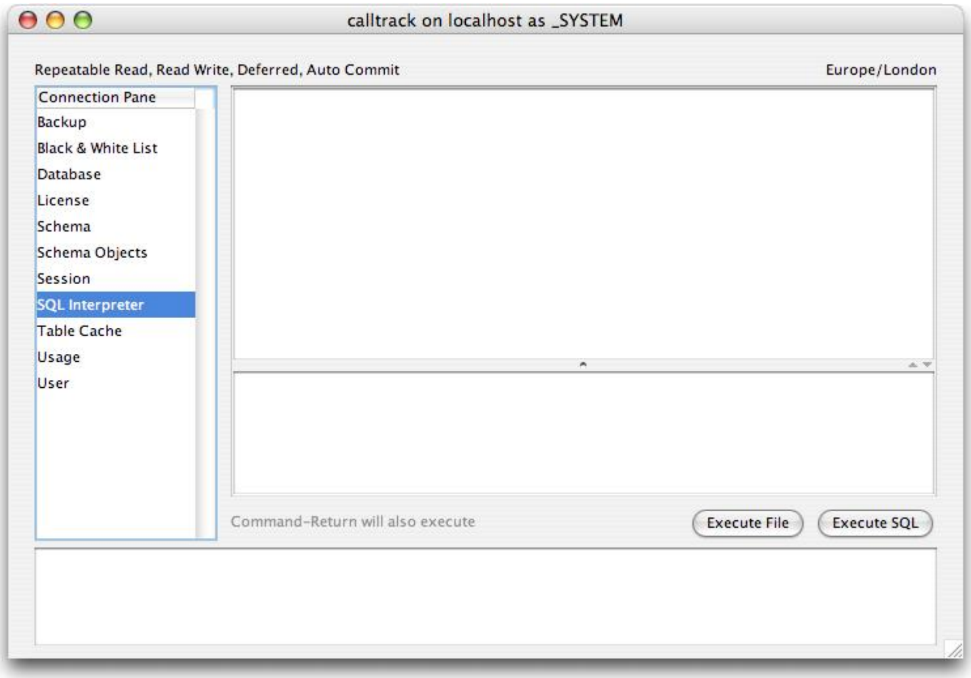
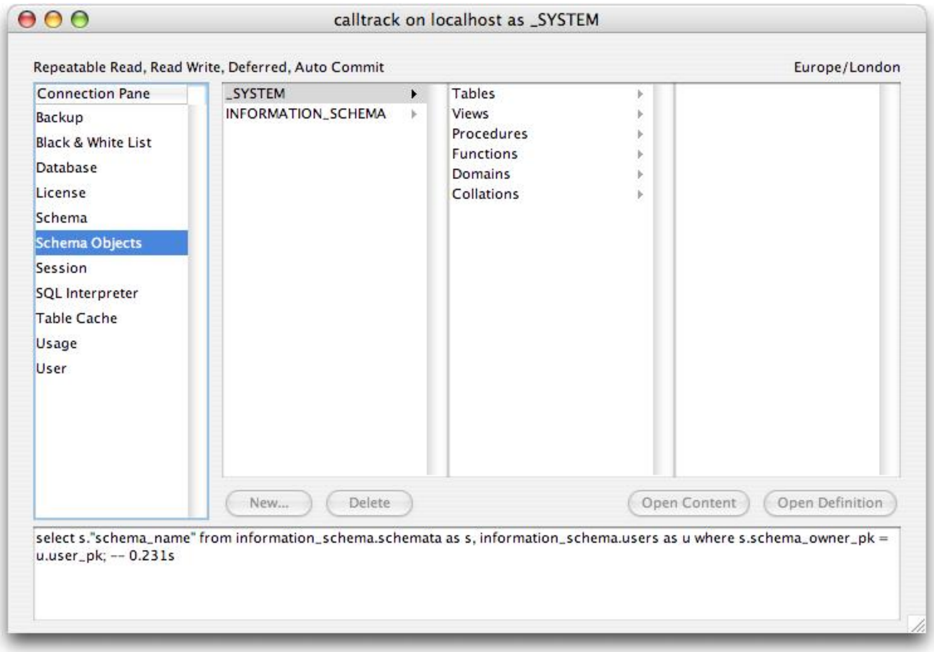


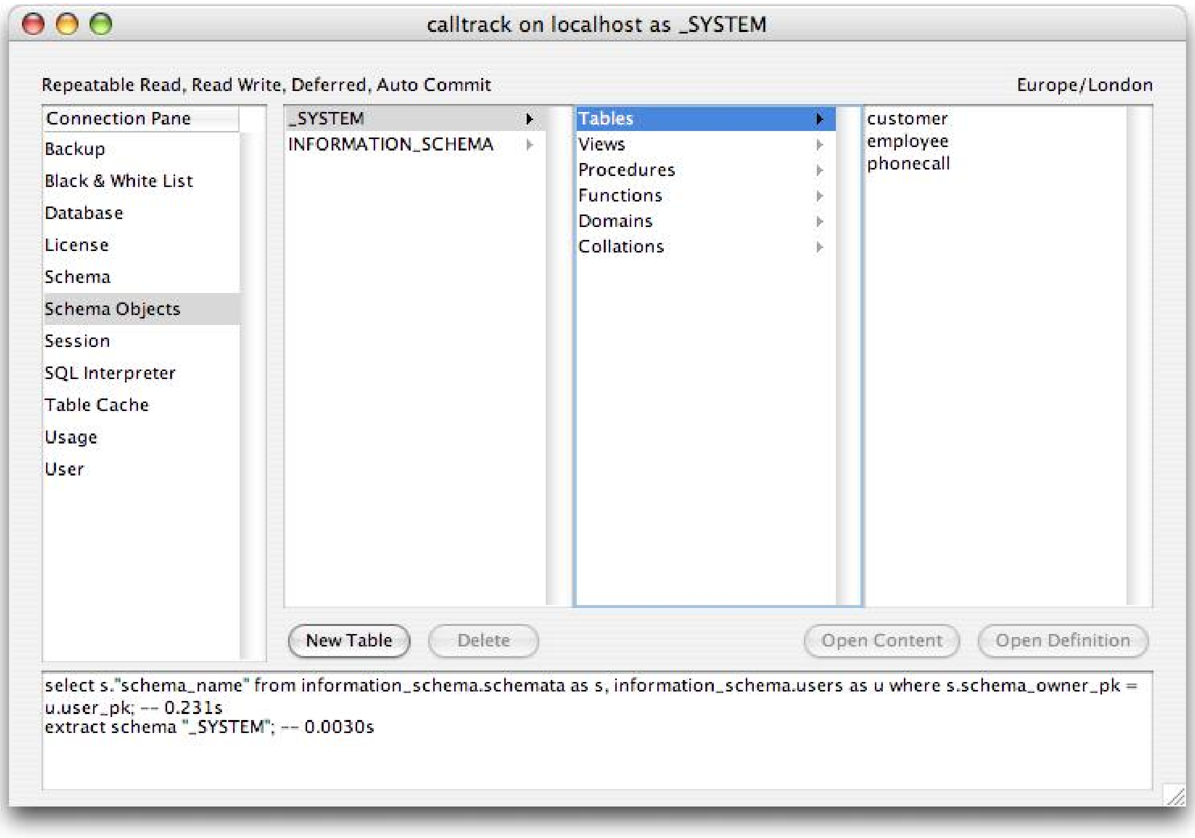
Figure 29. The SQL Interpreter Panel.



```
select s."schema_name" from information_schema.schemata as s, information_schema.users as u where s.schema_owner_pk =
u.user_pk; -- 0.231s
```

Figure 30. The Schema Objects Panel.

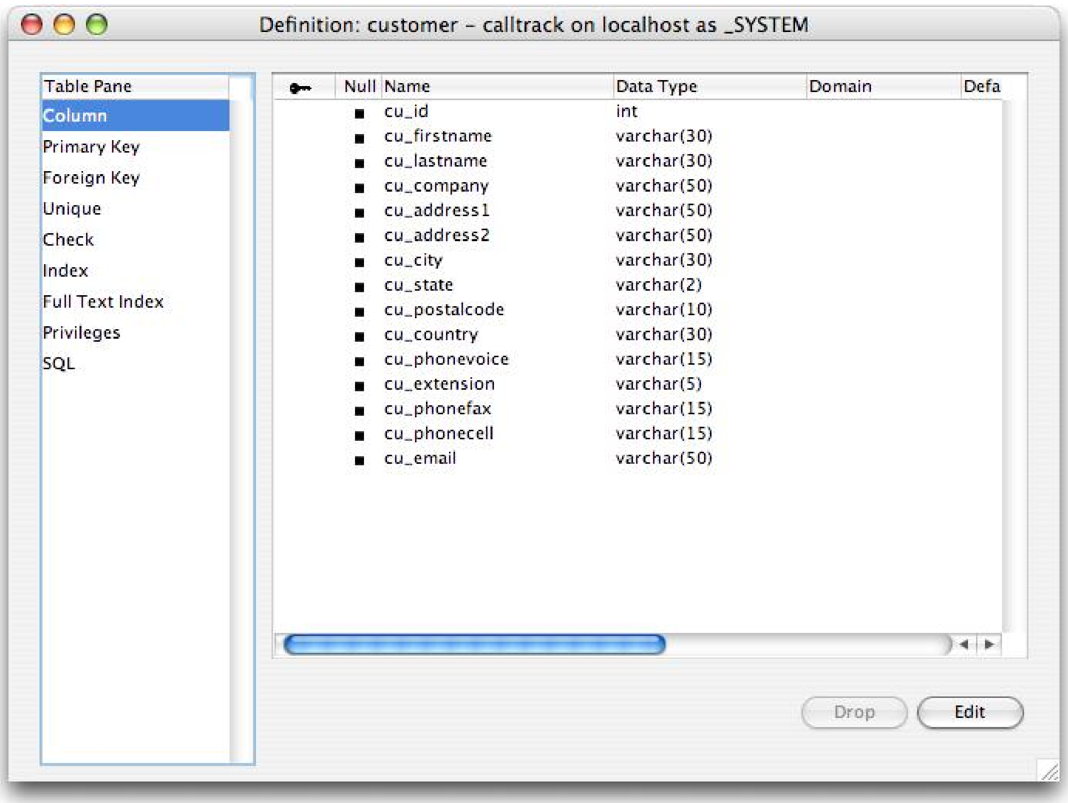
4. The tables that we created are part of the _SYSTEM schema so if we select **Tables** from the object type list (in the middle section), then the three tables are listed in the right hand column, as in **Figure 31**.



```
select s."schema_name" from information_schema.schemata as s, information_schema.users as u where s.schema_owner_pk =
u.user_pk; -- 0.231s
extract schema "_SYSTEM"; -- 0.0030s
```

Figure 31. The Three Tables in the Schema Objects Panel.

