

Web Database Tutorial

1. Constants

The WebClient_Constants file is one of the most important files in the Comparatio Flex framework. The first thing you need to define is the SERVER_URL variable. The variable should contain the address to where your Comparatio WebServer is running. For example:

```
public static var SERVER_URL:String = "http://localhost:8080/";
```

Next you will need to define the databases that your system will pull data from.

```
public static const DATABASE_NAME:String = "demo_database";  
public static const DATABASE_NAME_1:String = "comparatio_users";
```

Also, define the list of methods. Each unique list view will require a method for counting records and a method for getting the records. Each of these is simply a unique identifier that will be used later in the SQL Provider.

```
public static const METHOD_GET_NUM_OF_TEST_TABLE:String =  
    "getNumOfTestTable";  
public static const METHOD_GET_TEST_TABLE:String =  
    "getTestTable";
```

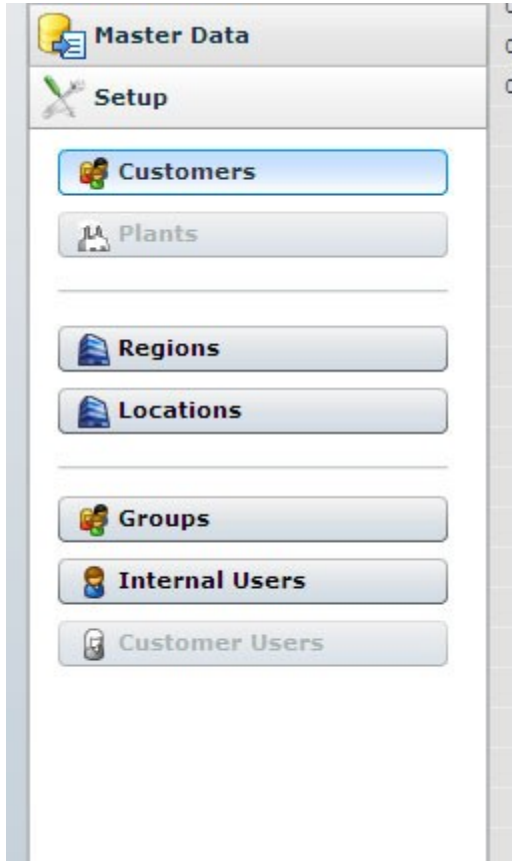
Again, these are used as unique identifiers, so there is no requirement for how you define these strings, as long as they are unique.

Finally, you will need to create unique caller IDs for each of the list views.

```
public static const CALLER_ID_TEST_TABLE:int = 0;
```

Like the method constants, make sure these are all unique. Later on you may need to add other things in this class for icons and relation buttons.

2. Menu Buttons



A. Adding a Menu Button

To add a menu button to the left hand navigation menu, begin by opening the class

`WebClient_WorkspaceView.mxml`.

Through the design view, you can see that this menu area is an Flex accordion component. Add a button to the menu and give it a label, icon, and ID. In the “On click” field, call the

`selectButton()` method by entering the following:

```
selectButton(event);
```

Position the new button where you want it and you’re finished.

B. Connecting a Menu Button to a List View

Now that your new menu button has been created, you need to define what list the button will open. Still in the `WebClient_WorkspaceView.mxml` class, switch to the source view. Within method `selectButton()` there is a switch statement that handles button clicks. Create a new case in the switch for your new button using the button ID as the case trigger.

