# 4D Mobile

Wakanda, published by 4D SAS, is a platform for developing and publishing web applications entirely based on standard technologies such as JavaScript and HTML5.

You can use the "4D Mobile" architecture to set up a direct link between 4D and Wakanda. With this configuration, you combine the graphical and functional richness of the latest generation of Wakanda Web interfaces with the power of your 4D databases.

If you want to create your first link between 4D and Wakanda right away, check in the **Configuration** paragraph that you have an appropriate configuration and then refer to the **Step-by-step Example** section.

# 4D Mobile Architecture

## Configuration

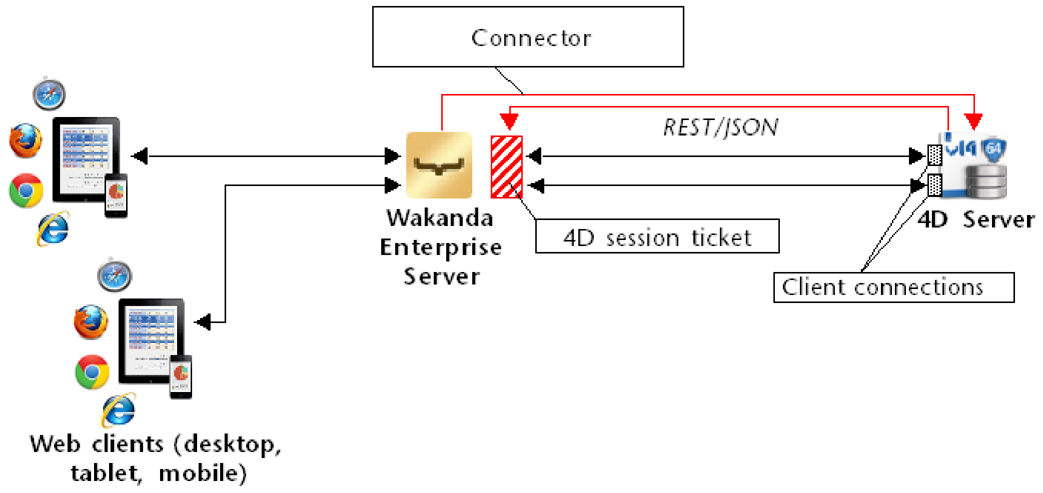
To set up an architecture that uses the 4D / Wakanda connector, you need:

- **4D** single-user (*Professional* version) to develop and test your solution using the 4D Mobile connector (three simultaneous 4D Mobile connections are allowed in this context) or **4D Server** with a 4D Mobile Expansion pack (two 4D Mobile connections allowed).
- **Wakanda Enterprise Server** as well as **Wakanda Enterprise Studio** for development. You can download both of these applications from the Wakanda download page (*Enterprise* tab).
- a 4D database and a Wakanda application that you want to communicate with each other.

On the 4D side, you have to configure every table, attribute and method to be accessed by the Wakanda applications (see the **Configuring the 4D Database** section).

## Description

The 4D Mobile architecture can be represented as follows:


Connector
REST/JSON
Wakanda Enterprise Server
4D session ticket
4D Server
Client connections
Web clients (desktop, tablet, mobile)

When the Wakanda solution is started, the Wakanda Enterprise server establishes a link with the 4D Server according to the settings defined in the "Connect to Remote Datastore" dialog box or using JavaScript connection methods. Once the connection is accepted by the 4D Server (see the **Control of REST accesses** paragraph), a 4D Mobile session "ticket" is delivered to the Wakanda server. This ticket will be used by the Wakanda server for all subsequent requests.

By means of this link, the Wakanda server can potentially access two types of resources in the 4D database:

- tables and their attributes (including their data)
- project methods

When authorized these resources are used directly on the Wakanda side, just as if they belonged to the local catalog of the Wakanda application (their access is transparent from the Wakanda application).

When a Web client sends a request to the Wakanda server that requires access to the 4D database, this request is sent to the 4D server using the current ticket and a 4D Mobile connection is opened on the 4D Server machine. This connection remains open as long as the user is performing requests and closes by default after a 60-minute timeout. This default timeout can be changed in the initial connection parameters.

During the session, if the number of licenses corresponding to the number of authorized 4D Mobile connections on the 4D Server is reached, an error message is returned to the Wakanda server.

# Step-by-step Example

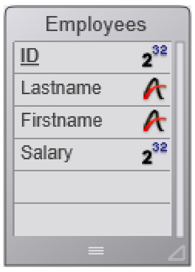
This section is intended to help familiarize you with Wakanda / 4D connector functionality by means of a step-by-step example. We are going to:

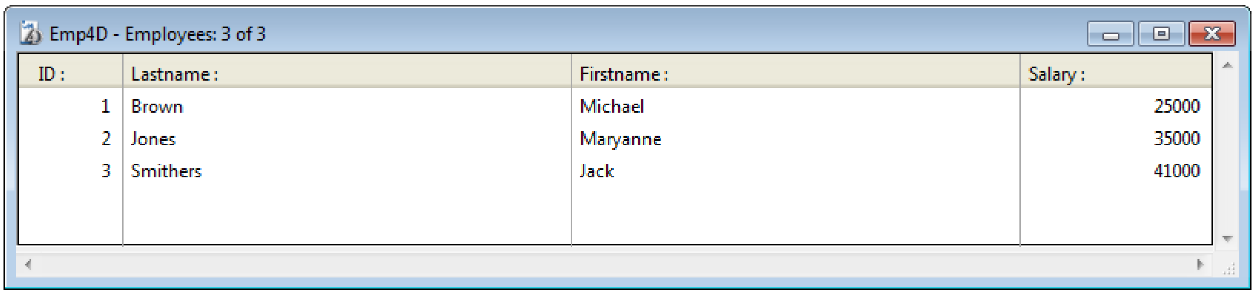
- create and configure a 4D database
- create a single-page Wakanda application
- display data from the 4D database in the Wakanda page.

To keep the example simple, we're going to use a 4D application and a Wakanda application that are running on the same machine. Of course, you could also use a remote architecture.

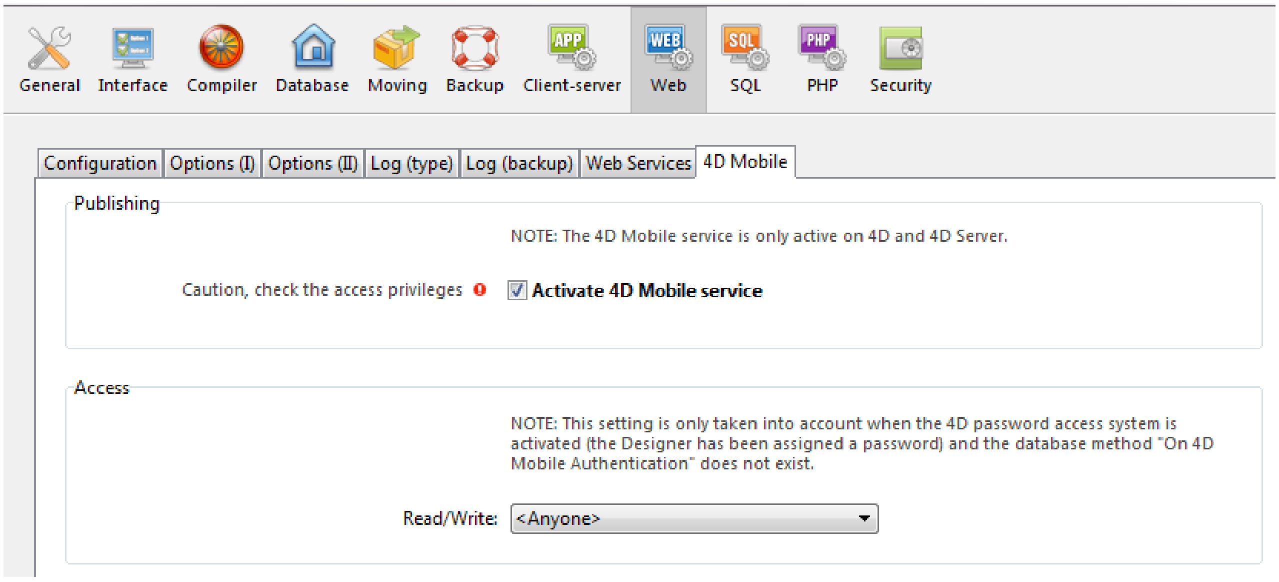
## 1- Creating and configuring the 4D database

1. Launch your 4D or 4D Server application and create a new database.
   You can name it "Emp4D", for example.
2. In the Structure editor, create an [Employees] table and add the following fields to it:
   - Lastname (Text)
   - Firstname (Text)
   - Salary (Longint)

   The "Expose with 4D Mobile Service" attribute is checked by default for the table and every field; do not change this setting.
3. Click on the **Tables** button and let 4D create default forms, then create a few employees:
   
   Emp4D - Employees: 3 of 3

   | ID : | Lastname : | Firstname : | Salary : |
   |---|---|---|---|
   | 1 | Brown | Michael | 25000 |
   | 2 | Jones | Maryanne | 35000 |
   | 3 | Smithers | Jack | 41000 |
4. Display the **Web** page of the Database Settings dialog box and click on the **4D Mobile** tab.
5. Check the "Activate 4D Mobile service" option then click on **OK**:



NOTE: The 4D Mobile service is only active on 4D and 4D Server.

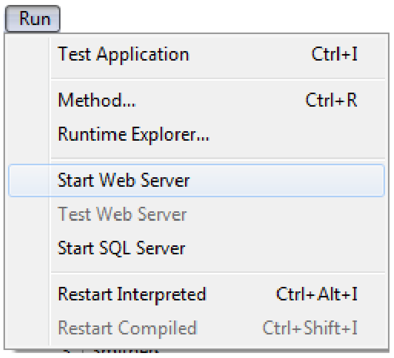
Caution, check the access privileges

[✓] Activate 4D Mobile service

NOTE: This setting is only taken into account when the 4D password access system is activated (the Designer has been assigned a password) and the database method "On 4D Mobile Authentication" does not exist.

Read/Write: <Anyone>

6. Select **Start Web Server** in the **Run** menu:

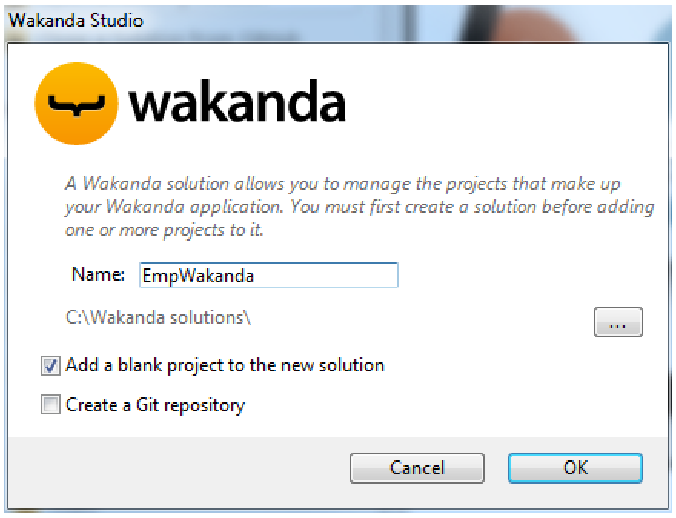


The 4D database is now ready to respond to 4D Mobile requests from Wakanda. Note that in order to simplify this example, we are not controlling 4D Mobile accesses. However, in a context of production or open architecture, it is indispensable to secure 4D Mobile (via REST) accesses (see the **About 4D Mobile application security** section).

## 2 - Creating the Wakanda application

1. Launch the "Wakanda Enterprise Studio" application and click on the **Create a New Solution** button:

2. In the creation dialog box, enter, for example, "EmpWakanda" and click on **OK**:



*A Wakanda solution allows you to manage the projects that make up your Wakanda application. You must first create a solution before adding one or more projects to it.*

Name: EmpWakanda

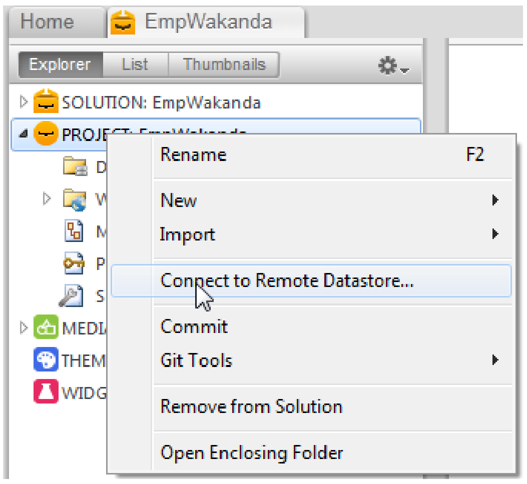
C:\Wakanda solutions\

[✓] Add a blank project to the new solution

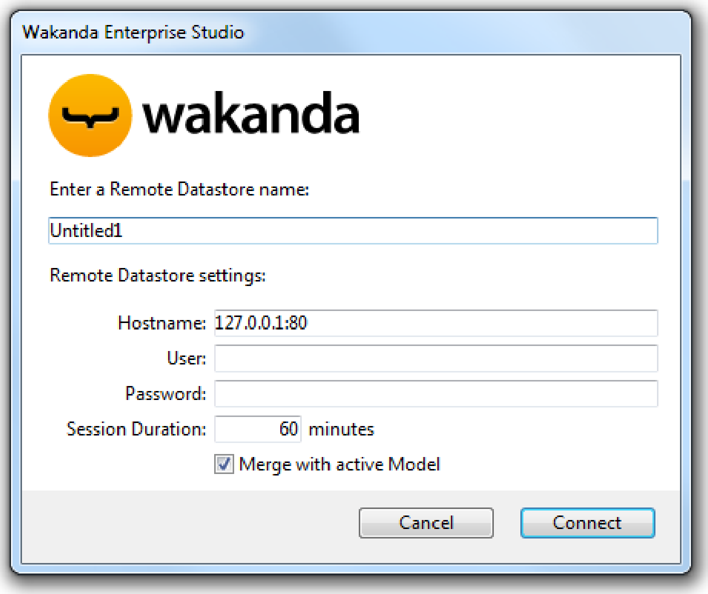
[ ] Create a Git repository

The application project is created and its default elements are shown in the Wakanda Studio Explorer, on the left side of the window.