5. Double-clicking on the **customer** entry in the table list column opens a separate table **Definition** window for the customer table, as shown in **Figure 32**. On the left side of the window is the **Table Pane.** This lists the separate panels available that allow different operations on a table, e.g. viewing and creating keys. By default, the **Column** panel is shown. This lists all of the columns defined for a table, which in this case are the columns of the customer table that were created based on the Omnis schema definition.


Definition: customer – calltrack on localhost as _SYSTEM

	Null	Name	Data Type	Domain	Defa
	■	cu_id	int		
	■	cu_firstname	varchar(30)		
	■	cu_lastname	varchar(30)		
	■	cu_company	varchar(50)		
	■	cu_address1	varchar(50)		
	■	cu_address2	varchar(50)		
	■	cu_city	varchar(30)		
	■	cu_state	varchar(2)		
	■	cu_postalcode	varchar(10)		
	■	cu_country	varchar(30)		
	■	cu_phonevoice	varchar(15)		
	■	cu_extension	varchar(5)		
	■	cu_phonefax	varchar(15)		
	■	cu_phonecell	varchar(15)		
	■	cu_email	varchar(50)		

Figure 32. The Customer Table Column Definitions.

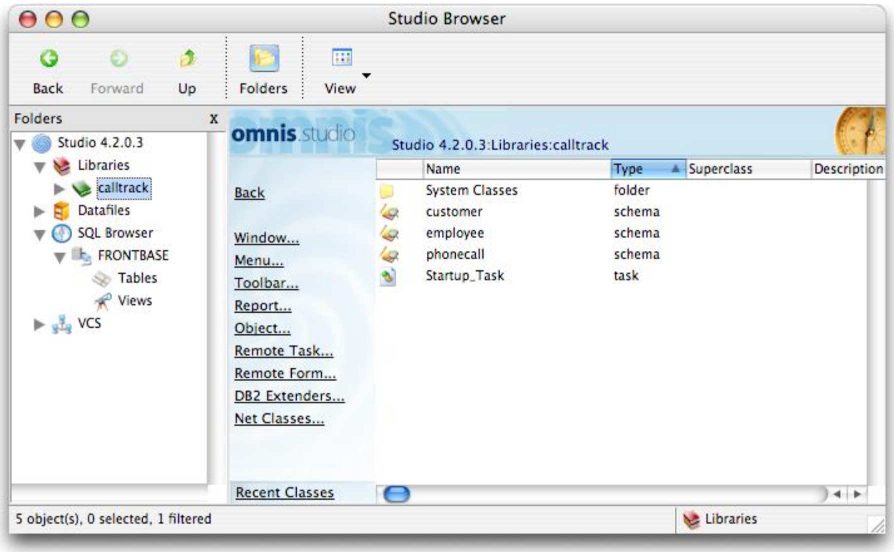
We can also open table definitions for the employee and phonecall tables by double-clicking on their entries in the connection window.

Now let's return to Omnis Studio to create some data entry windows for our application.

1.6 Creating Window Classes in Omnis Studio

Window classes are used for data browsing and data entry in an Omnis Studio application. We can create window classes from scratch as we did with our schema Classes, or we can employ wizards provided by Omnis to save time.

1. Select the calltrack library from the Folders tree list and this will display all the top-level components in our library. This will include the three schemas that were previously defined. Select the **Class Wizard** from the left hand option list of the main browser window and the classes that support wizards will be listed as in **Figure 33**.



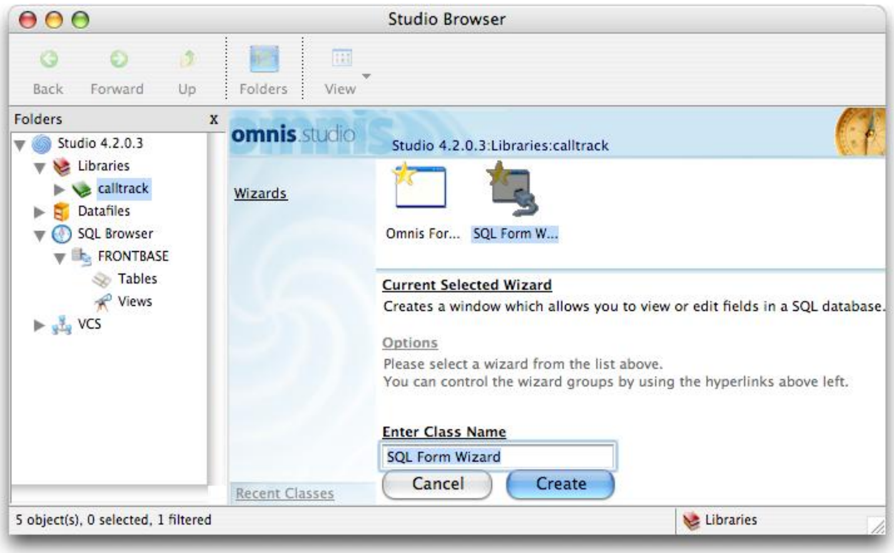
Studio 4.2.0.3:Libraries:calltrack

Name	Type	Superclass	Description
System Classes	folder		
customer	schema		
employee	schema		
phonecall	schema		
Startup_Task	task		

5 object(s), 0 selected, 1 filtered

Figure 33. The Wizard Classes.

2. To view the window class wizards click **Window...**. The browser will list the **Omnis Form Wizard** and the **SQL Form Wizard**. We will be using the SQL Form Wizard because this is used to create windows which operate against SQL databases. The other wizard is used for Omnis data files. The wizards are shown in **Figure 34,** with the SQL Form Wizard highlighted.



Studio 4.2.0.3:Libraries:calltrack

Current Selected Wizard
Creates a window which allows you to view or edit fields in a SQL database.

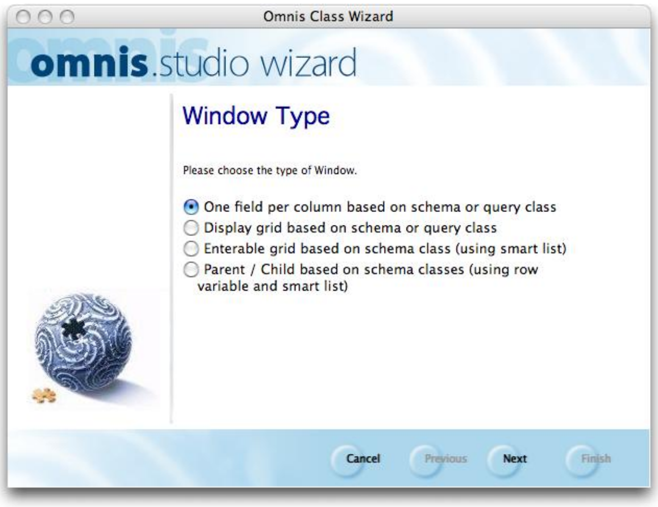
Options
Please select a wizard from the list above.
You can control the wizard groups by using the hyperlinks above left.

Enter Class Name
SQL Form Wizard

5 object(s), 0 selected, 1 filtered

Figure 34. The Window Class Wizards.

3. Enter the class name for the first window as **customerEntry** and hit **Enter/Return**. The first wizard panel, **Window Type,** will appear as shown in **Figure 35**.



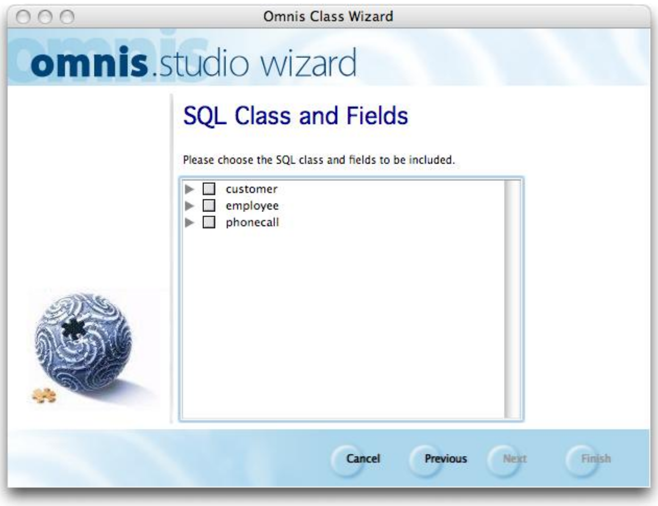
Window Type

Please choose the type of Window.

- (●) One field per column based on schema or query class
- ( ) Display grid based on schema or query class
- ( ) Enterable grid based on schema class (using smart list)
- ( ) Parent / Child based on schema classes (using row variable and smart list)

Figure 35. The Window Type Panel.

4. This window asks us to choose a window type from the options offered. The default option, "**One field per column based on schema or query class**" is what we want, so we make sure the corresponding radio button is selected and click the **Next** button at the bottom of the wizard window. We then move on to the **SQL Class and Fields** dialog shown in **Figure 36**.



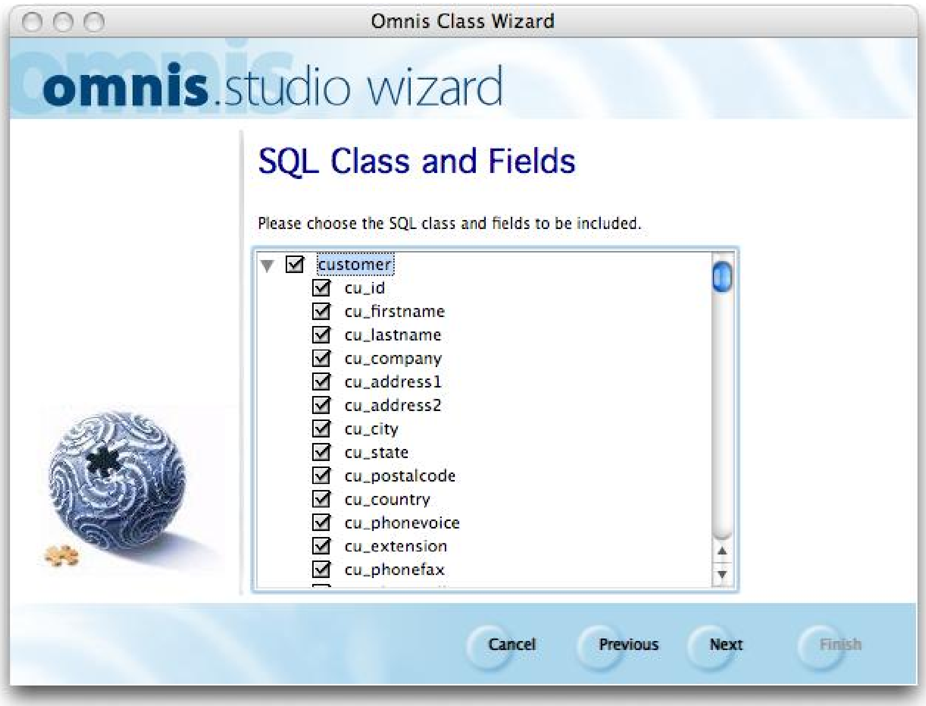
SQL Class and Fields

Please choose the SQL class and fields to be included.

- ☐ customer
- ☐ employee
- ☐ phonecall

Figure 36. The SQL Class and Fields Panel.

5. This window allows us to choose which schema class fields we want to access using the window we are building. The fields will correspond to the table columns in the database. We can select only one class here. The schemas are presented as a tree list with each schema expandable to list the fields it contains. The schema and fields can be selected/deselected by clicking on the checkbox next to each entry. To select/deselect all fields in the schema you can click the checkbox next to the schema name. We want all the columns from the customer schema class in this case, so check the **customer** checkbox. If you expand the customer branch you will end up with the window shown in **Figure 37**, showing that all the customer fields are now selected.



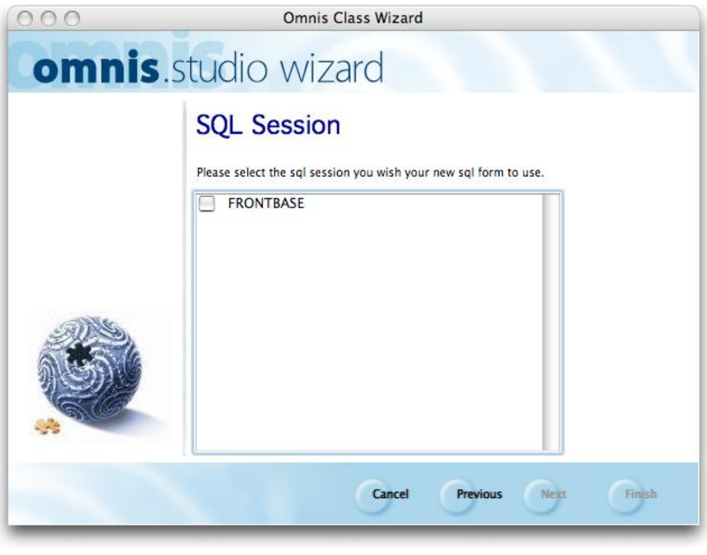
SQL Class and Fields

Please choose the SQL class and fields to be included.

- ✓ customer
  - ✓ cu_id
  - ✓ cu_firstname
  - ✓ cu_lastname
  - ✓ cu_company
  - ✓ cu_address1
  - ✓ cu_address2
  - ✓ cu_city
  - ✓ cu_state
  - ✓ cu_postalcode
  - ✓ cu_country
  - ✓ cu_phonevoice
  - ✓ cu_extension
  - ✓ cu_phonefax

Figure 37. The Selected Customer Fields.

6. Click **Next** and the **SQL Session** panel in **Figure 38** will be displayed. Here we are asked which currently open SQL session we wish to use for the window we are building. In our case we have only one, **FRONTBASE** so we select it and click the **Next** button. The **Window Themes** selection panel, shown in **Figure 39**, is then displayed.

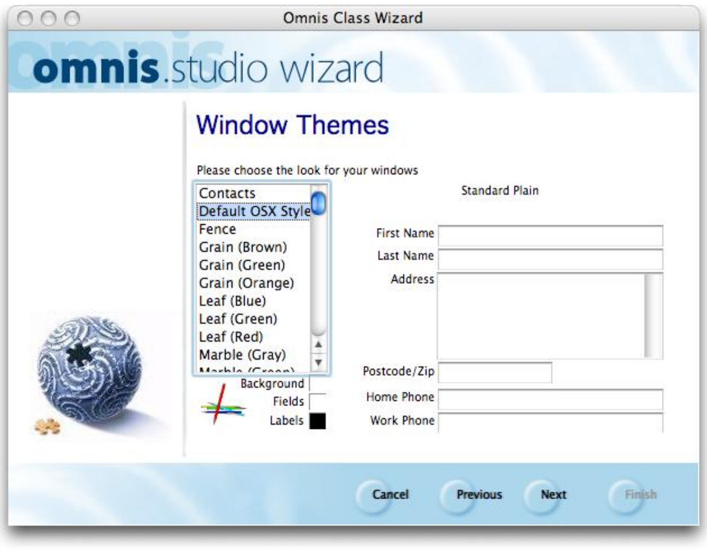


SQL Session

Please select the sql session you wish your new sql form to use.

FRONTBASE

Figure 38. The SQL Session Panel.



Window Themes

Please choose the look for your windows

Contacts
Default OSX Style
Fence
Grain (Brown)
Grain (Green)
Grain (Orange)
Leaf (Blue)
Leaf (Green)
Leaf (Red)
Marble (Gray)

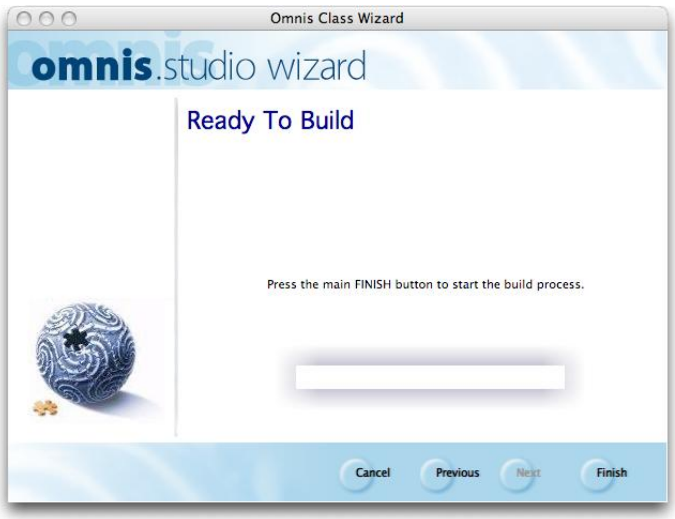
Background
Fields
Labels

Standard Plain

First Name
Last Name
Address
Postcode/Zip
Home Phone
Work Phone

Figure 39. The Window Themes Panel.