3. Right click on the PROJECT row and select the **Connect to Remote Datastore...** command in the contextual menu.



The connection dialog box appears:


Wakanda Enterprise Studio

wakanda

Enter a Remote Datastore name:

Untitled1

Remote Datastore settings:

Hostname: 127.0.0.1:80

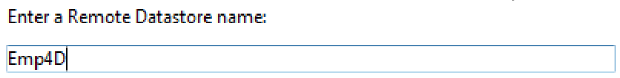
User:

Password:

Session Duration: 60 minutes

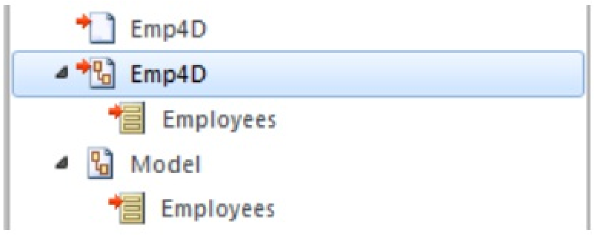
[✓] Merge with active Model

4. Enter a name for the link, for instance "Emp4D":


Enter a Remote Datastore name:

Emp4D

This is the local name of the link, as it will appear in Wakanda Enterprise Studio. You can enter any name but to keep things simple we are using the name of the 4D database.
5. (optional) If your 4D Server is located on a different machine than the Wakanda Enterprise Studio, enter its hostname or its IP address as the Hostname parameter. Otherwise, you can just keep the local address "127.0.0.1:80" (or "localhost").
6. Keep the other default parameters and click on the **Connect** button.
After a few moments, you can see that the "Emp4D" external model is listed in the files of the Wakanda application and the [Employees] table of the 4D application is listed in the datastore classes of the local model. External elements are indicated by a red arrow:



**Note:** The first Emp4D file contains the connection parameters.

## In case of problems...

If the table is not shown in the list at this time, check that:

- no third-party service or software (instant messenger, for instance) is in conflict with the publishing port of the 4D HTTP server (80 by default),
- on the 4D side, the 4D Web server is started, 4D Mobile services are activated and the table has been

exposed,
- the address passed in the "Hostname" parameter is valid.

To check whether the 4D server is actually responding to REST requests, you can try the following URLs in your browser:

```
<address>/rest/$catalog/$all
```

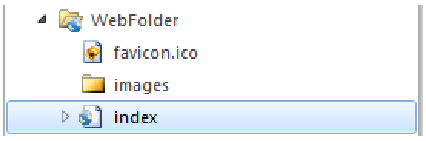
(returns all the tables exposed in 4D Mobile)

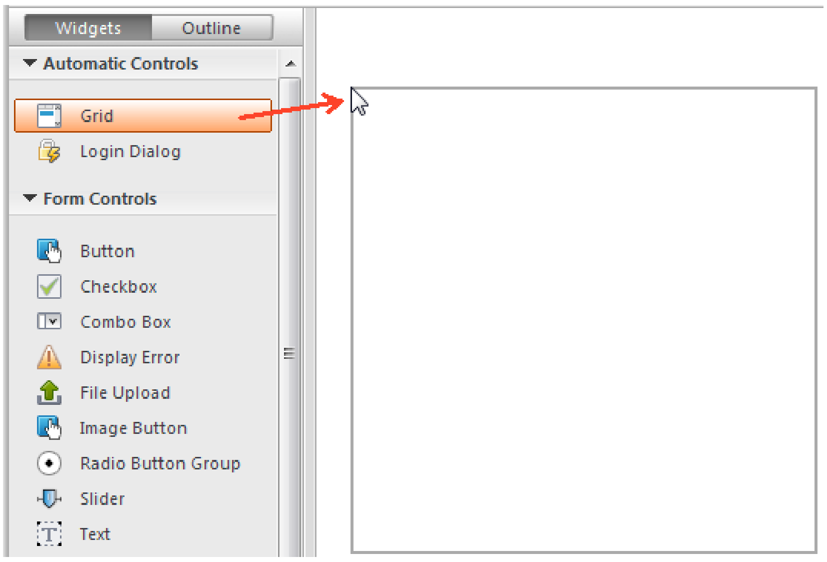
```
<address>/rest/my_table/my_method
```

(returns the result of the method - if it returns a result)

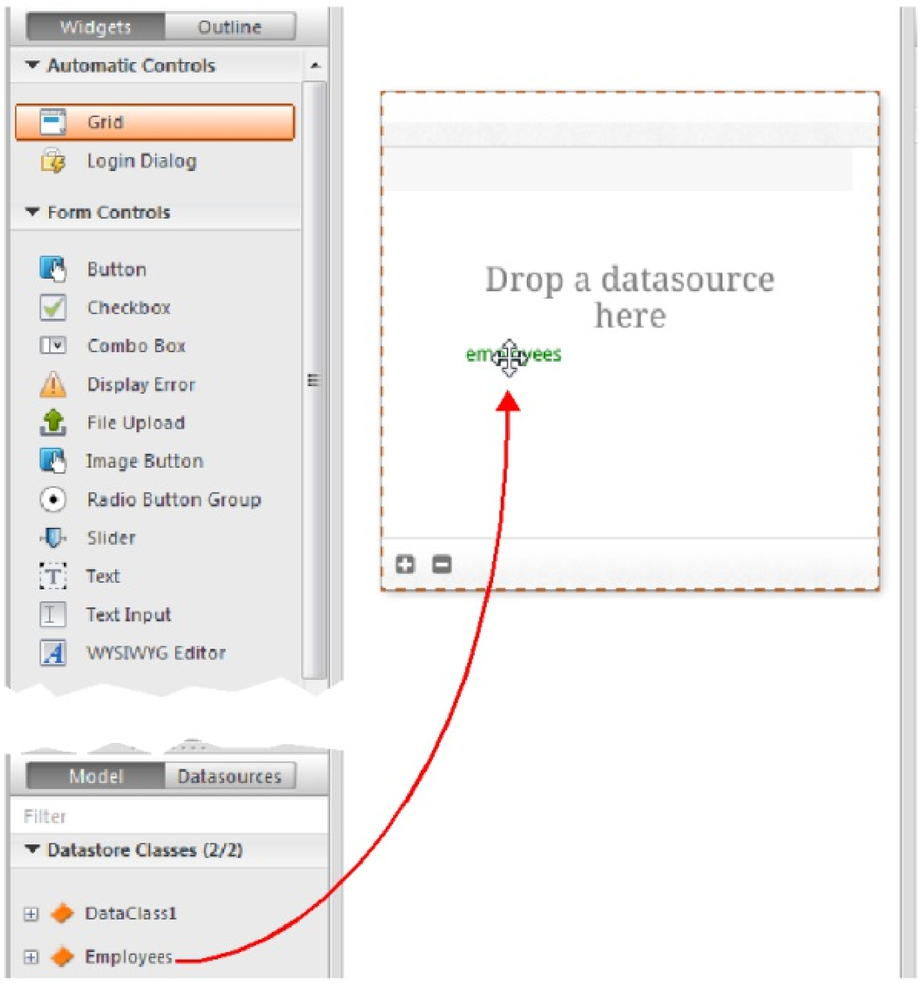
## 3 - Displaying 4D data using a Wakanda widget

Now we are going to associate a 4D table with a Wakanda widget by simple drag-and-drop, then launch the Wakanda Enterprise Server and view the data.

1. Open the "WebFolder" folder in the Explorer and double-click on the Index page to open the GUI Designer of Wakanda.


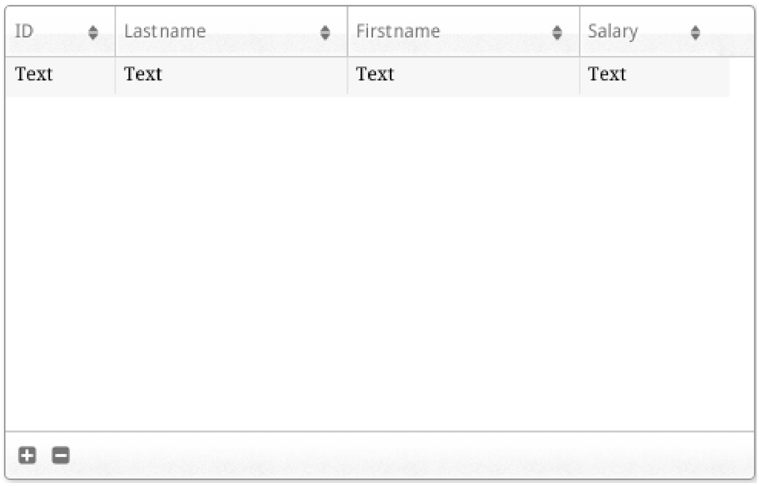
**Note:** The "WebFolder" contains elements intended for Web publication of your project. "Index" is the default page of the project.
2. In the list of Widgets, click on "Grid" and drop it in the work area:


3. In the list of Datastore Classes for the model, click on "Employees" and drop it onto the grid that you just created:



At this point, the editor automatically create a datasource for you based on the "Employees" class, which will manage the contents of the widget. This *datasource* is a JavaScript object managed by Wakanda, named "employees" by default, i.e. the class name with its first letter in lowercase.

The widget displays a preview of its contents. You can enlarge it to display all the fields of the datasource:



| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| Text | Text | Text | Text |

The association between the *datasource* and the widget is now established.

4. Click on the **Save** button in the tool bar of the editor.

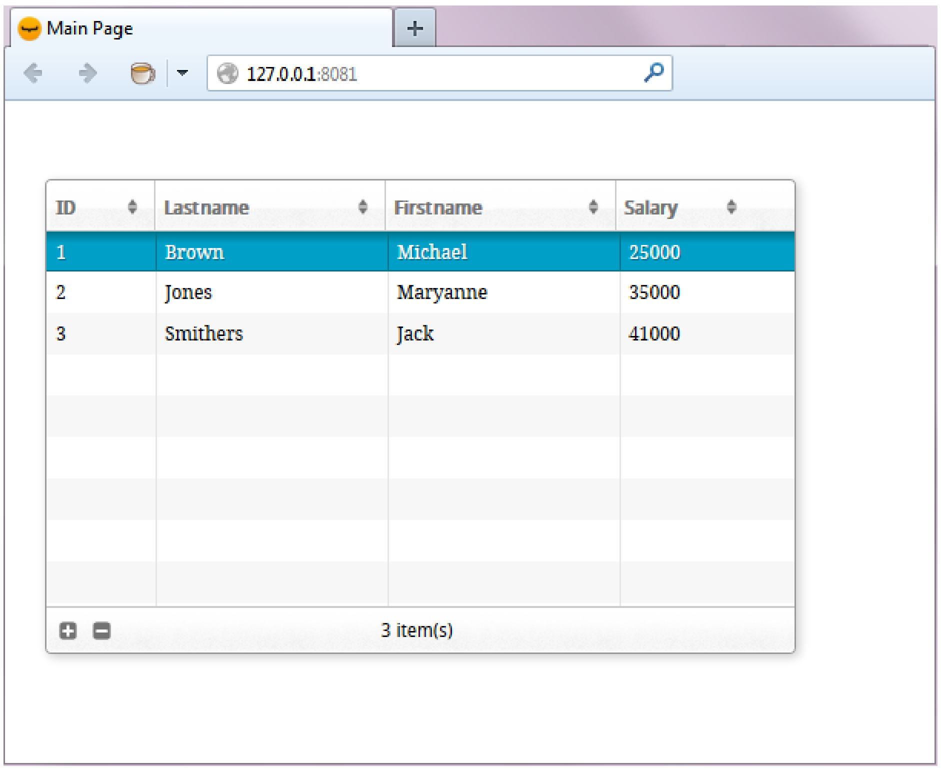
Now we are going to display the data using a browser.

5. Click on the **Run project** button in the tool bar of the Wakanda Enterprise Studio:

Run Project

This starts the Wakanda Enterprise Server and publishes the "EmpWakanda" application.

Thanks to the 4D Mobile link that we set up, the data of the 4D database is displayed in a window of your default browser:

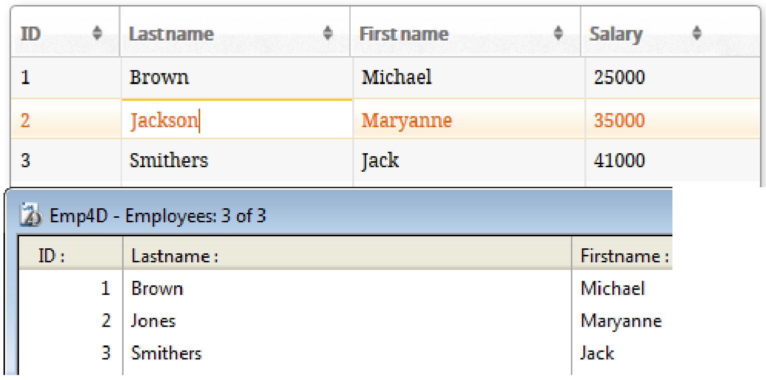

Main Page

127.0.0.1:8081

| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| 1 | Brown | Michael | 25000 |
| 2 | Jones | Maryanne | 35000 |
| 3 | Smithers | Jack | 41000 |

3 item(s)

You can test the dynamic properties of the link by changing data on the Web side. For instance, here we changed Maryanne Jones' last name to "Jackson" and this change was immediately reflected in 4D:



| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| 1 | Brown | Michael | 25000 |
| 2 | Jackson | Maryanne | 35000 |
| 3 | Smithers | Jack | 41000 |

Emp4D - Employees: 3 of 3

| ID : | Lastname : | Firstname : |
|---|---|---|
| 1 | Brown | Michael |
| 2 | Jones | Maryanne |
| 3 | Smithers | Jack |

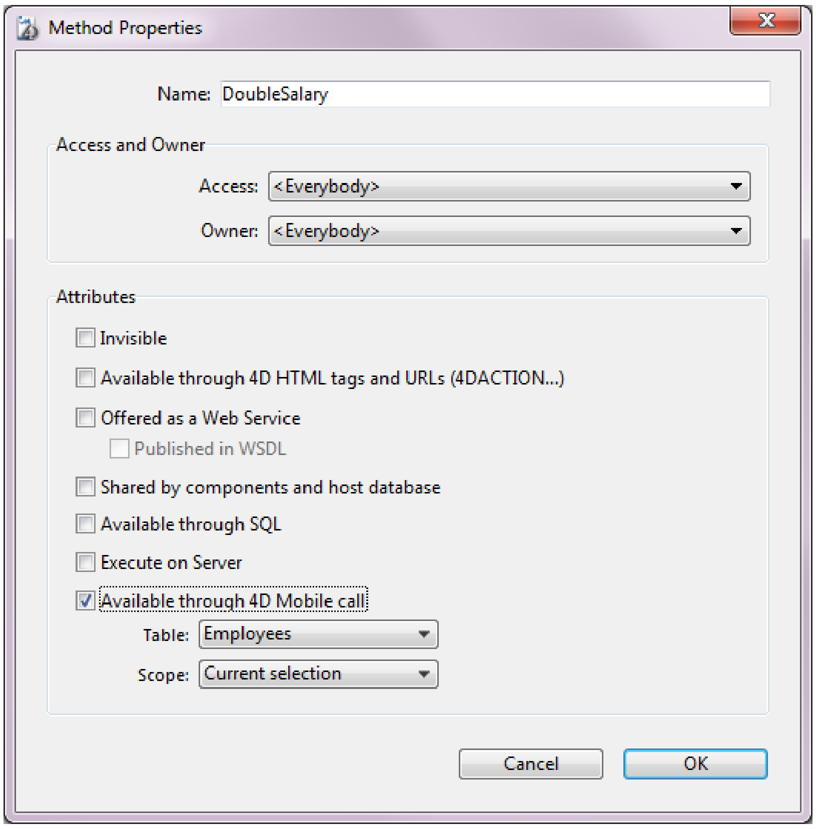
## 4 - Creating and calling a 4D method

Now we are going to create a very simple project method on the 4D side and execute it from our Web page. This method will double all the salaries.