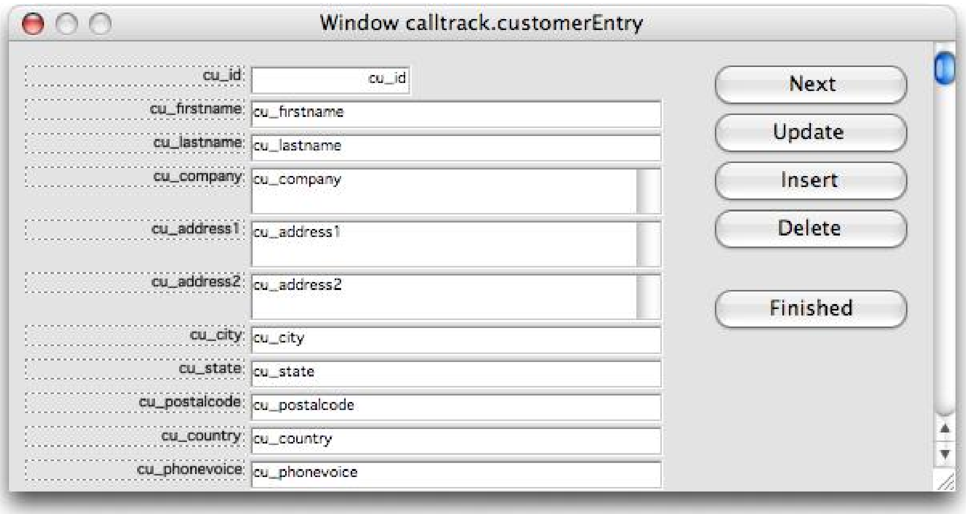
7. Here is where we can have some fun. Omnis Studio ships with many themes installed and each theme allows us to modify the window background color, field background color and label text color right here in the wizard. Of course, we have control over even more properties with the main editing tools of Omnis Studio but this gets us off to a nice start. Select **Standard (3D Inset)** from the list and click the **Next** button. The Wizard now displays the **Ready To Build** panel as shown in **Figure 40**.



Press the main FINISH button to start the build process.

Figure 40. The Ready To Build Panel.

8. As the wizard says, click the **Finish** button at the bottom of the window and Omnis Studio will build a window to the specifications we selected, complete with methods for accessing and updating our FrontBase database. After a few moments, a Window Class Editor for our finished window is shown as in **Figure 41**.



cu_id:	cu_id
cu_firstname:	cu_firstname
cu_lastname:	cu_lastname
cu_company:	cu_company
cu_address1:	cu_address1
cu_address2:	cu_address2
cu_city:	cu_city
cu_state:	cu_state
cu_postalcode:	cu_postalcode
cu_country:	cu_country
cu_phonevoice:	cu_phonevoice

Figure 41. The Customer Entry Window Class.

The window we built here may not be all that we want it to be, but it is a great start for the effort involved! There are a limitless number of enhancements we could make to its appearance but next we will test the window to ensure it works as we would like.

1.7 Viewing and Entering Data

Omnis Studio allows us to open a window as we are designing it to test it. We do this by either typing Command-T (on Macintosh), Control-T (on Windows and Linux), or by using the **Open Window** item from the window's context menu (accessed by Control/Right-click). This context menu is shown in **Figure 42**.

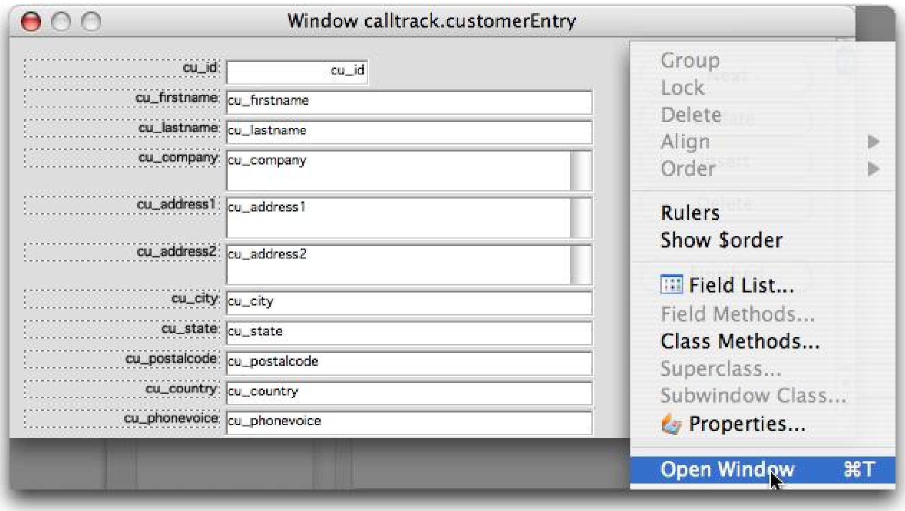


Figure 42. Opening the Window in Test Mode.

1. Perform one of these actions for this window class, and a test instance of the window is created, as shown in **Figure 43**.

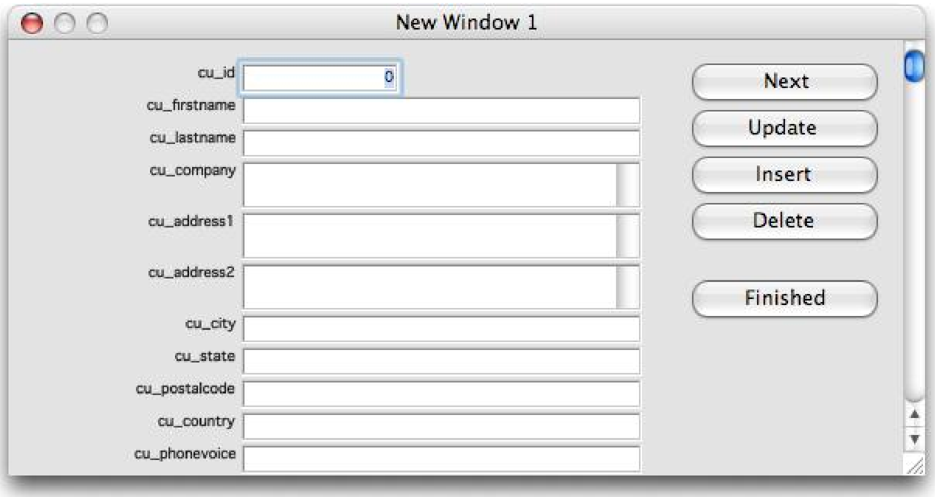
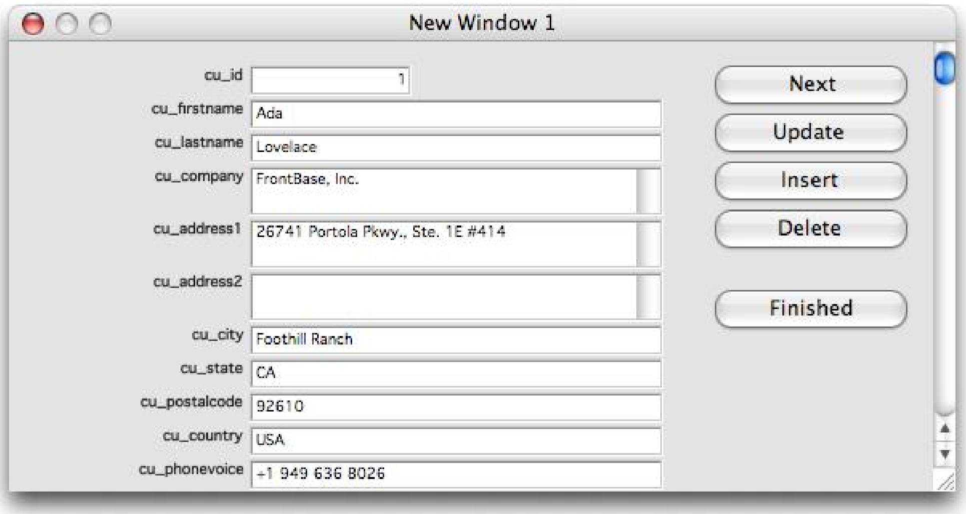


Figure 43. The Window Test Instance.

2. Since we have no data in our database yet, there is nothing to view. We can quickly change that, though. The fields on this window are editable, so we can type in some data. **Figure 44** shows the window with the fields populated.



cu_id	1
cu_firstname	Ada
cu_lastname	Lovelace
cu_company	FrontBase, Inc.
cu_address1	26741 Portola Pkwy., Ste. 1E #414
cu_address2	
cu_city	Foothill Ranch
cu_state	CA
cu_postalcode	92610
cu_country	USA
cu_phonevoice	+1 949 636 8026

Figure 44. The Populated Window Fields.

3. There is still no data in our database. We create a record in the database from the data entered on this window by clicking the **Insert** button. Each time this button is clicked, a new record is added to the customer table. Go ahead and add a few records but make sure that the value for **cu_id** is unique for each entry.

4. We can now browse through the records we've entered by repeatedly clicking on the **Next** button. The fact that the contents of the window change indicates that we are connected to FrontBase and affecting the calltrack database.

5. Records in the database can also be modified. Just make a change to an existing record and click the **Update** button. The change will be committed to the database.

6. When we have no more operations to perform using this window, we either click the **Finished** button or the close box in the upper left corner of the window. Either action closes the window instance and returns us to the Window Class Editor. The Window Class Editor was always open just behind the window instance spawned from it. Clicking on the Window Class Editor to bring it into focus also closes the test instance.

1.8 Modifying the Window Class

As stated above, there are many things we could do to improve the appearance of this window. We will examine two here; adjusting the content of the field labels and resizing and arranging the fields themselves. We will also give the window instance a more appropriate title.

1. We can access the contents of a field label just by double-clicking on it. When we do this, an edit window for the label content appears as in **Figure 45**.