1. On the 4D side, create a project method named ***DoubleSalary*** and enter the following code:

```
FIRST RECORD([Employees])
While(Not(End selection([Employees])))
   [Employees]salary:=[Employees]salary*2
   SAVE RECORD([Employees])
   NEXT RECORD([Employees])
End while
```

2. Set the 4D Mobile call properties for the method and click on **OK**:


Method Properties

Name: DoubleSalary

Access and Owner

Access: <Everybody>

Owner: <Everybody>

Attributes

- [ ] Invisible
- [ ] Available through 4D HTML tags and URLs (4DACTION...)
- [ ] Offered as a Web Service
  - [ ] Published in WSDL
- [ ] Shared by components and host database
- [ ] Available through SQL
- [ ] Execute on Server
- [✓] Available through 4D Mobile call

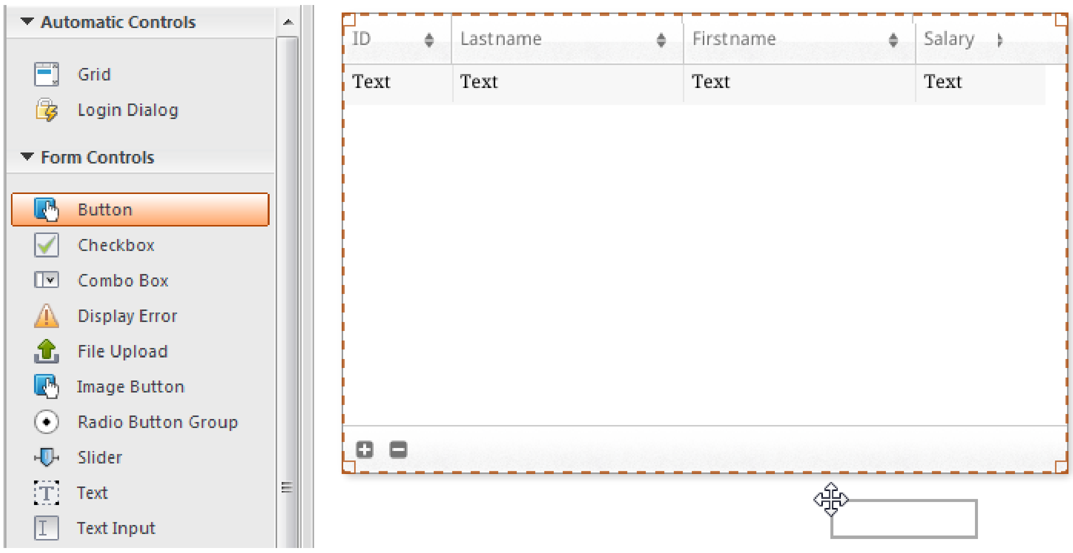
Table: Employees

Scope: Current selection

Cancel | OK

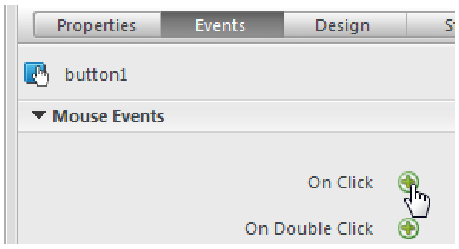
In Wakanda, class methods apply to one of the following contexts: the entity (record), the entity collection (selection) or the datastore class (all records). You will need to specify this context on the 4D side.

3. On the Wakanda Enterprise Studio side, return to the **Index** page in the *GUI Designer* and add a button from the list of widgets:



4. Double-click on the button and name it, for instance, "Double salaries":

5. Make sure that the "Double salaries" button is selected and then click on the **Events** button in the right-hand area of the *GUI Designer*.
6. Click on the icon for adding the "On Click" event:



The code editor appears and you can enter the code to be executed when the button is clicked. We are just going to call the ***DoubleSalary*** method of 4D and then, in the callback function (*onSuccess*), cause all the records to be reloaded.

7. Enter the following code:

```
    sources.employees.DoubleSalary({              onSuccess:function(event){
sources.employees.allEntities();            }});
```

In the code editor:

```
button1.click = function button1_click (event)
{
        sources.employees.DoubleSalary({
            onSuccess:function(event){
            sources.employees.allEntities();
        }});
};
```

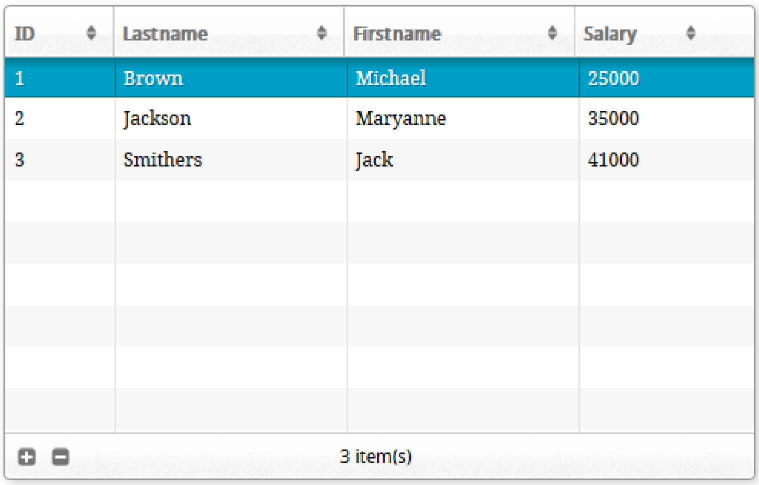
Note the lowercase "e" used for "employees"; we are using the datasource that was created automatically when the class was associated with the widget.

8. Click on the **Save** button in the tool bar of the editor.

   We can test calling the 4D method but beforehand you must reload the model on the Wakanda Enterprise Server.

9. Click on the **Reload Models** button in the tool bar of Wakanda Enterprise Studio.

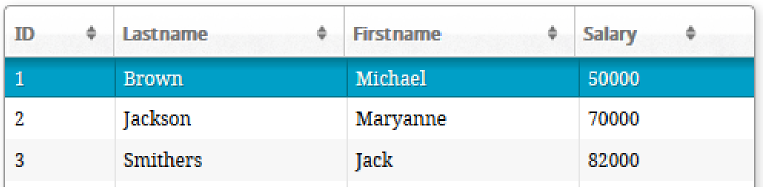
10. Refresh your browser page so that the **Double salaries** button appears and then click on this button:



| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| 1 | Brown | Michael | 25000 |
| 2 | Jackson | Maryanne | 35000 |
| 3 | Smithers | Jack | 41000 |

3 item(s)

You can see that the salary values have all doubled:



| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| 1 | Brown | Michael | 50000 |
| 2 | Jackson | Maryanne | 70000 |
| 3 | Smithers | Jack | 82000 |

Note that this example is simply intended to show how to use the Wakanda / 4D connector. The simplified methods shown here cannot be used in a production context.

# Configuring the 4D Database

For security and performance reasons, access to the tables, data and methods of the 4D database by means of 4D Mobile (Wakanda server) requests must be enabled and explicitly authorized. You must configure three levels of access:

- startup of 4D Mobile services,
- control of 4D Mobile accesses (optional but recommended),
- individually setting the exposure of each database object (table, field or project method) in 4D Mobile services according to your needs. By default:
  - all tables and all fields are accessible to 4D Mobile,
  - project methods are not accessible to 4D Mobile.

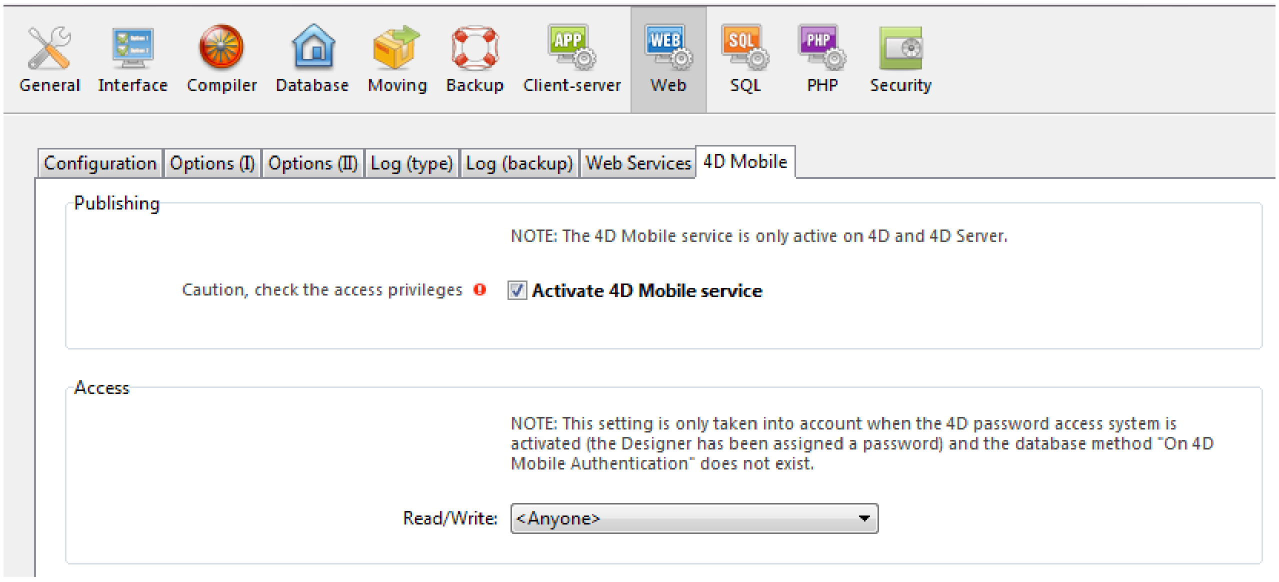
## Activating 4D Mobile services

By default, 4D Server does not respond to 4D Mobile requests. You must enable 4D Mobile services in order for these requests to be processed and so that the Wakanda / 4D connector can be used.

**Note:** 4D Mobile services use the 4D HTTP server, so you need to make sure that the 4D Web server or 4D Server is started.

To activate 4D Mobile services:

1. In the Database Settings, display the Web/4D Mobile page.
2. Check the **Activate 4D Mobile service** option:



NOTE: The 4D Mobile service is only active on 4D and 4D Server.

Caution, check the access privileges [✓] **Activate 4D Mobile service**

NOTE: This setting is only taken into account when the 4D password access system is activated (the Designer has been assigned a password) and the database method "On 4D Mobile Authentication" does not exist.

Read/Write: <Anyone>

The warning message "Caution, check the access privileges" is displayed to draw your attention to the fact that when 4D Mobile services are activated, by default access to database objects is free as long as the 4D Mobile accesses (via REST) have not been controlled (see below).

## Control of 4D Mobile accesses

Control of 4D Mobile accesses lets you authorize (or not) the opening of a session on the 4D side following a Wakanda request.

As part of a 4D Mobile access, the identifiers checked are the name and password sent during the connection request performed by:

- the "Connect to Remote Datastore" dialog box of the Wakanda Enterprise Studio

- the **mergeOutsideCatalog()**, **openRemoteStore()** or **addRemoteStore()** SSJS methods.

At the global level, there are two ways to control 4D Mobile accesses:

- either automatically, by means of 4D passwords,
- or by programming by means of the **On 4D Mobile Authentication database method**.

These two control modes are exclusive: if an **On 4D Mobile Authentication database method** is defined, the control of automatic accesses by 4D passwords is disabled.

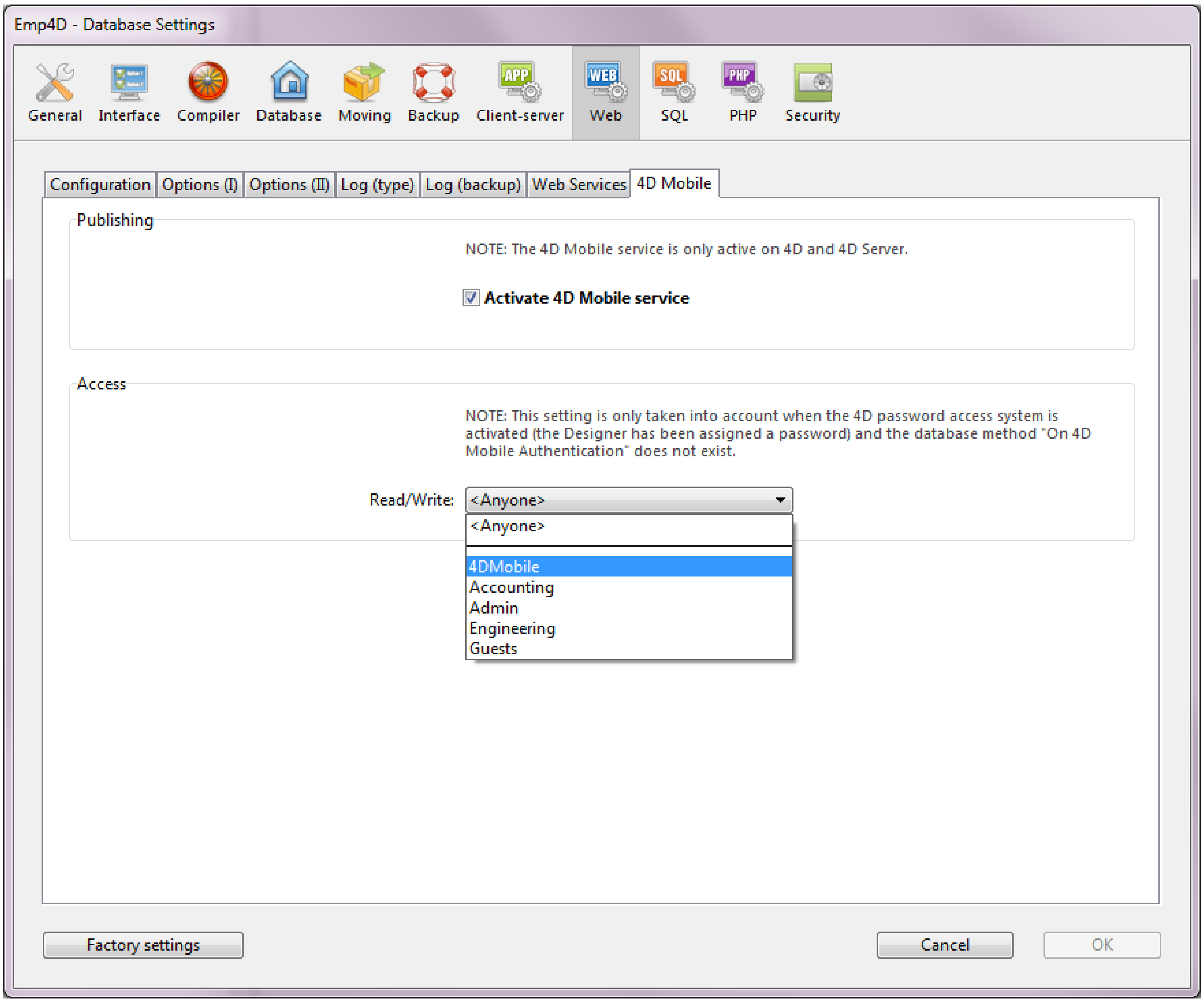
Warning: if neither of these two control modes are enabled, accesses to the database through 4D Mobile are always accepted (not recommended).

## Automatic controls by 4D password

In 4D, you can specify the group of 4D users that is authorized to establish the link to the 4D server from the Wakanda application.

To designate the logon account:

1. In the Database Settings, display the Web/4D Mobile page.
2. Choose the group to use in the "Read/Write" menu of the Access area:



NOTE: The 4D Mobile service is only active on 4D and 4D Server.

☑ Activate 4D Mobile service

NOTE: This setting is only taken into account when the 4D password access system is activated (the Designer has been assigned a password) and the database method "On 4D Mobile Authentication" does not exist.

Read/Write: <Anyone>

<Anyone>
4DMobile
Accounting
Admin
Engineering
Guests

By default, the menu displays **<Anyone>**, which means that 4D Mobile accesses are open to all users.

Once you have specified a group, only a 4D user account that belongs to this group may be used to access 4D by means of a Wakanda request -- to open a session on the 4D Server using the **mergeOustideCatalog()** method, for example. If an account is used that does not belong to this group, 4D returns an authentication error to the sender of the request.

Note that in order for this setting to take effect:

- the 4D password system must be activated (a password must have been assigned to the Designer),

- the **On 4D Mobile Authentication database method** must not be defined. If it exists, 4D does not take into account any access settings defined in the Database Settings.

### Using the On 4D Mobile Authentication Database Method

You can use the **On 4D Mobile Authentication database method** to control access rights for 4D Mobile requests sent to the Web server engine. When it is defined, it is automatically called by 4D or 4D Server when the HTTP server receives a 4D Mobile request.

When the request to open a 4D Mobile session comes from Wakanda Server (general case), the connection identifiers are provided in the header of the request. The **On 4D Mobile Authentication database method** is called so that you can evaluate these identifiers. You can use the list of users for the 4D database or you can use your own table of identifiers.

For more information, please refer to the **On 4D Mobile Authentication database method** description in the 4D *Language Referenc*e.

## Setting 4D objects exposed in 4D Mobile

Once 4D Mobile services are enabled in the 4D database, by default a 4D Mobile session can access all tables and fields of the database, and thus use their data. For example, if your database contains an [Employee] table, it is possible to write, on the Wakanda Server side:

```
var emp=ds.Employee.query("name == 'Martin'");     //Return all employees whose name field is ’Martin’
```

**Note:** 4D tables and/or fields that have the "Invisible" attribute are also exposed in 4D Mobile by default.

The Wakanda server can also access the project methods of the 4D database. However, by default, this access is disabled for security reasons.

If you want to customize the list of database objects accessible in 4D Mobile, you must:

- disable the exposure of each table and/or field that you want to hide,
- enable exposure of each project method to which you want to give access.

When a 4D Mobile request attempts to access an unauthorized resource (table or project method), 4D returns an error.

### Exposing tables

By default, all tables are exposed in 4D Mobile.

For security reasons, you may want to only expose certain tables of your database to 4D Mobile calls. For instance, if you created a [Users] table storing user names and passwords, it would be better not to expose it.

To modify the 4D Mobile exposure for a table:

1. Display the Table Inspector in the Structure editor and select the table you want to modify.
   By default, the **Expose with 4D Mobile Service** option is checked:

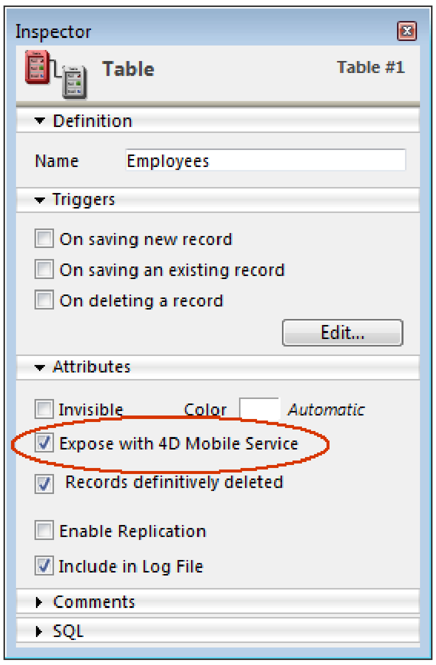

Inspector

Table — Table #1

Definition

Name: Employees

Triggers

- [ ] On saving new record
- [ ] On saving an existing record
- [ ] On deleting a record

Edit...

Attributes

- [ ] Invisible — Color: Automatic
- [x] Expose with 4D Mobile Service
- [x] Records definitively deleted
- [ ] Enable Replication
- [x] Include in Log File

Comments

SQL

2. Uncheck the **Expose with 4D Mobile Service** option.
   OR
   Check this option to expose a table.
   Do this for each table whose exposure needs to be modified.

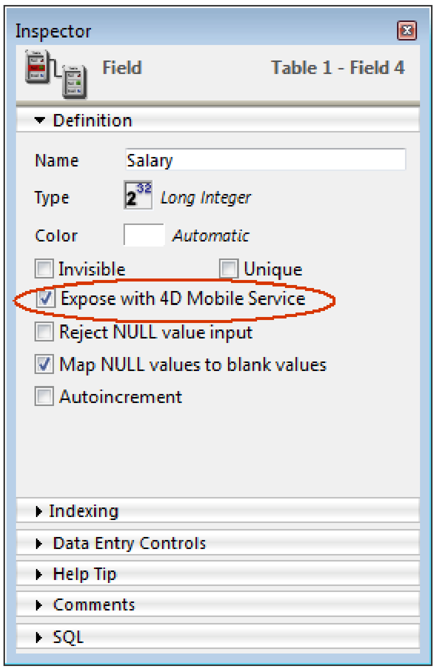
## Exposing fields

By default, all 4D database fields are exposed in 4D Mobile.

You may not want to expose certain fields of your tables to 4D Mobile. For example, you may not want to expose the [Employees]Salary field.

To modify the 4D Mobile exposure for a field:

1. Display the Field Inspector in the Structure editor and select the field you want to modify.
   By default, the **Expose with 4D Mobile Service** option is checked:


Inspector

Field — Table 1 - Field 4

Definition

Name: Salary

Type: Long Integer

Color: Automatic

- [ ] Invisible
- [ ] Unique
- [x] Expose with 4D Mobile Service
- [ ] Reject NULL value input
- [x] Map NULL values to blank values
- [ ] Autoincrement

Indexing

Data Entry Controls

Help Tip

Comments

SQL

2. Uncheck the **Expose with 4D Mobile Service** option for the field.
   OR
   Check this option to expose a field that was unchecked.
   Repeat this for each field whose exposure needs to be modified.

Note that in order for a field to be accessible through 4D Mobile, the parent table must be as well. If the parent table is not exposed, none of its fields will be, regardless of their status. Because of this, you can temporarily enable/disable 4D Mobile exposure for a table, while the individual values of the **Expose with 4D Mobile Service** option for each field remain unchanged.

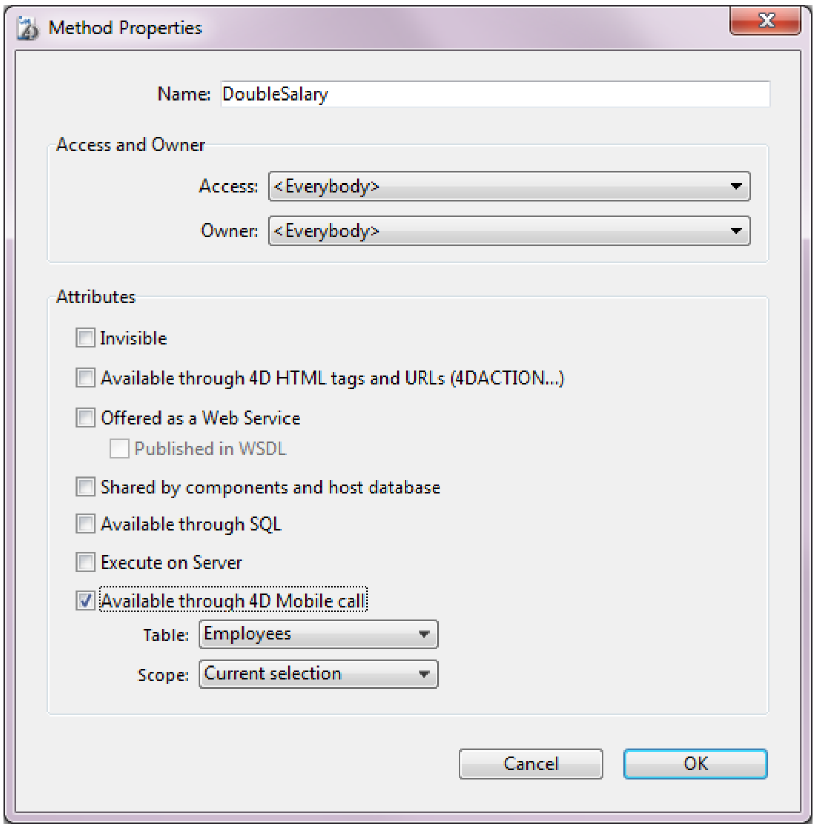
## Exposing project methods

No project methods are exposed in 4D Mobile by default.

You may want to make certain project methods of your 4D database accessible by means of 4D Mobile. To do this, you must check the appropriate option and define the Wakanda execution context of the method.

Note: If an access group is associated with the 4D method, you need to make sure that the 4D Mobile group is part of this group.

To set 4D Mobile exposure for a project method:

1. Display the "Method Properties" dialog box.
   **Note:** You can access the method properties dialog box using the context menu on the "Methods" page of the Explorer, or from the **Method/Method Properties...** menu in the Method editor.
2. Check the **Available through 4D Mobile call** option:
   
   Method Properties

   Name: DoubleSalary

   Access and Owner

   Access: <Everybody>

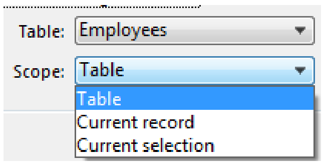
   Owner: <Everybody>

   Attributes

   ☐ Invisible
   ☐ Available through 4D HTML tags and URLs (4DACTION...)
   ☐ Offered as a Web Service
   ☐ Published in WSDL
   ☐ Shared by components and host database
   ☐ Available through SQL
   ☐ Execute on Server
   ☑ Available through 4D Mobile call

   Table: Employees

   Scope: Current selection

   Cancel OK
3. Define the Wakanda execution context for the project method by means of the **Table** and **Scope** menus.
   
   Table: Employees

   Scope: Table

   These settings are required in order to respect the logic of Wakanda. For more information about this point, refer to the following section.
4. Click on **OK** to validate the changes.
   Project methods available through 4D Mobile are listed in the "4D Mobile Methods" section of the 4D Explorer (see the “Explorer” paragraph below).

## Parent table and Scope of project methods

When you declare a project method available through 4D Mobile requests, you must explicitly declare its calling context using the **Table** and **Scope** parameters:

- **Table**: table to be attached with the project method. This setting is not linked directly to the use of the table's data, but allows you to designate the *datastore class* object through which you want to access the method using JavaScript code.
  The menu displays the list of database tables that are exposed in 4D Mobile. If the method specifically uses a table's data, you can select that table. If the method is not related to a single table, you can use any table that

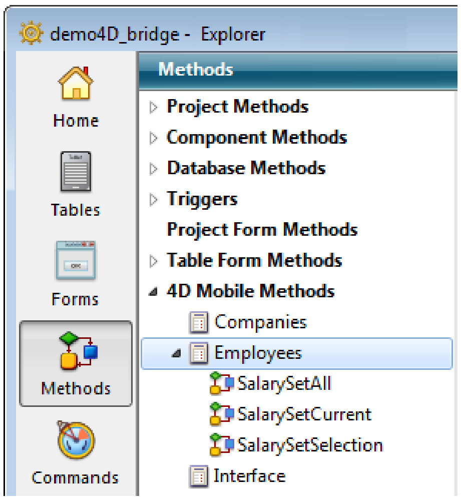
is exposed. Or, yet again, if you only want to expose methods that correspond with the business logic of your 4D application, you can create and expose a dedicated table, for example, a [4D MobileInterface] table and then associate all the project methods exposed in 4D Mobile with it.

- **Scope**: span of records where method is applied. This declaration is necessary because, on the Wakanda side, methods are properties of JavaScript objects and can only be called by means of these objects. Each 4D method that is exposed must be explicitly associated with the database context where it will be called: **Table**, **Current record** and **Current selection**.
    - **Table**: This option indicates that the 4D method will be executed using all the records of the designated table.
      On the Wakanda side, the method will be called on an object of the *Datastore class* type, for example *ds.MyTable.MyMethod*.
    - **Current record**: This option indicates that the 4D method will be executed using the current record of the designated table.
      On the Wakanda side, the method will be called on an object of the *Entity* type, for example *ds.MyTable(1).MyMethod*.
    - **Current selection**: This option indicates that the 4D method will be executed using the current selection of records of the designated table.
      On the Wakanda side, the method will be called on an object of the *Entity Collection* type, for example *ds.MyTable.all().MyMethod*.

**Warning:** When you change the exposure or scope of a project method on the 4D side, you have to reload the remote model on the Wakanda side for these changes to be taken into account.

## Explorer

When 4D Mobile services are enabled, the tables exposed in 4D Mobile and the project methods attached to them are displayed on the "Methods" page of the 4D Explorer in the 4D Mobile Methods section:



## On 4D Mobile Authentication database method

$1, $2, $3 -> On 4D Mobile Authentication database method -> $0

| Parameter | Type | | Description |
|---|---|---|---|
| $1 | Text | ⇐ | User name |
| $2 | Text | ⇐ | Password |
| $3 | Boolean | ⇐ | True = Digest mode, False = Basic mode |
| $0 | Boolean | ⇒ | True = request accepted, False = request rejected |

### Description

The **On 4D Mobile Authentication database method** provides you with a custom way of controlling the opening of 4D Mobile sessions (via REST) on 4D. This database method is mainly intended for filtering connections when setting up a connection between a Wakanda Server and 4D.