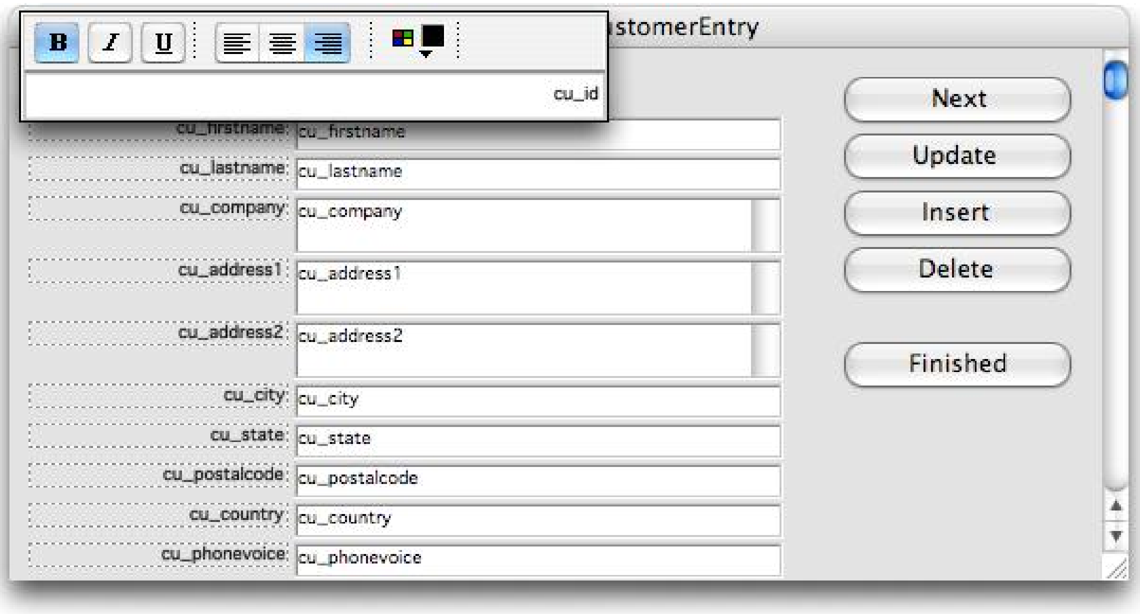


Figure 45. The Label Edit Window.

2. We can simply type in new content. For example, here we can change **c_id** to **ID#**. We can close the edit window by clicking on the main Window Class Editor or by using Command-W on Mac and Control-W on Windows/Linux. With updated labels the window could look like the one in **Figure 46**.

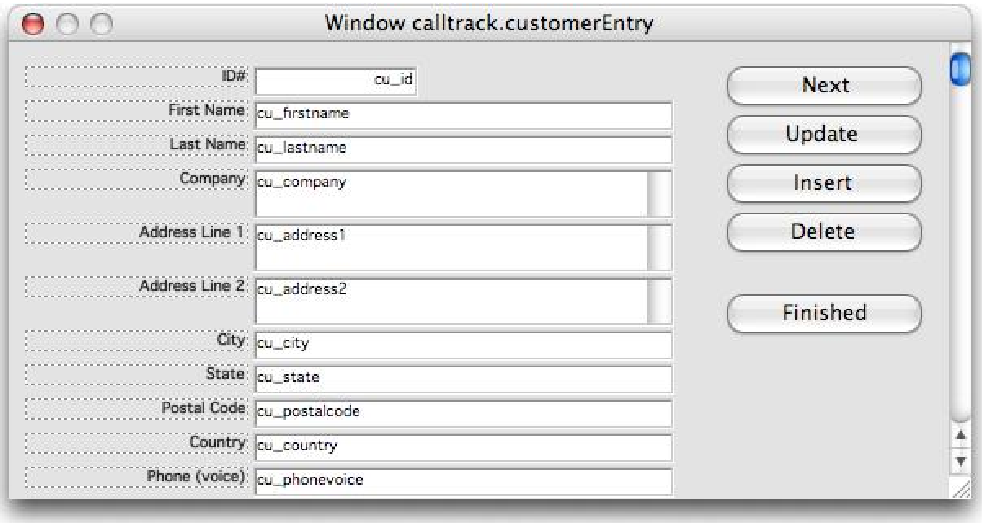


Figure 46. The Updated Window Labels.

3. If we click on a field or other object in this design window, "handles" appear around the object as shown in **Figure 47**.

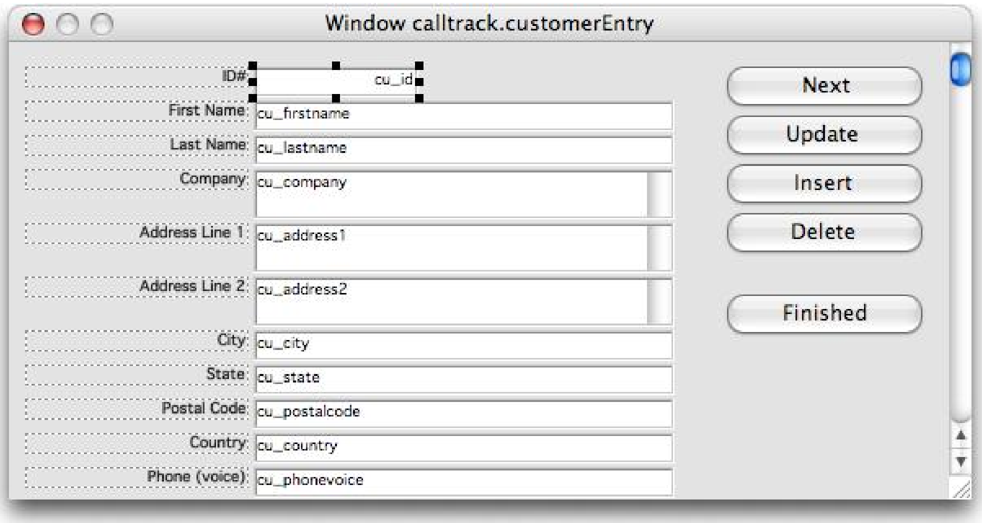


Figure 47. Selecting a Field.

Moving the cursor over a handle changes the cursor icon to a resize arrow. This shows that the object can be resized by dragging the handle. Dragging the handle allows us to resize the object in the direction implied by the position of the handle on the object. With a few objects given more appropriate sizes, the window could look like the one in **Figure 48**.

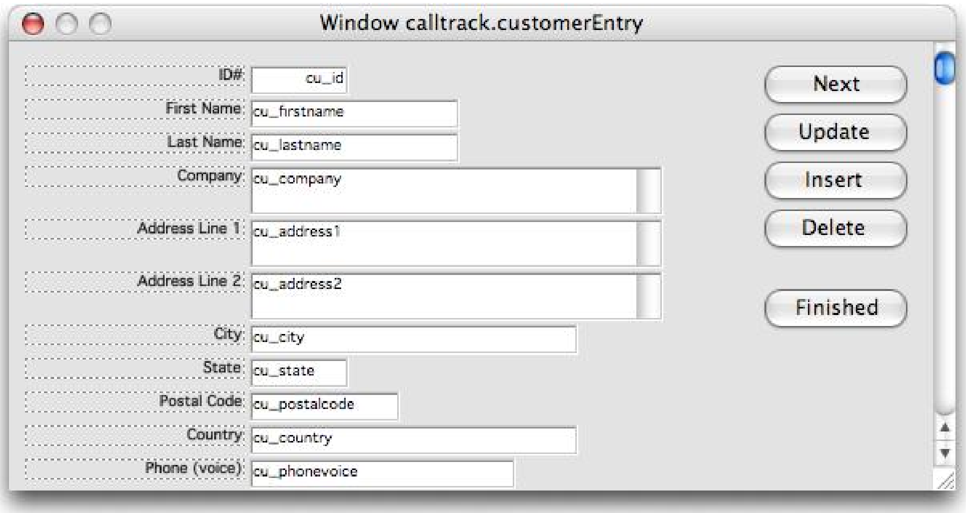
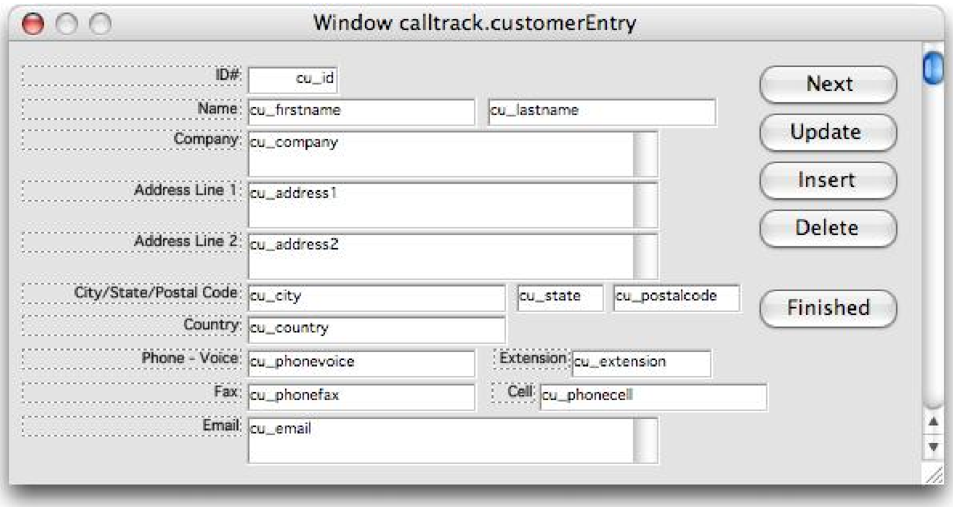


Figure 48. The Reformatted Window.