When the request to open a 4D Mobile session comes from Wakanda Server by means of the ***mergeOutsideCatalog()*** method (general case), the connection identifiers are provided in the header of the request. The **On 4D Mobile Authentication database method** is called so that you can evaluate these identifiers. You can use the list of users for the 4D database or you can use your own table of identifiers.

**Important:** When **On 4D Mobile Authentication database method** is defined (i.e. when it contains code), 4D fully delegates control of 4D Mobile requests to it: any setting made using the "Read/Write" menu on the Web/4D Mobile page of the Database Settings is ignored (see the *Design Reference* manual).

The database method receives two parameters (*$1* and *$2*) of the Text type and a Boolean (*$3*), passed by 4D, and returns a Boolean, *$0*. You must declare these parameters as follows:

```
 //On 4D Mobile Authentication database method
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
... // Code for the method
```

*$1* contains the user name and *$2* the password used for the connection.

The password (*$2*) can be received either in clear or hashed form, depending mode used by the request. This mode is indicated by the *$3* parameter to enable you to perform the appropriate processing:

- If the password is sent in clear (Basic mode), *$3* returns **False**.
- If it is sent in hashed form (Digest mode), *$3* returns **True**.

When a 4D Mobile connection request comes from Wakanda Server, the password is always sent in hashed form.

You must check the identifiers of the 4D Mobile connection in the database method. Usually, you check the name and password using a custom user table. If the identifiers are valid, pass **True** in *$0*. The request is then accepted; 4D executes it and returns the result in JSON.
Otherwise, pass **False** in *$0*; in this case, the connection is rejected and the server returns an authentication error to the sender of the request.

If the user is referenced in the list of 4D users of the database, you can check the password directly by means of the following statement:

```
$0:=Validate password($1;$2;$3)
```

The **Validate password** command has been extended to accept a user name as first parameter as well as an optional parameter indicating whether the password is expressed in hashed form.

If you want to use your own list of users external to the 4D database list, you can save their passwords in hashed form using the same algorithm as that used by Wakanda Server when sending the connection request to the **On 4D Mobile Authentication database method** in *$2*. To hash a password using this method, you can write:

```
  $HashedPasswd :=Generate digest($ClearPasswd ;4D digest)
```

The **Generate digest** command accepts 4D digest as a hashing algorithm, corresponding to the method used by 4D for its internal management of passwords.

## Example 1

This example only accepts the "admin" user with the password "123" that does not match a 4D user:

```
  //On 4D Mobile Authentication database method
 C_TEXT($1;$2)
 C_BOOLEAN($0;$3)
  //$1: user
  //$2: password
  //$3: digest mode
 If($1="admin")
    If($3)
       $0:=($2=Generate digest("123";4D digest))
    Else
       $0:=($2="123")
    End if
 Else
    $0:=False
 End if
```

## Example 2

This example of the **On 4D Mobile Authentication database method** checks that the connection request comes from one of the two authorized Wakanda servers, saved in the users of the 4D database:

```
 C_TEXT($1;$2)
 C_BOOLEAN($0)
 ON ERR CALL("4DMOBILE_error")
 If($1="WAK1")|($1="WAK2")
    $0:=Validate password($1;$2;$3)
 Else
    $0:=False
End case
```

# Configuring the Wakanda Application

On the Wakanda Enterprise side, you can connect to a 4D database:

- either using the "Connect to Remote Datastore" dialog box (found in Wakanda Enterprise Studio),
- or by executing a JavaScript method (mergeOutsideCatalog(), openRemoteStore() or addRemoteStore()).

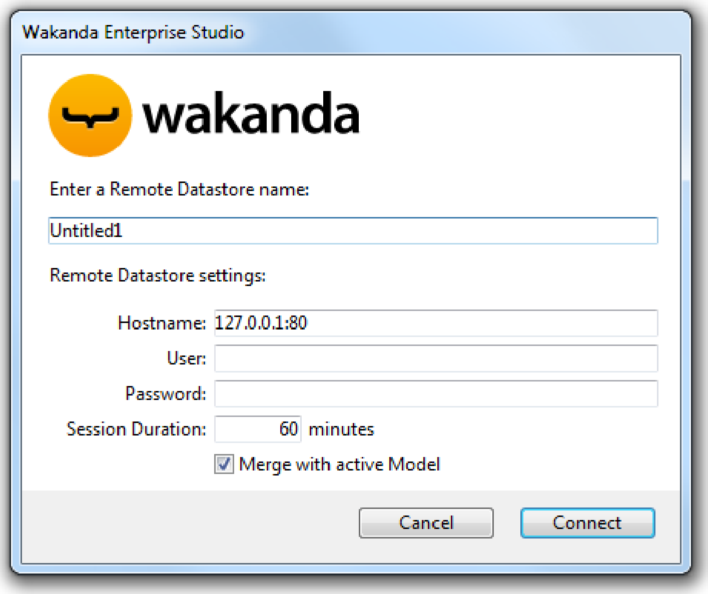
Once a connection is established between Wakanda and 4D, the Wakanda application can use all the exposed tables, attributes and project methods of the 4D application as local objects.

It is also possible to execute additional JavaScript code. For example, you could locally modify properties of remote attributes, or extend classes, or add calculated attributes.

## Using the Connect to Remote Datastore dialog box

In Wakanda Enterprise Studio, the **Connect to Remote Datastore...** command (found in the **File** menu as well as the project's contextual menu) opens a link with a remote *datastore*. This remote *datastore* can be a 4D database or another Wakanda application. In both cases, the HTTP server of the remote *datastore* must be started in order for the Wakanda Enterprise Studio to be able to access the remote model.
Once the link is defined, it is automatically restored each time the application is opened using the connection parameters saved in the ".waRemoteConfig" file (see below).

When you select the **Connect to Remote Datastore...** command, the connection dialog box appears:


Wakanda Enterprise Studio

Enter a Remote Datastore name:

Untitled1

Remote Datastore settings:

Hostname: 127.0.0.1:80

User:

Password:

Session Duration: 60 minutes

[✓] Merge with active Model

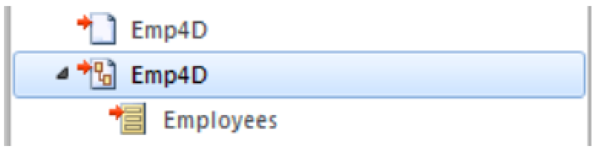
It contains the following connection parameters:

- **Remote datastore name**: Local name of the remote catalog, displayed in the Solution Explorer. If you uncheck the **Merge with active Model** option, this name is used as the datastore id instead of "ds" (see below). In this case, make sure to use compatible characters (see Programming and Writing Conventions in the Wakanda documentation).
- **Hostname**: Address of remote data server (use HTTPS for better security)
- **User** and **Password**: User name and password for opening the 4D Mobile session on the 4D database
- **Session duration**: Number of minutes (60 by default) to maintain the session connected to the remote 4D database. This parameter is only taken into account if the connection is open with a user and password that are not blank (it is strongly recommended to protect 4D Mobile accesses on the 4D side).
- **Merge with active Model** (option checked by default): Merge the remote *datastore* with the project's active

model (**ds** object) so that the remote *datastore classes* are included in the **ds** namespace and more particularly appear in the list of classes of Wakanda's GUI Designer. For more information, refer to **Integrating with the active model or using a dedicated model**.

### Parameter files

When a connection is established between Wakanda and 4D Server by means of the "Connect to a Remote Datastore" dialog box, Wakanda Enterprise Studio automatically creates two files (icons with a red arrow) in the folder of the project:



- the first file (extension ".waRemoteConfig") saves the connection parameters defined in the dialog box,
- the second file (extension ".waRemoteModel") contains the local representation of the model of the remote *datastore*. Its contents can be displayed (but not modified) in the Wakanda model editor window.

**Note:** You can see a file's extension in a help tip that appears when files are selected in the Explorer of Wakanda Studio.

## Using a JavaScript method

Wakanda Enterprise Server lets you establish a link with a 4D database by executing a JavaScript method. The connection method must usually be placed in the code that runs when the application is opened (bootstrap.js), or when the model is opened (model.js) in order for the link to be available during each session.

There are three methods you can use to establish a 4D Mobile link:

- model.mergeOutsideCatalog()
- addRemoteStore()
- openRemoteStore()

The main difference between these methods concerns the way objects coming from the remote *datastore* are integrated in the Wakanda application: **model.mergeOutsideCatalog()** merges the remote catalog with the active model, while **addRemoteStore()** and **openRemoteStore()** generate dedicated models. For more information about this point, refer to **Integrating with the active model or using a dedicated model** below.

### Execution of the mergeOutsideCatalog() method

The **mergeOutsideCatalog()** JavaScript method designates a catalog of remote data and merges it within your current Wakanda model. You must call this method in the .js file associated with the current model and executed by the Wakanda server.

There are two possible syntaxes:

- Direct syntax:

```
model.mergeOutsideCatalog(localName, address, user, password);
```

- Syntax using an object:

```
model.mergeOutsideCatalog(localName, {     hostname: address,     user: userName,     password: password,
    jsFile: jsFilePath     timeout: minutes });
```

The advantage of using the syntax with an object is that you can add a .js file that is executed after connection to the 4D database. This file can locally modify the catalog referenced from the remote database.

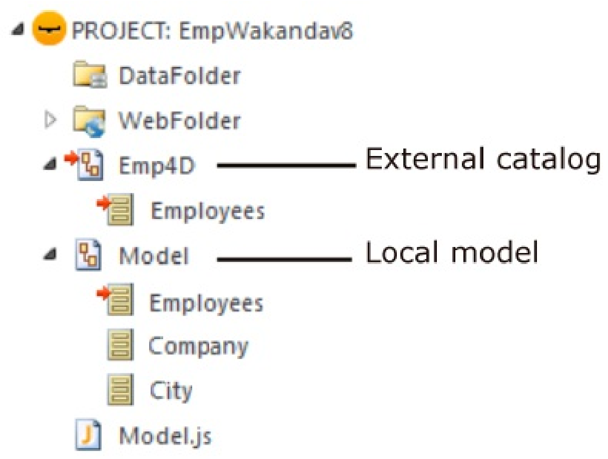
| Parameter | Type | Description |
|---|---|---|
| localName | String | Local name of remote catalog |
| ipAddress | String | Address of remote data server (use HTTPS for added security) |
| userName | String | User name for opening of session |
| password | String | Password for opening of session |
| jsFile | String | (optional) Relative pathname of JavaScript file located in the same folder as the model (see **Modifying the external file**) |
| timeout | Num | (optional) Timeout for client connection to 4D database in minutes (60 by default). Note that this parameter is only taken into account if the connection is open with a user and password that are not blank (it is strongly recommended to protect 4D Mobile accesses on the 4D Server side) |

For a more detailed description, refer to the documentation of the mergeOutsideCatalog() method in the Wakanda Server-Side API manual.

### *model*

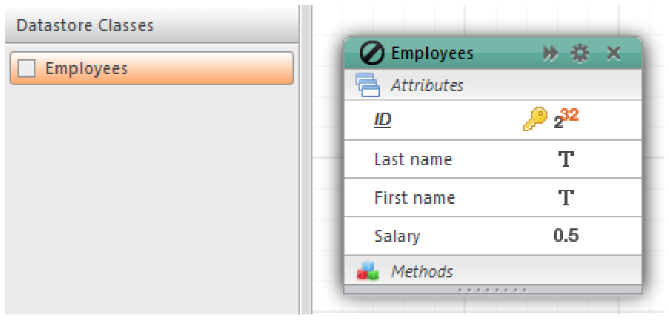
The *model* object indicates the current "model" of the Wakanda application, in other words, the set of its "datastore classes" (tables) and methods. In the context of a 4D Mobile architecture, the Wakanda model can be empty. If the Wakanda application already contains objects, the classes and methods referenced from the remote 4D application are merged with the local model when you use the **mergeOutsideCatalog()** method.

When the connection is established successfully, the "exposed" 4D tables are added to the classes of the model on the Wakanda side. In Wakanda Enterprise Studio, you can see the remote tables among the list of classes for the local model. External elements are indicated by a red arrow. The external catalog is also represented in Wakanda Studio by a specific catalog (named *localName*.waRemoteCatalog) that is also indicated by a red arrow:



**Note:** File extensions can be hidden in Wakanda Studio.

You can double-click on this file to view the external catalog in the model editor of Wakanda Enterprise Studio:



### *Example*

- Example of direct connection:

```
model.mergeOutsideCatalog("base4D","localhost:80", "admin", "123456");
```

- Example of connection using an object:

```
model.mergeOutsideCatalog("base4D", {        hostname: "http://localhost:8050",     user: "wak",
 password: "123456",      jsFile: "Model2.js"     timeout: 15 });
```

### openRemoteStore() and addRemoteStore()

The **openRemoteStore()** and **addRemoteStore()** methods are alternative ways of establishing dynamic links between a Wakanda application and a 4D application.

Like **mergeOutsideCatalog()**, these methods provide dynamic access to the data of 4D databases but they work in a different way:

- they can reference a remote model at any time during the Wakanda session -- and not just when the solution is loaded.
- tables, attributes and methods of the external model can be accessed by means of a separate datastore; they are not merged with the local model of the Wakanda application (accessed by means of the **ds** object).

openRemoteStore() only returns a valid reference in the current JavaScript context, whereas addRemoteStore() maintains the reference throughout the session.

For more information, refer to the description of the **openRemoteStore()** and **addRemoteStore()** methods in the Wakanda documentation.

## Integrating with the active model or using a dedicated model

Whichever way you connect with the remote 4D *datastore* (using the "Connect to Remote Datastore" dialog box of Wakanda Studio or executing a JavaScript method), you have to choose whether the remote classes (tables) must be merged with the active model, or placed in a dedicated model.

This choice is summarized in the following table:

| For... | merging with active model | using a dedicated model |
|---|---|---|
| "Connect to Remote Datastore" dialog box | Check *Merge with active Model* | Uncheck *Merge with active Model* |
| JavaScript method | mergeOutsideCatalog() | openRemoteStore() or addRemoteStore() |

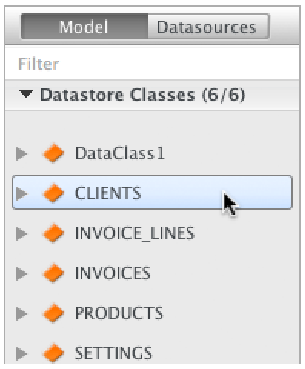
### Merged with active model

When you merge remote 4D tables with the active model, they are integrated into the default model of the application (whose datastore is the **ds** object), as local classes. The data access principles are:

- on the server side, you access remote 4D tables and methods by means of the **ds** object (see **Calling 4D Tables and Methods**). Example:

```
var invoiceList = ds.INVOICES.all(); //access to the INVOICES table of the catalog by default
```

- on the client side, you have the automatic features of the Wakanda Ajax Framework (WAF) library: remote 4D tables are available by means of high-level *datasource* objects, or using the *dataprovider* API, which provides lower-level access.
- in Wakanda Enterprise Studio, the tables of the 4D database are listed with the local classes in Wakanda's GUI Designer:



These principles facilitate the development of 4D Mobile applications but may lead to naming conflicts between tables, in particular when the Web application calls on several remote *datastores*. In this case, it may be useful to place the remote elements in a dedicated model.