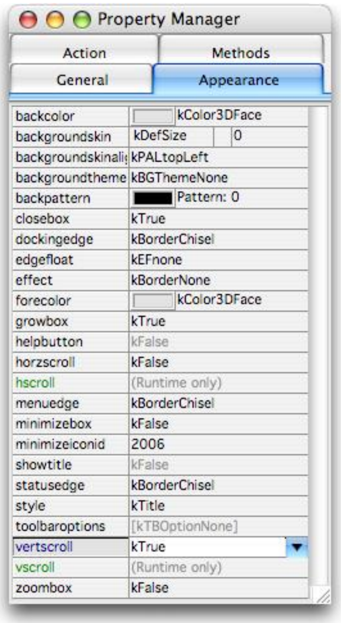
4. We can also move an object to a new position on the window by selecting the object and then dragging with the cursor inside the object. With a few more adjustments, the window could look like the one in **Figure 49**.



Window calltrack.customerEntry
ID#: cu_id
Name: cu_firstname cu_lastname
Company: cu_company
Address Line 1: cu_address1
Address Line 2: cu_address2
City/State/Postal Code: cu_city cu_state cu_postalcode
Country: cu_country
Phone - Voice: cu_phonevoice
Extension: cu_extension
Fax: cu_phonefax
Cell: cu_phonecell
Email: cu_email
Next
Update
Insert
Delete
Finished

Figure 49. The Window Further Redesigned.

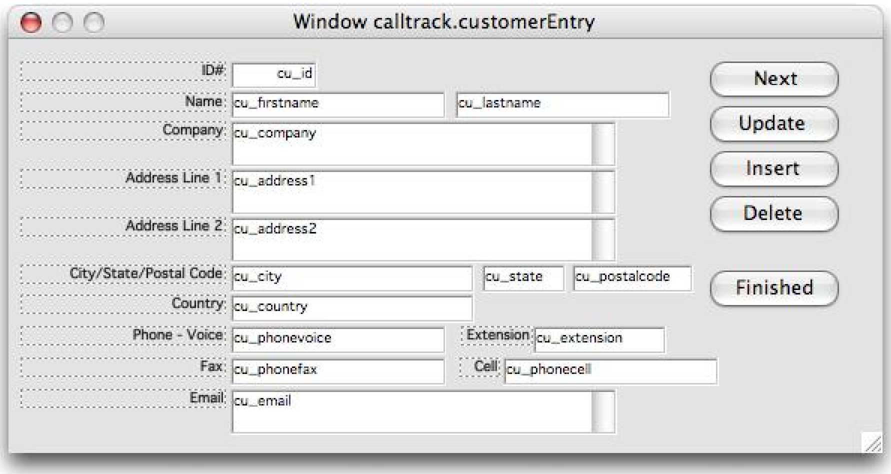
5. Having consolidated the objects on this window to require less space, it is no longer necessary to have a vertical scroll bar on the window. We can remove this by modifying one of the properties of the window using the Property Manager, shown in **Figure 50**.



backcolor	kColor3DFace
backgroundskin	kDefSize 0
backgroundskinali	kPALtopLeft
backgroundtheme	kBGThemeNone
backpattern	Pattern: 0
closebox	kTrue
dockingedge	kBorderChisel
edgefloat	kEFnone
effect	kBorderNone
forecolor	kColor3DFace
growbox	kTrue
helpbutton	kFalse
horzscroll	kFalse
hscroll	(Runtime only)
menuedge	kBorderChisel
minimizebox	kFalse
minimizeiconid	2006
showtitle	kFalse
statusedge	kBorderChisel
style	kTitle
toolbaroptions	[kTBOptionNone]
vertscroll	kTrue
vscroll	(Runtime only)
zoombox	kFalse

Figure 50. Modifications via the Property Manager.

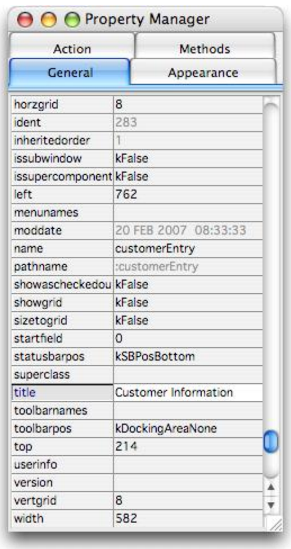
This shows the Appearance tab of the Property Manager for the window and the choices we have for the **vertscroll** property. If we set its value to **kFalse**, the scroll bar is removed. The window then looks like the one in **Figure 51**.



ID#:	cu_id		
Name:	cu_firstname	cu_lastname	
Company:	cu_company		
Address Line 1:	cu_address1		
Address Line 2:	cu_address2		
City/State/Postal Code:	cu_city	cu_state	cu_postalcode
Country:	cu_country		
Phone - Voice:	cu_phonevoice	Extension:	cu_extension
Fax:	cu_phonefax	Cell:	cu_phonecell
Email:	cu_email		

Figure 51. The Window without a Scroll Bar.

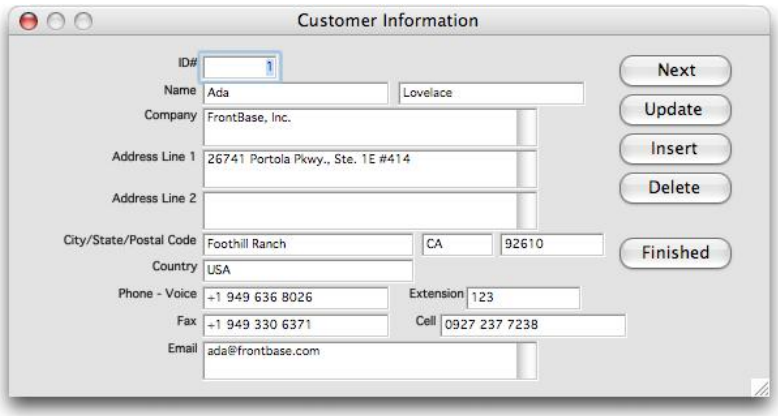
6. The title shown on the window instance is also a property of the window, but under the **General** tab. We can change this to read **Customer Information** instead of **New Window 1**, as shown in **Figure 52**.



horzgrid	8
ident	283
inheritedorder	1
issubwindow	kFalse
issupercomponent	kFalse
left	762
menunames	
moddate	20 FEB 2007 08:33:33
name	customerEntry
pathname	:customerEntry
showascheckedou	kFalse
showgrid	kFalse
sizetogrid	kFalse
startfield	0
statusbarpos	kSBPosBottom
superclass	
title	Customer Information
toolbarnames	
toolbarpos	kDockingAreaNone
top	214
userinfo	
version	
vertgrid	8
width	582

Figure 52. Modification of the Window Title.

The window instance will then look like the one in **Figure 53**.



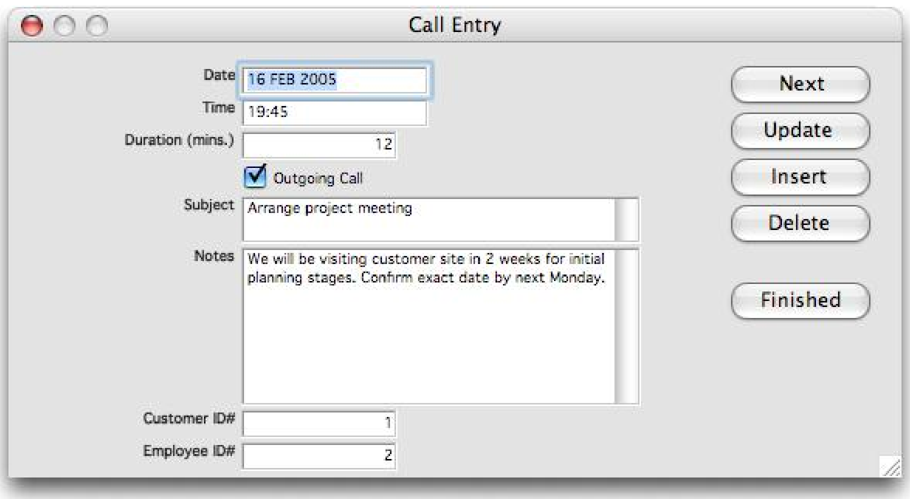
ID#	1
Name	Ada / Lovelace
Company	FrontBase, Inc.
Address Line 1	26741 Portola Pkwy., Ste. 1E #414
Address Line 2	
City/State/Postal Code	Foothill Ranch / CA / 92610
Country	USA
Phone - Voice	+1 949 636 8026
Extension	123
Fax	+1 949 330 6371
Cell	0927 237 7238
Email	ada@frontbase.com

Figure 53. The Customer Information Window.

This is still not the ultimate look for this window, but you can see that we can make many changes. All of the changes we have made are purely cosmetic and none of them affect the operation of the window.

7. Now follow the same steps as above and create a window for managing employee data. Name that window **employeeEntry** and give it a title of **Employee Information**. Add a few employee records to the database to test whether this window works as expected. Make sure that the employee id is unique for each record.

8. Again, follow the same steps as above and create a window for managing calls data. Name the window **phonecallEntry** and give it a title of **Call Entry**. To alter the label associated with the **ph_outgoing** checkbox field you will have to select that field and then use the Property Manager to update the text property under the General tab. Once you have the window formatted, add a few employee records to the database to test whether this window works as expected. Make sure you provide values for both the date and the time fields* and that the primary key combination (ph_date, ph_time, ph_cu_id, ph_em_id) is unique for each record. A cleaned up version of this window with a record entered is shown in **Figure 54**.



Date: 16 FEB 2005
Time: 19:45
Duration (mins.): 12
☑ Outgoing Call
Subject: Arrange project meeting
Notes: We will be visiting customer site in 2 weeks for initial planning stages. Confirm exact date by next Monday.
Customer ID#: 1
Employee ID#: 2

Figure 54. The Call Entry Window.

Notice how we can specify the ID number of a customer and an employee using this window, but that we cannot see any information about either party. There are many ways to make this data entry window more sophisticated but none of them are available through wizards in Omnis Studio. The programming techniques for providing this functionality are beyond the scope of this simple tutorial.

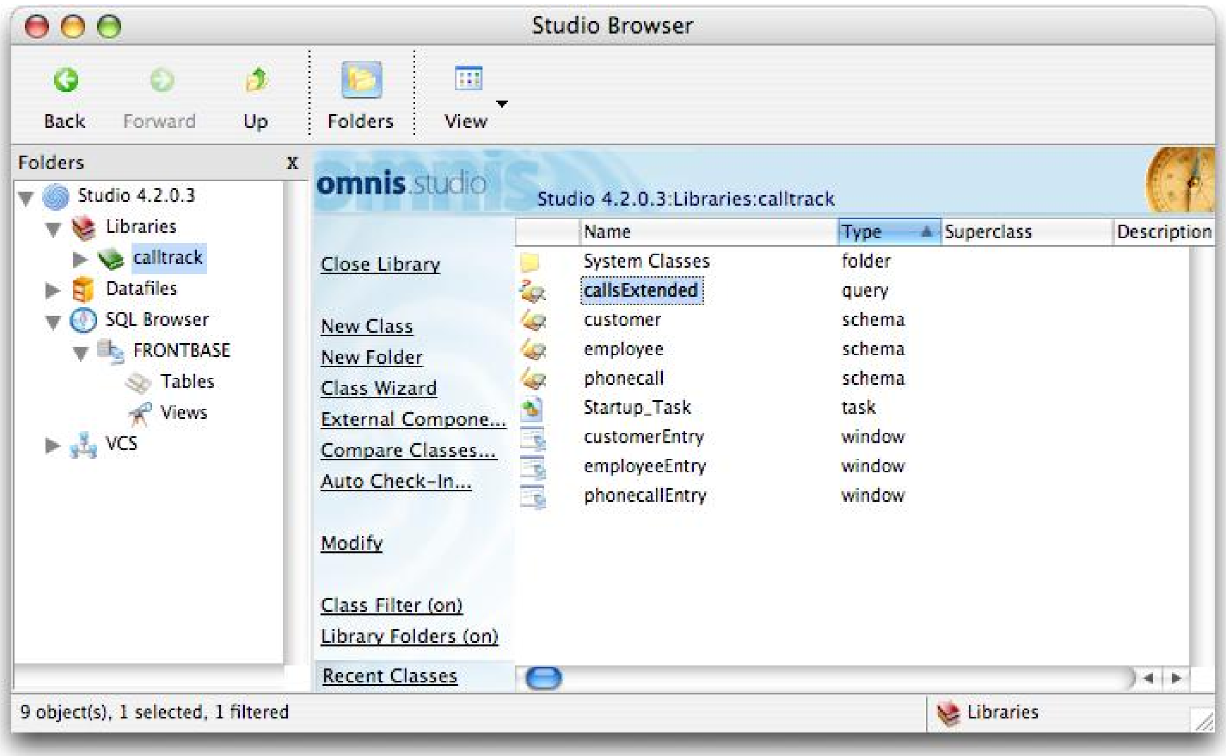
What we can do, though, is set up a window to simply query this cross section of the three tables, but this requires that we first create a query class to define the data we wish to retrieve.

* If you don't provide a value for these fields when you insert a record then this will break the UPDATE and DELETE operations. Why? Well, Omnis rolls a SQL statement to send to the server to do the update/delete. If the values are not defined, i.e. NULL, then the SQL will not work with FrontBase. Of course, if we weren't using wizards or we tweaked the Omnis code then we could fix the problem.

1.9 Creating a Query Class

A query class is similar to a schema class in that it is a collection of column definitions. A query class allows us to specify either a subset of the columns from an existing schema class or a combination of columns from multiple schema classes. In addition, we can provide a WHERE clause or other supplemental query text to further specify which records come within the scope of the query.

1. We create a new query class in a similar way to that used to create our schemas. Make sure the Studio Browser shows our library components in the main window. If not, click the calltrack library in the Libraries branch of the Folders tree list. Then click **New Class** followed by **Query** and give the new query a name of **callsExtended**. The new query will be listed in the browser window as in **Figure 55**.



Name	Type	Superclass	Description
System Classes	folder		
callsExtended	query		
customer	schema		
employee	schema		
phonecall	schema		
Startup_Task	task		
customerEntry	window		
employeeEntry	window		
phonecallEntry	window		

9 object(s), 1 selected, 1 filtered

Figure 55. The callsExtended Query Class.

2. Open our new query class by double-clicking on its icon in the browser. The Query Class Editor window that appears is shown in **Figure 56**.

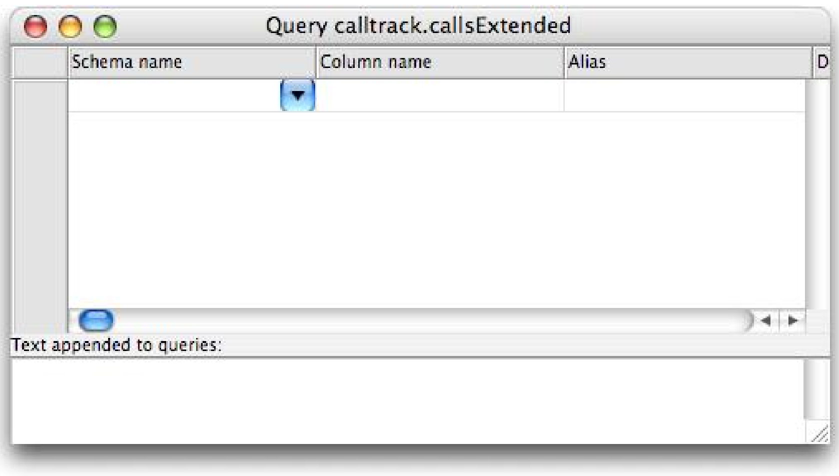


Figure 56. The Query Class Editor.

3. We can only specify columns from existing schema classes in this editor. To do so, we select a schema from the list provided in the **Schema name** column of the editor. Select **phonecall** as shown in **Figure 57**.

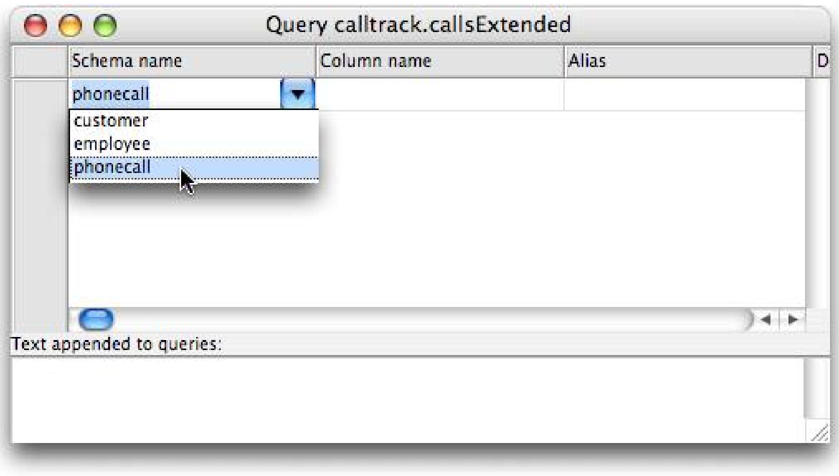


Figure 57. Selecting the Schema.

4. Once a schema has been selected for a line in the editor's grid, we can select a column from that schema in the **Column name** column of the editor. Choose **ph_date** as shown in **Figure 58**.

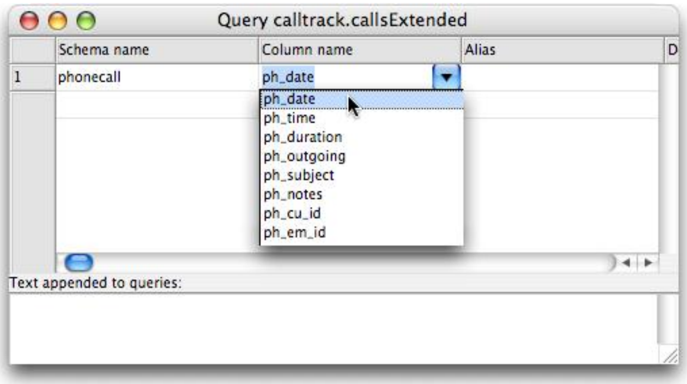


Figure 58. Selecting the ph_date Column.

5. Continue adding column names according to the following table. Once finished the Query Class Editor should look like the one in **Figure 59**.