### Using a dedicated model

When remote 4D tables are not merged with the active model, they use a "dedicated" model. Remote classes then use a namespace that is specific to the *datastore* to which the application is connected and they cannot be accessed in the **ds** object. This way it is possible to use several tables with the same name in several different datastores:

- on the server side, you access remote 4D tables and methods by means of a custom catalog whose name is the one that you passed in the **Remote datastore name** connection parameter (dialog box), or in *localName* (JavaScript method). For example, if you created a link called "my4Dstore", in the code of the application you can write:

  ```
  var invoiceList = my4Dstore.INVOICES.all(); //access to the INVOICES table of the my4Dstore datastore
  ```

However, this principle has certain limitations in the current version of Wakanda Enterprise:

- It is not possible for client applications to access remote classes directly by means of the WAF library or using REST,
- Remote classes are not listed in the *GUI Designer* of Wakanda Enterprise Studio.

So it is usually recommended to choose the merged mode for remote *datastores* when your client application must access the data of the 4D remote tables directly.

## Modifying the external file

Wakanda Enterprise lets you modify certain characteristics of the local version of the external model, for the purpose of customizing, security or optimization.

To do this, you just need to add the appropriate JavaScript code in a .js file that has the same local name as the catalog plus the suffix ".js" and put this file in the same folder as the model. For example, if the name of the local catalog is *Emp4D.waRemoteMode*l, you need to use a file named *Emp4D.js* placed in the folder of the model.

**Notes:**

- Starting with version 11, by default this file is created automatically by Wakanda Studio.
- When you establish the connection using a JavaScript method, it is possible to use another name by means of the *jsFile* parameter.

Wakanda executes this file when the external catalog is initialized. Using this file, you can:

- modify properties of datastore class attributes, such as events or the scope. Example:

  ```
  model.className.attributeName.scope ="publicOnServer"
  ```

- add calculated attributes to datastore classes. Example:

  ```
  model.className.calcAtt = new Attribute("calculated", "string"); model.className.calcAtt.onGet =
  function(); model.className.calcAtt.onSet = function();
  ```

- add alias attributes to datastore classes. Example:

  ```
  model.className.newAlias = new Attribute("alias", "number", "Link_15.cinteger");
  ```

- create local datastore classes derived from tables of the external catalog, to fully control data sent to clients. A derived datastore class can present a custom view of an external table, while maintaining overall access to the extended (parent) datastore class on the Wakanda server. Example:

  ```
  model.DerivedClass = new DataClass("Emps", "public", "My4DTable")
  ```

- remove attributes from derived local datastore classes, by security or to optimize network traffic. Example:

  ```
  model.DerivedClass = new DataClass("Emps", "public", "My4DTable")
  model.DerivedClass.removeAttribute("salary"); model.DerivedClass.removeAttribute("comments");
  model.DerivedClass.removeAttribute("...");
  ```

  With this example, you create a derived class named "DerivedClass", based on the "My4DTable" class, which sends only the attributes that you want using the network.

For more information about JavaScript code for working with models, refer to the Model API section in the Wakanda documentation.

## Defining permissions

You can set specific permissions to Wakanda Server overall for the remote model and/or individually for each class. For more information about this point, refer to the Assigning Group Permissions section of the Wakanda documentation.

# Calling 4D Tables and Methods

## Calling 4D tables

The access mode for 4D tables referenced in the Wakanda application depends on how the external catalog is integrated, which is defined in Wakanda during its connection to the remote application (see **Integrating with the active model or using a dedicated model**):

- merged with active model (default option): in this case, remote tables are used exactly like local classes through the **ds** object.
- using a dedicated model: in this case, remote tables are properties of the dedicated model object.

### Tables merged with active model

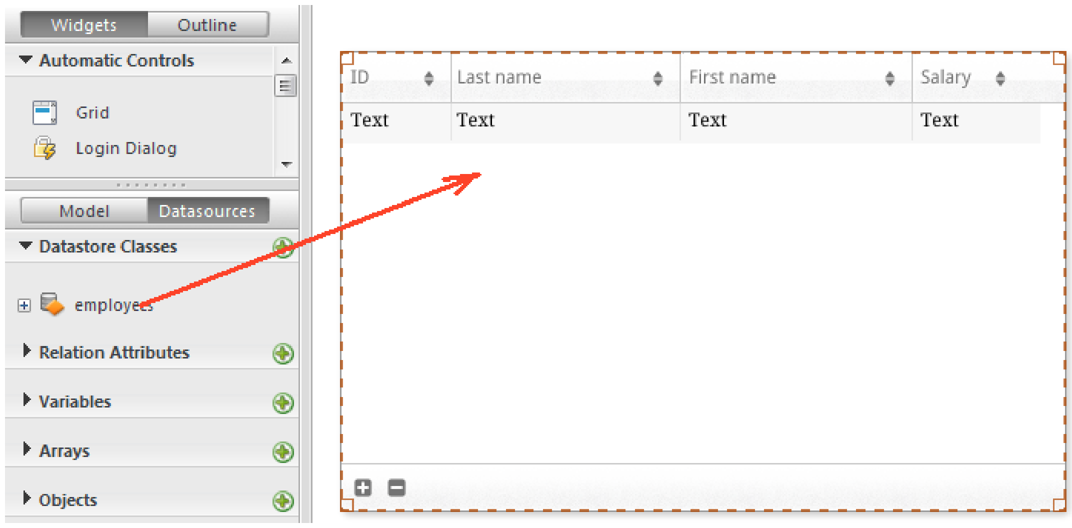
When merged with the active model, 4D tables referenced in the Wakanda application can be used directly in the Server-side JavaScript code as properties of the **ds** object, just like local *datastore classes*.

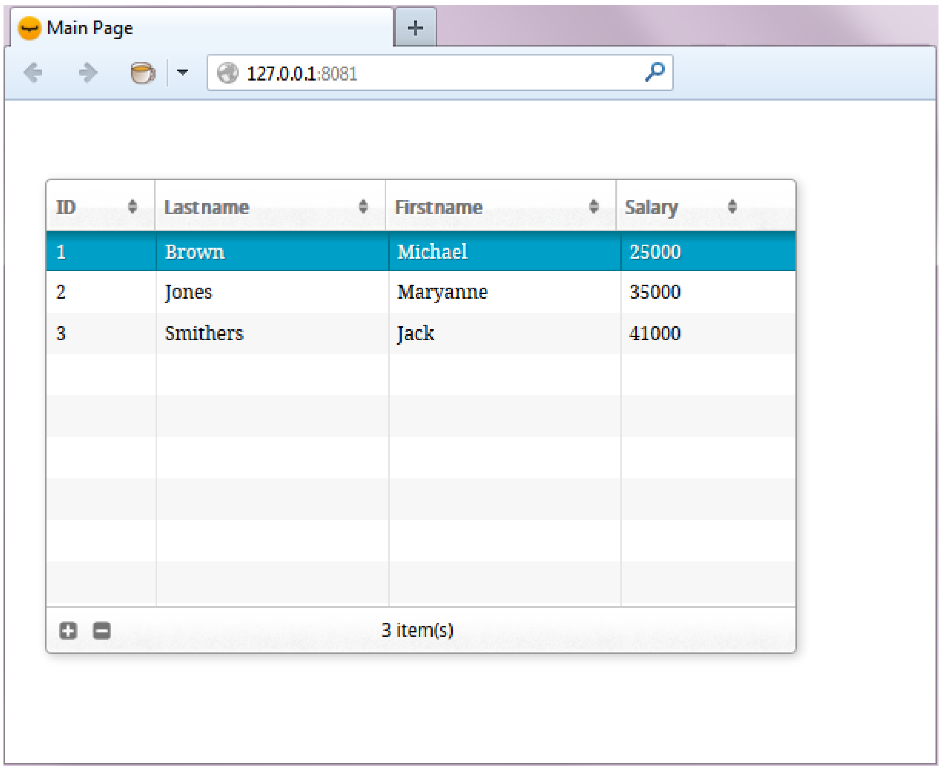
**Note:** The ds object contains the current *datastore* of the Wakanda application.

For example, to perform a query in the records of the [Employees] 4D table, you can write:

```
var emp = ds.Employees.query("age > :1",30);         //retrieve a collection of records from the Employees table
        //where age is greater than 30 in the emp variable
```

On the client side, you can also take advantage of automatic mechanisms of the *datasources* based on the *datastore classes* and associated with the widgets. For example, if you associate the 'employees' datasource with a 'Grid' type widget, you can display the list of employees automatically:





| ID | Lastname | Firstname | Salary |
|---|---|---|---|
| 1 | Brown | Michael | 25000 |
| 2 | Jones | Maryanne | 35000 |
| 3 | Smithers | Jack | 41000 |

3 item(s)

When the table is associated with a *datasource*, you can also access its data using this *datasource*. For example, to sort the collection of records of the 'employees' *datasource*, you can write:

```
sources.employees.orderBy("age");     //sorts the collection of employees by their age
```

For more information about working with *datastore classes*, refer to the Wakanda documentation.

### Tables placed in a dedicated model

The 4D tables referenced are used in the Server-side JavaScript code as properties of the catalog where they were placed when the link was created. The name of this catalog is the one that you passed in the **Remote datastore name** connection parameter (Wakanda Studio connection dialog box) or in *localName* (JavaScript methods).

For example, if you created a link called "my4Dstore" and want to perform a query among the records of the [Employees] table, you can write:

```
var emp2 = my4Dstore.Employees.query("age > :1", 30);     // search in the records of the Employees table     //
in the link named "my4Dstore"
```

***Implementation note:*** *On the client side, using a dedicated model in the current version of 4D Mobile does not allow access to remote classes at the present time.*

## Calling 4D methods

### Scope and objects

4D methods referenced in the Wakanda application can be used directly in the JavaScript code as properties of the **datastore class**, **entity collection** or **entity** objects, depending on their scope as defined on the 4D side (see the **Parent table and Scope of project methods** paragraph). Here is the correspondence between Wakanda objects and the scope of the project methods:

| 4D scope | Wakanda object |
|---|---|
| table | datastore class |
| current selection | entity collection |
| current record | entity |

**Note:** 4D methods can also be called on the client side using *datasources* (see below); in this case all the methods are available, and the datasource applies them automatically to the current collection or the current entity depending on the context.

For example, if you perform a query using the query method (see previous section), Wakanda returns an entity collection. You can execute any 4D project method whose scope is declared as "current selection" on this collection.

## Server or Client API

There are three ways for 4D methods to be called by JavaScript code:

- From the JavaScript code executed on the server (SSJS), using the SSJS Datastore API. In this case, 4D methods are called as properties of the **datastore class**, **entity collection** or **entity objects**, as described above.
  Examples:

  ```
  var vTot = ds.Emp.raiseSalary(param))     //raiseSalary is a datastore class property     //the catalog is
  merged with the active model var vTot2 = my4DStore.Company.first().capital(param))     //capital is an
  entity property since first() returns an entity     //using the my4DStore dedicated model
  ```

- From the JavaScript code executed on the client (i.e., the browser) using the Wakanda Ajax Framework (WAF). There are two possibilities:
  ***Implementation note:*** *In the current version of Wakanda Enterprise, client access to 4D database methods is only avaialble when the remote database is connected and merged with the active model.*
  - using WAF Datasource API: this high-level API provides many automatic functions for managing data. With this API, 4D methods are called as **properties of datasources associated with datastore classes** and will be applied automatically to the datastore class, the current entity collection, or the current entity depending on the context. You can manage return values of the methods or any errors using the asynchronous syntax (required for code executed on the client). Example:

    ```
    sources.employee.raiseSalary(param,     {onSuccess: function(event)     { ... //code to execute when
    method has finished} }))
    ```

    It is not mandatory to use a callback function because datasource objects have automatic functions that support, for example, updating data displayed in the current collection after a query.
  - using WAF Dataprovider API: this low-level client API lets you work with objects directly. As with the SSJS Datastore API, 4D methods are called as properties of the **datastore class**, **entity collection** or **entity objects**. However, you must manage the return values of the methods or any errors using the asynchronous syntax (required for code executed on the client). Example:

    ```
    ds.Employee.raiseSalary(param, // syntax resembles a SSJS call     {onSuccess: function(event)    //
    but it’s client-side code so you must     // manage the callback method of the asynchronous call     {
    ... //code to execute when 4D method has finished} }))
    ```

  The choice of location (server or client) and API depends on the needs of the application and is described in the Wakanda documentation.

## Parameters

As with standard methods, you can pass parameters during the call, which are received in order in the parameters $1, $2, and so on. Similarly, the method can return a result in the $0 variable.

Example: You want to give a 5% raise to employees whose salary is less than 1500.

- On the 4D side, the ***IncreaseSalary*** method was exposed through 4D Mobile and its scope is the "Current selection". Its code is as follows:

  ```
  C_REAL($1)
  READ WRITE([Employees])
  FIRST RECORD([Employees])
  While(Not(End selection([Employees])))
     [Employees]salary:=[Employees]salary*$1
     SAVE RECORD([Employees])
     NEXT RECORD([Employees])
  End while
  UNLOAD RECORD([Employees])
  ```

- On the Wakanda side, you execute the following code on the server:

  ```
  var emp = ds.Employees.query("salary < :1",1500);         // emp contains the collection of employees whose
  salary is <1500     emp.IncreaseSalary(1.05);         //execute the IncreaseSalary method on the collection
          //You could also write:         //"ds.Employees.query("salary < :1",1500).IncreaseSalary(1.05);
  ```

You can also return a 4D selection directly as a Wakanda collection using the **MOBILE Return selection** command.

For example:

```
 //FindCountries project method
 //FindCountries( string ) -> object

C_TEXT($1)
C_OBJECT($0)
QUERY([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Return selection([Countries])
```

### Update of 4D context

When calling a 4D method by means of the Wakanda link:

- If the method applies to a selection (*entity collection*), it becomes the current selection and 4D is positioned on the first record of this selection without loading it or activating the links. If the selection is empty, the **Selected record number** command returns 0 instead of 1.
- If the method applies to a record (*entity*), it becomes the current record. The current selection is reduced to just this record and the **Selected record number** command returns 1.
  **Note:** For optimization reasons and to avoid unnecessary locking, the record is loaded in read only mode. However, the table is in read-write mode, so you can simply call the **LOAD RECORD** command to force the record to be loaded in read-write mode whenever necessary.
- If the method applies to a table (*datastore class*), neither the current selection nor the current record are affected.

Note that after executing a method through 4D Mobile, the 4D context is reset:

- selections are reduced to 0,
- records are unstacked and unloaded,
- local selections and sets for the process are destroyed,
- any transactions open during method execution are canceled,
- the configuration of automatic relations by fields, query destinations or queries on the server are reset,
- print jobs are canceled,
- windows are closed,
- any SQL, PHP or HTTP connections are closed.

## Scope error

You must make sure the scope of the 4D method corresponds to the type of Wakanda object that is calling it, otherwise a "*TypeError: 'undefined' is not a function*" error is returned by Wakanda.

For example, given the 4D "getcursel" method containing the following code:

```
$0:=Records in selection([Table_1])
```

Given the run method on the Wakanda side:

```
var tt = ds.Table_1.query("Field_2 = 'a*'").getcursel();
```

The **query( )** method returns a collection. If the scope of the *getcursel* method was set as "Current record", Wakanda returns the following error:
*TypeError: 'undefined' is not a function (evaluating 'ds.Table_1.query("Field_2 = 'a*'").getcursel()')".*

## MOBILE Return selection

MOBILE Return selection ( aTable ) -> Function result

| Parameter | Type | | Description |
|---|---|---|---|
| aTable | Table | → | Table whose current selection you want to return |
| Function result | Object | ↩ | Wakanda-compliant selection |

### Description

The **MOBILE Return selection** command returns a JSON *object* that contains the current selection of *aTable* transformed into a Wakanda-compliant entity collection.

This command is intended to be called in the context of a 4D Mobile connection, usually between your 4D application and a Wakanda application (via REST). When a 4D Mobile connection is established and appropriate access rights have been configured, a Wakanda application can execute a 4D project method that returns a value in the *$0* parameter.

The **MOBILE Return selection** command allows you to return, in *$0*, the current selection of records of the *aTable* table, in the form of an *entity collection* object in JSON format. This object is compliant with Wakanda entity collections that contain a selection of records (i.e. of *entities*).

Keep in mind that 4D Mobile accesses require specific configurations in your 4D database:

- The Web server must be launched,
- The "Activate 4D Mobile Service" option must be checked in your Database settings,
- You must have a valid license,
- Tables and fields used must have the "Expose for 4D Mobile" option checked (set by default).
- Called methods must have the "Available through 4D Mobile call" option checked (not set by default).

Note that you can pass any valid table of the database in *aTable*, and not necessarily the table with which the project method has been associated in its properties. This parameter is only used on the Wakanda side to define the objects for which the method can be called.

For more information on 4D Mobile configuration, please refer to the **4D Mobile** documentation.

### Example

You want to display the current selection of the [Countries] table in a Wakanda grid, based on a query.

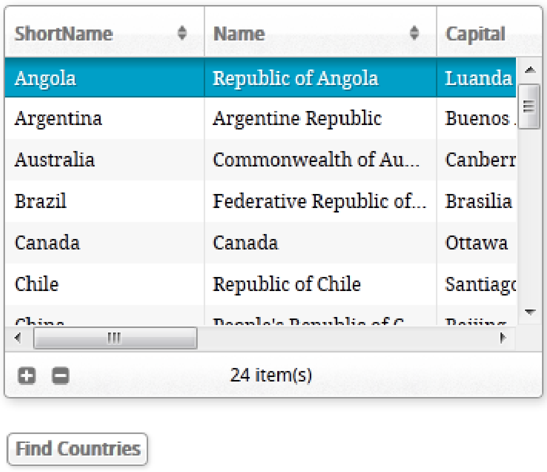
You write the following 4D method:

```
  //FindCountries project method
  //FindCountries( string ) -> object

C_TEXT($1)
C_OBJECT($0)
QUERY([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Return selection([Countries])
```

The returned selection can be used directly in Wakanda as a valid collection.

In the Wakanda server model connected to 4D via 4D Mobile, you have created a page with a grid bound to the 4D Countries table. By default, at runtime, all entities from the 4D table are displayed:



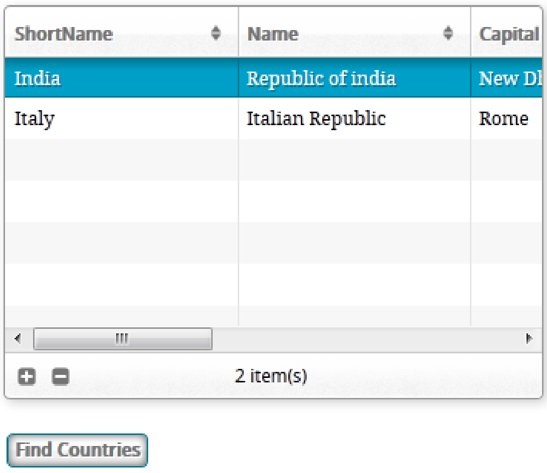
| ShortName | Name | Capital |
|---|---|---|
| Angola | Republic of Angola | Luanda |
| Argentina | Argentine Republic | Buenos . |
| Australia | Commonwealth of Au... | Canberr |
| Brazil | Federative Republic of... | Brasilia |
| Canada | Canada | Ottawa |
| Chile | Republic of Chile | Santiagc |
| China | People's Republic of C | Beijing |

24 item(s)

The code of the button is:

```
button1.click = function button1_click (event)     {          sources.countries.FindCountries("i", {  //we call
the 4D method, "i" is passed as $1                 onSuccess:function(coll){  //callback function
(asynchronous), receives $0 as parameter                      sources.countries.setEntityCollection(coll.result);
//replace the current entity collection                                   // with the one in the coll.result
object                  }             });            };
```

As a result, the grid is updated:



| ShortName | Name | Capital |
|---|---|---|
| India | Republic of india | New Dl |
| Italy | Italian Republic | Rome |

2 item(s)

# Using Relations

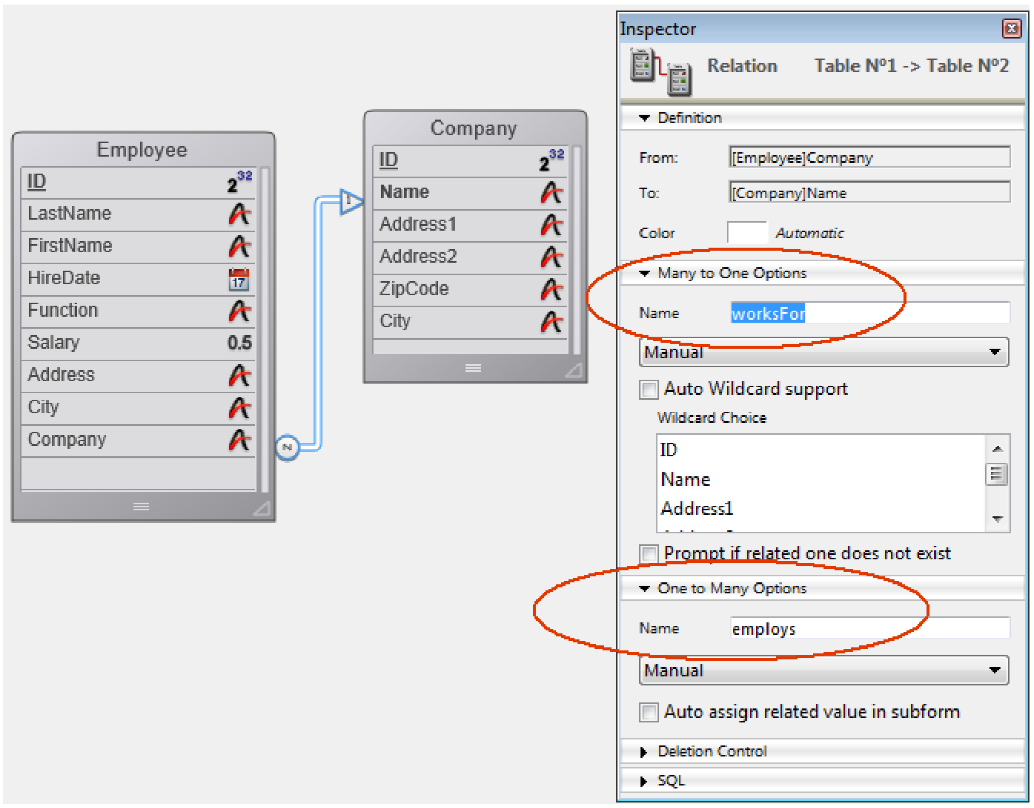
Relations established between 4D tables are used tranparently in the context of a 4D Mobile link. However, the way these relations are represented differs in Wakanda at the model level. In the model editor, relations are linked with specific attributes, called relational attributes. These attributes can be used to display the linked data directly or to perform queries. For more information about this, refer to the "Attributes" section in the Wakanda documentation.

For each relation established on the 4D side, two relational attributes are added in the model representation on the Wakanda side:

- an n->1 attribute in the source table (class) of the relation
- a 1->n attribute in the destination table (class) of the relation.

Both these attributes are given the name of the relation as it was defined respectively for the many-to-one and one-to-many relation in the Inspector on the 4D side.

For example, let's imagine that in the context of a "Employee/Company" structure, you create a relation from the [Employee] table to the [Company] table. You can characterize this relation by means of the name you give it: for example, you could name the many-to-one relation "*worksFor*" and the one-to-many relation could be "*employs*":

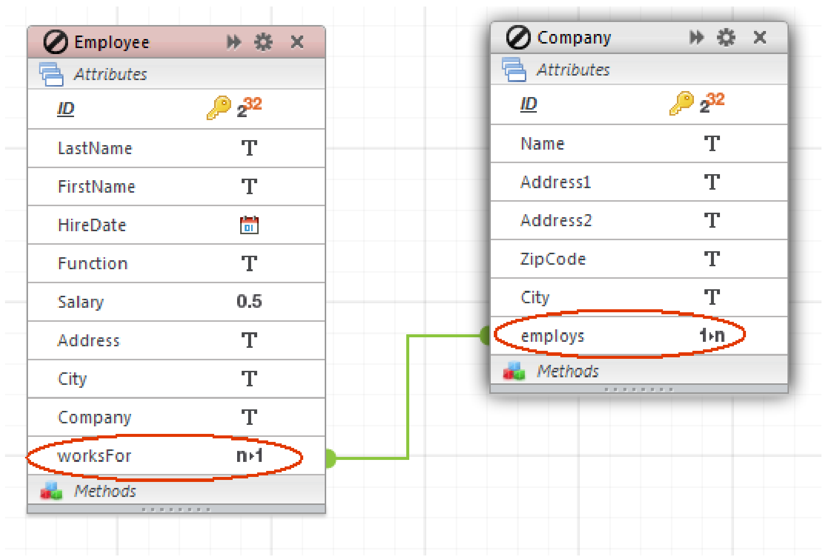


From: [Employee]Company
To: [Company]Name
Color Automatic

Many to One Options
Name worksFor
Manual
Auto Wildcard support
Wildcard Choice: ID, Name, Address1
Prompt if related one does not exist

One to Many Options
Name employs
Manual
Auto assign related value in subform

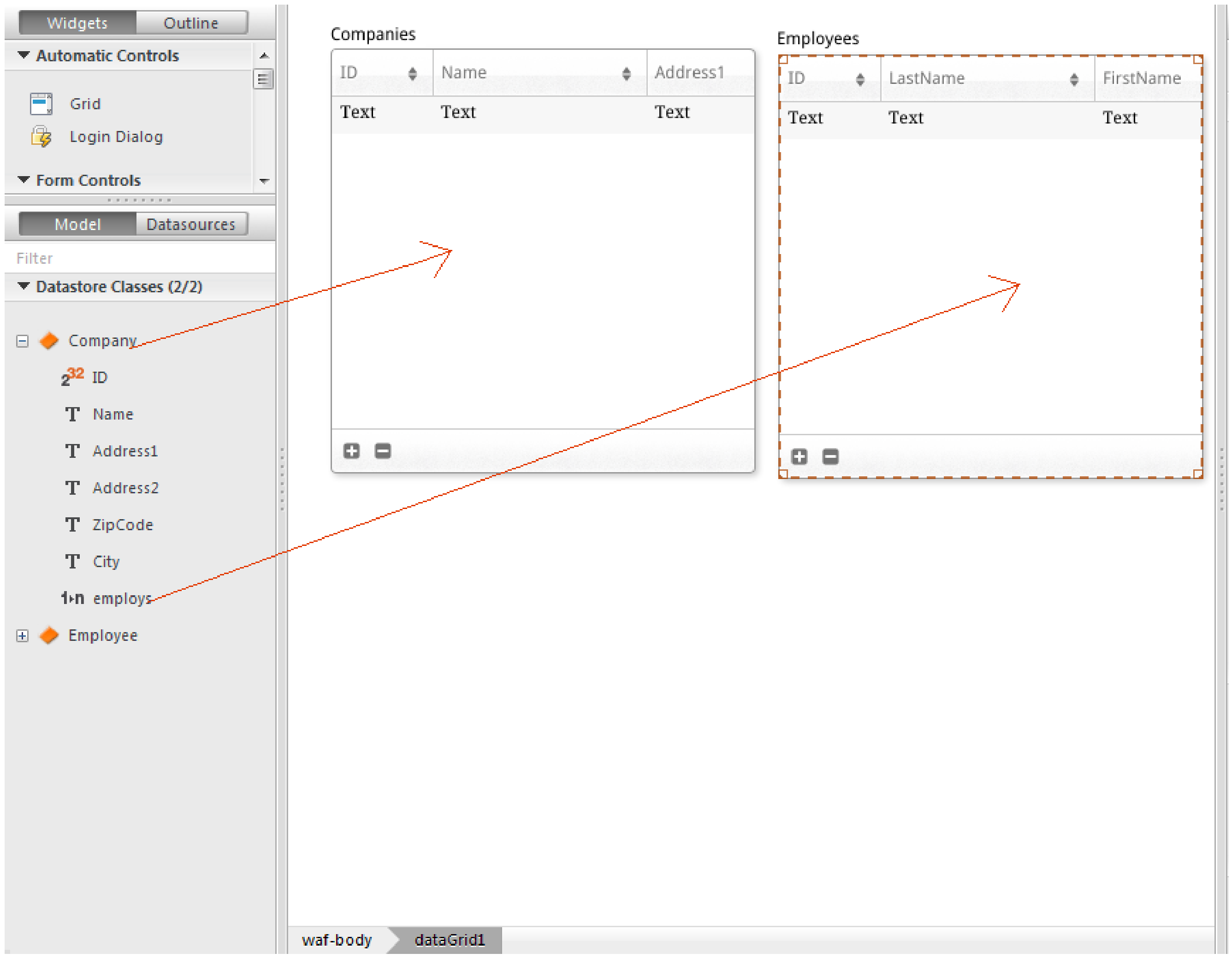
On the Wakanda side, through a link via the connector, these relations are automatically materialized by two additional relational attributes, that you can see in the Model editor:



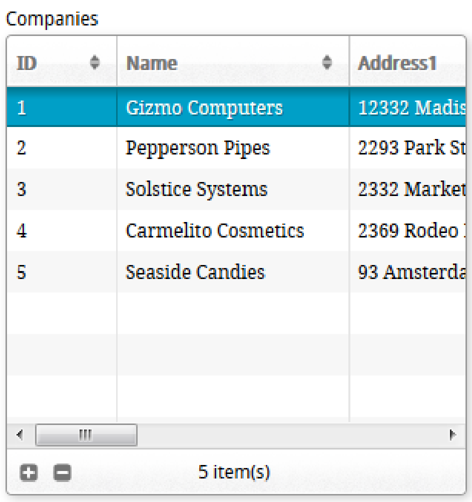
You can give these relations (and thus their corresponding relational attributes) any name you want, based on the logic of your application.