Schema name	**Column name**
phonecall	ph_date
phonecall	ph_duration
phonecall	ph_subject
customer	cu_id
customer	cu_firstname
customer	cu_lastname
employee	em_id
employee	em_firstname
employee	em_lastname

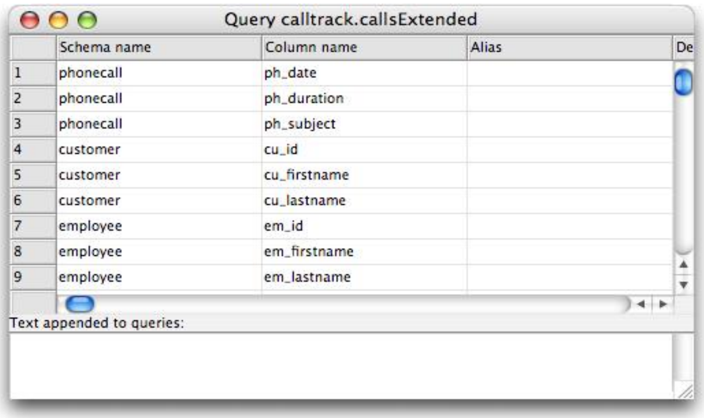


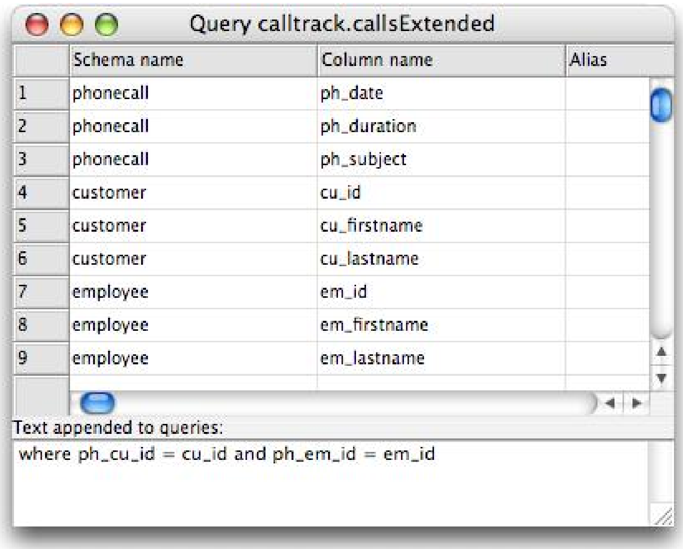
Figure 59. The Query Class Column Definitions.

The **Alias** column is used to resolve conflicts between columns with the same name in different schemas. Since we have given all our schema columns a prefix based on the table name we don't have to do anything here. If a column could not be uniquely identified then an alias name would be required.

6. The field at the bottom of the editor window labeled "**Text appended to queries**" is for specifying how our records are to be combined. As the name implies, the value specified in this field is concatenated directly onto the basic SELECT statement generated by this query. Enter the following text into this field.

```
where ph_cu_id = cu_id and ph_em_id = em_id
```

This text specifies that we want the customer and employee records that match their corresponding _id values in the calls record. The editor window should now look like it does in **Figure 60**.



	Schema name	Column name	Alias
1	phonecall	ph_date	
2	phonecall	ph_duration	
3	phonecall	ph_subject	
4	customer	cu_id	
5	customer	cu_firstname	
6	customer	cu_lastname	
7	employee	em_id	
8	employee	em_firstname	
9	employee	em_lastname	

Text appended to queries:

```
where ph_cu_id = cu_id and ph_em_id = em_id
```

Figure 60. The Completed Query Class.

□
We have now created a Query Class. The next step is to create a window based upon it.

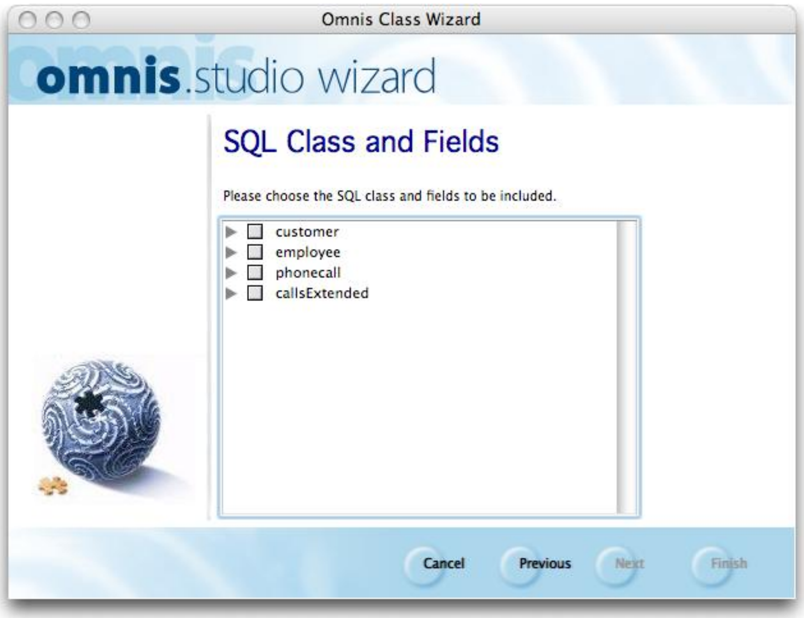
1.10 Creating a Window for a Query Class

We can create a window for our query class using the same wizard we used for creating windows for our schema classes. There will be only one slight change in the process, in that the wizard will now offer our query class as well as the schema classes for selection.

1. Select the **Class Wizard** from the left hand option list in the main browser window and then the **Window...** class type.

2. Select the SQL Form Wizard, name the new window class **extCallsView** and then click **Create**.

3. Select "**One field per column based on schema or query class**" on the **Window Type** screen as we did before and click the **Next** button. The **SQL Classes and Fields** panel appears, as shown in **Figure 61**, with our new **callsExtended** query class included in the list.



Please choose the SQL class and fields to be included.

customer
employee
phonecall
callsExtended

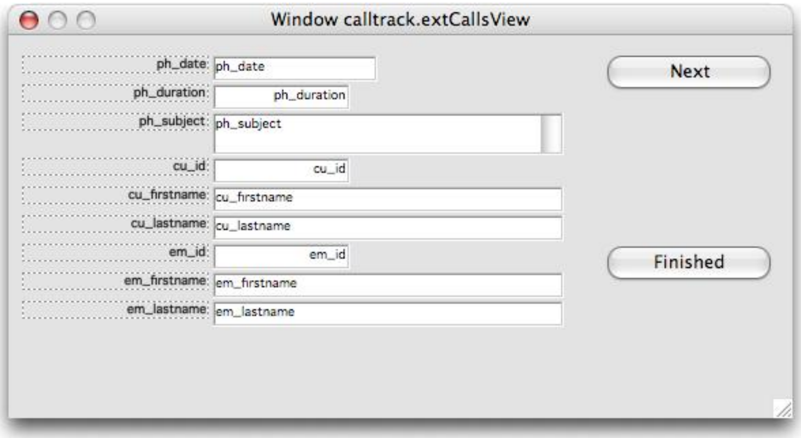
Figure 61. The SQL Class and Fields Panel.

4. Select **callsExtended** from the list and click the **Next** button. The **SQL Session** panel will appear as before.

5. Select **FRONTBASE** and click the **Next** button. The **Window Themes** panel will appear as before.

6. Select **Standard (3D Inset)** and click the **Next** button.

7. Click the **Finish** button on the **Ready To Build** panel. After a few moments, the finished window class will appear in its editor as shown in **Figure 62**.

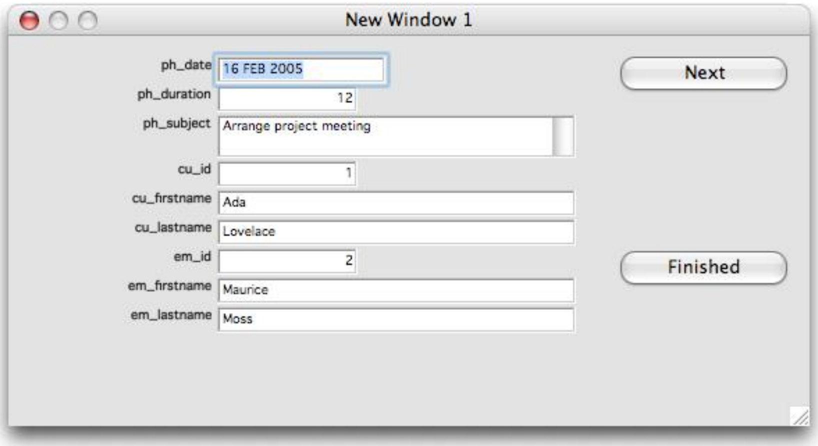


ph_date: ph_date
ph_duration: ph_duration
ph_subject: ph_subject
cu_id: cu_id
cu_firstname: cu_firstname
cu_lastname: cu_lastname
em_id: em_id
em_firstname: em_firstname
em_lastname: em_lastname

Figure 62. The Calls Extended Window.

8. Notice that since we based this window on a Query Class, there are no buttons for Insert, Update or Delete. We can only use this window to browse through calls records.

9. Now let's open a test instance of our new window and observe how it works. Press **Command-T/Control-T** and then click the **Next** button. If you had entered any calls records and those records contained valid customer and employee ID values, your window should look similar to the one shown in **Figure 63**.



ph_date 16 FEB 2005
ph_duration 12
ph_subject Arrange project meeting
cu_id 1
cu_firstname Ada
cu_lastname Lovelace
em_id 2
em_firstname Maurice
em_lastname Moss

Figure 63. The Calls Extended Test Instance.

1.11 Summary

With a minimum of effort, we have created a simple three table database application using Omnis Studio as a front end to FrontBase. We were able to use Omnis Studio for everything from defining tables and their columns to entering data and querying the database.

A more complete application would include menus for navigating through the application, reports for summarizing data, and more sophisticated windows with controls that offer more power to the end user.

Using Omnis Studio, we can even deploy a complex application directly to the worldwide web using techniques like the ones we used in this example - not with multiple HTML pages and hyperlinks, but with the entire dynamic multi-window application running in a single web page!