The advantage of this is that it is very simple to use these attributes on the Wakanda side to work with the related data. More particularly, you can create widgets associated with *datasources* that are based on relational attributes. These widgets are then managed and updated automatically according to user actions.

For instance, it is very simple to create a page containing one grid with the list of companies, and another listing the employees of the company selected. To do this, you just associate the "Company" datastore class with one grid, and the "employs" relational attribute with the other:

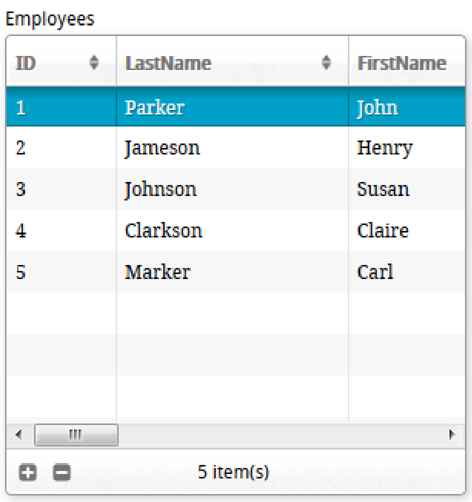


The corresponding datasources are automatically created, and during execution, both grids are synchronized automatically:


Companies

| ID | Name | Address1 |
|---|---|---|
| 1 | Gizmo Computers | 12332 Madis |
| 2 | Pepperson Pipes | 2293 Park St |
| 3 | Solstice Systems | 2332 Market |
| 4 | Carmelito Cosmetics | 2369 Rodeo I |
| 5 | Seaside Candies | 93 Amsterda |

5 item(s)


Employees

| ID | LastName | FirstName |
|---|---|---|
| 1 | Parker | John |
| 2 | Jameson | Henry |
| 3 | Johnson | Susan |
| 4 | Clarkson | Claire |
| 5 | Marker | Carl |

5 item(s)

# Managing 4D Mobile sessions

## Overview

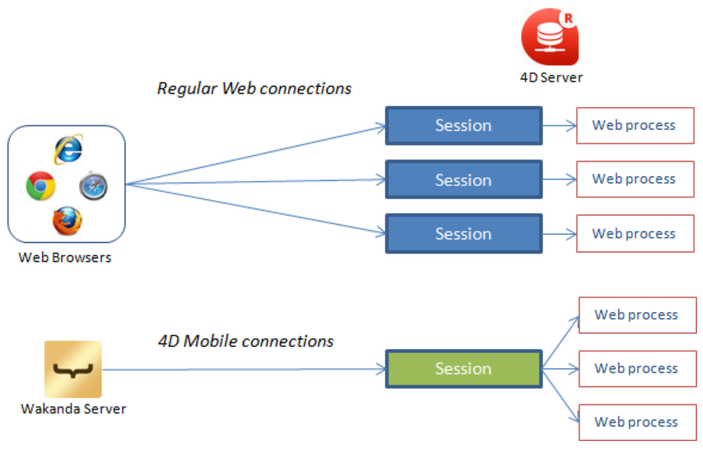
Starting with 4D v15 R4, it is possible to access the ID of a 4D Mobile session on the 4D Server by programming. This feature allows developers to get or set session-related information locally (see example below).

4D Mobile sessions are handled using regular 4D Web session commands. Several 4D Web commands, as well as the **WEB Get session process count** command and the **On Web Close Process database method** support 4D Mobile sessions.

## 4D Mobile sessions vs Web sessions

4D Mobile sessions and Web sessions are two different kinds of sessions. Although they share some concepts (and commands), they do not have the same properties. The main difference is the relation between a session, a process and the process context:

- A Web session is linked to a single Web process; thanks to the Automatic Session Management feature, the process context (instances of variables, selections, etc.) of the session can be reused.
- A 4D Mobile session can be linked to several Web processes; each process context is automatically reset at the end of the execution of the process method.



As a result, the sharing of information related to the session between 4D Mobile Web processes requires specific implementations on 4D Server.

## Supported commands with no change

The following existing Web session handling commands support 4D Mobile sessions.

### WEB CLOSE SESSION(sessionID)

The **WEB CLOSE SESSION** command closes the 4D Mobile session whose ID is passed in *sessionID*. Since a 4D Mobile session can handle several processes, this command actually requests all related Web processes to terminate their execution.

### WEB Get Current Session ID -> sessionID

The **WEB Get Current Session ID** command returns the UUID associated with the current 4D Mobile session.

### WEB GET SESSION EXPIRATION ( sessionID ; expDate ; expTime )

The **WEB GET SESSION EXPIRATION** command returns expiration information related to the cookie of a 4D Mobile session.

The same cookie is used for all processes attached to a 4D Mobile session.

### New WEB Get session process count command

The **WEB Get session process count** command allows you to find out the number of existing processes related to a given session.

- For regular Web sessions, the command always return 1 (one Web session = one process),
- For 4D Mobile sessions, the command returns all related Web processes. The command is useful in this context for example to execute a loop on all processes of a 4D Mobile session.

## On Web Close Process database method (formerly On Web Session Suspend)

The **On Web Close Process database method** is called by 4D each time a Web process is about to finish its execution. It fully supports 4D Mobile session processes: in this context, it is called for each Web process closed, allowing you to save any data (variable, selection, etc.) generated by the 4D Mobile session process.

**Note:** For regular Web sessions, the **On Web Close Process database method** database method is called each time the Web session, i.e. the Web session unique process, is closed.

## Example

If you want to share or reuse information between several processes of a single 4D Mobile session, you can use the UUID of the 4D Mobile session to identify session-related data. For example, after a query of records, you want to keep a named selection on the 4D Server so that any subsequent REST request in the same session will be able to access this selection directly. You can write, after the query statement:

```
 //create an interprocess selection including session UUID
COPY NAMED SELECTION([Emp];"<>EmpSel"+WEB Get Current Session ID)

 //later on, you can reuse this selection from the same session
USE NAMED SELECTION([Emp];"<>EmpSel"+WEB Get Current Session ID)
```

# WEB Get session process count

WEB Get session process count ( sessionID ) -> Function result

| Parameter | Type | | Description |
|---|---|---|---|
| sessionID | Text | → | Session UUID |
| Function result | Longint | ↩ | Number of processes attached to the session |

## Description

The **WEB Get session process count** command returns the number of running processes attached to the session whose UUID you passed in *sessionID*.

This command is added in the context of the **Handling of 4D Mobile sessions by programming** feature introduced in 4D v15 R4. It is mainly designed to count the number of processes run by a 4D Mobile session.

- For a 4D Mobile session, this command returns the actual number of processes. A 4D Mobile session can run several processes.
- For a regular Web session, this command always return 1 (one Web session = one process).

## Example

You want to store information on the current 4D Mobile session in arrays:

```
C_TEXT($sessionID)
C_LONGINT($count)
C_DATE($expDate)
C_TIME($expTime)

$sessionID:=WEB Get Current Session ID
$count:=WEB Get session process count($sessionID)
WEB GET SESSION EXPIRATION($sessionID;$expDate;$expTime)

APPEND TO ARRAY($aTimestamp;String(Current date)+" "+String(Current time))
APPEND TO ARRAY($aSessionUID;$sessionID)
APPEND TO ARRAY($aNbProcesses;$count)
APPEND TO ARRAY($aExpirationDate;$expDate)
APPEND TO ARRAY($aExpirationTime;$expTime)
```

## WEB CLOSE SESSION

WEB CLOSE SESSION ( sessionID )

| Parameter | Type | | Description |
|---|---|---|---|
| sessionID | Text | → | Session UUID |

### Description

The **WEB CLOSE SESSION** command invalidates an existing session designated by the *sessionID* parameter. If the session does not exist, the command does nothing.

When this command is called from a Web process or any other process:

- The cookie expiration date is set to 0,
- The **On Web Close Process database method** is called, allowing you to store session information,
- Selections are erased, records are unlocked and variables are reset.

After this command is executed, if a Web client sends a request using an invalid cookie, a new session is opened and a new cookie is sent.

**Note:** In the context of a 4D Mobile session, the **WEB CLOSE SESSION** command closes the 4D Mobile session whose ID is passed in the *sessionID* parameter. Since a 4D Mobile session can manage several processes, this command actually requests all the Web processes related to the session to finish their execution.

## WEB Get Current Session ID

WEB Get Current Session ID -> Function result

| Parameter | Type | | Description |
|---|---|---|---|
| Function result | Text | ↩ | Session UUID |

### Description

The **WEB Get Current Session ID** command returns the ID of the session open for the Web request. This ID is generated automatically by 4D as an UUID.

If this command is called outside of the context of a Web session, it returns an empty string "".

# WEB GET SESSION EXPIRATION

WEB GET SESSION EXPIRATION ( sessionID ; expDate ; expTime )

| Parameter | Type | | Description |
|---|---|---|---|
| sessionID | Text | ⇒ | Session UUID |
| expDate | Date | ⇐ | Date of cookie expiration |
| expTime | Time | ⇐ | Time of cookie expiration |

## Description

The **WEB GET SESSION EXPIRATION** command returns the expiration information for the cookie of the session whose UUID you passed in *sessionID*.

The *expDate* parameter receives the expiration date and the *expTime* parameter receives the expiration time of the cookie.

**Note:** Each time a web request is sent, the expiration date and time of the cookie are reset to a value corresponding to the time of the request+the value of the Web Inactive session timeout. For example:
*First request, Monday at 1:00*
-> Sends a cookie 4DSID xxxyyy expiration I+24h = Tuesday 01:00
*Second request, Monday at 1:10*
-> Sends a cookie 4DSID xxxyyy expiration I+24h = Tuesday 01:10
*Third request, Tuesday at 4:00: cookie expired*
-> Sends a cookie 4DSID aaabbb expiration I+24h = Wednesday 01:00

## On Web Close Process database method

On Web Close Process database method
Does not require any parameters

---

The **On Web Close Process database method** is called by the 4D Web server each time a Web session is about to be closed. A session can be closed in the following cases:

- when the maximum number of simultaneous sessions is reached (100 by default, modifiable using the **WEB SET OPTION** command), and 4D needs to create new ones (4D automatically kills the process of the oldest inactive session),
- when the maximum period of inactivity for the session process is reached (480 minutes by default, modifiable using the **WEB SET OPTION** command),
- when the **WEB CLOSE SESSION** command is called.

When this database method is called, the context of the session (variables and selections generated by the user) is still valid. This means that you can save data related to the session in order to be able to use them again subsequently, more specifically using the **On Web Connection Database Method**.

**Note:** In the context of a 4D Mobile session (which can generate several processes), the **On Web Close Process database method** is called for each Web process that is closed, allowing you to save all types of data (variables, selection, etc.) generated by the 4D Mobile session process.

An example of the **On Web Close Process database method** is provided in the **Web Sessions Management** section.

# About 4D Mobile application security

Once data from 4D database tables exposed through 4D Mobile is integrated into the Wakanda catalog, you need to restrict access to certain "sensitive" resources.

Unlike 4D applications, with Web applications you cannot use the interface to control the data exposed: for example, just because a field is not displayed in the form does not mean that it is inaccessible to the user. HTTP requests and the use of JavaScript can allow malicious users to potentially obtain any information from an insufficiently protected Web server.

The purpose of this section is not to list all the security measures to take with 4D Mobile applications but to provide you with leads to help secure the data exposed in a minimal way.

- **Protection of 4D Mobile accesses to the 4D database**: You must control 4D Mobile (via REST) connection requests. You can use either:
  - 4D passwords (see **Automatic controls by 4D password** paragraph),
  - the **On 4D Mobile Authentication database method**.
- **Control 4D Mobile exposure on the 4D side**: Each table, attribute and method can be exposed (or not) through 4D Mobile. Only expose the data and methods that are strictly necessary; for instance, there is no need to expose any unused fields.
- **Protection of exposed data**: You must use the security systems provided by Wakanda to control the contents that are accessible by means of browsers. There are several ways to do so (not exclusive):
  - **Adjust the scope** of the 4D database attributes and methods in Wakanda at the model level (refer to the **scope** property for attributes or for methods in the Wakanda documentation). In particular, you can set their scope as **Public on Server**, which means that they can be access freely for code run on the server, but they will not be accessible on Web clients. This setting is done in the .js configuration file of the external model (see **Modifying the external file**).
  - **Use calculated attributes**: calculated attributes work like standard attributes but their values are returned by specific functions **(onGet**, **onSet**...) that are executed when accessing the fields. This means that you can expose only the necessary calculated attributes without exposing the fields of the 4D database. Access to the 4D fields are performed from the Wakanda server in a secure manner.
    You can add calculated fields in the .js configuration file of the external model (see **Modifying the external file**). For more information, refer to the Attributes page of the Wakanda documentation.
  - **Combine extended datastore classes and restricting queries**: this powerful concept lets you control not only the attributes exposed but also the data that they can display. Extending a *datastore class* means creating a copy (the derived class) that you can alter by adding calculated attributes or by deleting existing attributes. You can also associate a *restricting query* with it: in this case, any access to the data of the derived class automatically triggers this query, which returns only the records matching the criteria. This principle allows you to relate the data with the user connected to the Wakanda server. For example, in the context of a sales database, the query returns all the customers related to the current salesperson. Of course, only the derived class can be accessed by Web clients.
    You can create extended *datastore classes* and add *restricting queries* in the .js configuration file of the external model (see **Modifying the external file**). For more information, refer to the Programming Restricting Queries page of the Wakanda documentation.

  **Note:** The following minimum configuration is required to support restricting queries in 4D Mobile:
  - 4D and 4D Server **v14.1**
  - Wakanda Enterprise Server **v8**