Each case makes a call to the `setParameters()` function in the `WebClient_List` class. Copy a case from an existing button. There are a number of parameters that you can set, but we will only cover the most important ones here. The rest have comments next to them explaining what they do.

```
WebClient_List.setParameters(  
    currentState,  
    "Table Display Name",  
    "",  
    WebClient_Constants.DATABASE_NAME,  
    "table_name",  
    WebClient_Constants.METHOD_GET_XML,  
    WebClient_Constants.CALLER_ID_FILES,  
    WebClient_Constants.PAGE_CONTROL_ID_DEFAULT,  
    WebClient_Constants.METHOD_GET_INDICES,
```

```
WebClient_Constants.METHOD_GET_NUM_OF_FILES,  
WebClient_Constants.METHOD_GET_FILES,
```

- `currentState` is where you define what state you want the `ListView` to use.
- `Table Display Name` is the table name that the list will use in all display areas.
- Leave parameter 3 alone.
- This parameter is the name of the database the list will get its data from. – *Defined in Constants*
- `"table_name"` is the actual table name in the database that this list will get its data from.
- Leave parameter 6 alone.
- Defines which `callerID` will be used. – *Defined in Constants*
- Leave parameter 7 alone.
- Leave parameter 8 alone.
- Defines the query to use for counting the number of records in this list. – *Defined in Constants*
- Defines the query to use for getting the records in this list. – *Defined in Constants*

Once you have those parameters defined, you should also add your button to the `controlButton()` method. Search for that method name and add your button to the list of existing buttons.

Adding Totals to a List

In the `setParameters()` method, there are three things you need to set to display totals. First, the `currentState` should be set to a state containing a total field. Examples include `ListView_TotalState` and `ListView_TotalState_NoMenu`. Next, there is a flag for displaying total dollar values; set this to true. Finally, there is a column name for the total dollar value. This is the name of the column in your database query that contains the total. Total columns should be included and returned in the query for counting the number of records.

Export to PDF function

As you can see from the above example of using states to include total, you can also use states to add exporting to Alive PDF functionality. Choose a suitable state including PDF or create your own state by deriving from existing states and adding buttons. When creating states, remember to create a short state for the new state, as narrower screen will need to omit the button text for all the buttons to fit on the screen.

Following the File List PDF example, create your PDF button event handler and corresponding codes. A lot of the PDF codes should stay the same; the only thing you need to modify a lot is again, the SQL queries, along with your desired PDF content and format, in the file `Printing/WebClient_Printing.as`.

For details on how to use the AlivePDF API, go to <http://www.alivepdf.org/> for the complementary documentation on AlivePDF classes.

3. Lists

[illegible]

Each list requires you to create a number of configuration variables in the `WebClient_Workspace.as` class. Each column in your table will need to have things defined for each variable set. The easiest way to understand is to look at the examples of the code already there for the users table.

These variables are used for a number of processes, including sorting, searching, column visibilities, and labeling.

In order for the list to get the data, you need to write the sql queries. Open your `WebClient_SQLProvider`. Within function `getRequest()` you will find a switch that is based on the method constants you defined in the constants file. Add your methods to the switch and look at an existing function for how to write your SQL calls.

You should have two methods for each list; a method for counting the number of records, and another method for getting the actual records.

4. Forms

The screenshot shows a web application window with a title bar containing 'Close' and 'Save' buttons. The main content area is titled 'Internal User' next to a user icon. A central form contains the following fields:

- User ID:
- Password:
- Group:
- User Full Name:
- Email:
- Location ID:
- Location Name:

There is a 'Location' button with a folder icon and a red 'X' icon next to the Location ID field.

To create your forms, you first need to create the mxml file. Copy an existing form (renaming all the form specific references) and build your form in design mode, being sure to assign IDs to all fields.

Within the `initForm()` method, initialize all the fields to an empty string. This ensures that every time the form is loaded, old data is cleared from the fields. There are other methods in here for dealing with relation fields that will not be covered in this brief tutorial.

The core of the form logic is located within the `ActionScript` file. Copy an existing one of these as well. Create a global variable for the primary, unique column of this table and initialize it in the constructor. In the `fillForm` method, write your query for the required data.

Next is the `saveData()` method. There is a statement for inserting records and one for updating records. Modify both. You may also require fields here before saving.

Scroll down to method `storeFieldValue()` and copy the existing examples for each of your fields. Do the same for method `isModified()`.

Finally, edit method `formDataHandler()` to fill the fields with the correct data from the server response.

After you have created these forms, they need to be added to the form viewstack. Open class `WebClient_FormView.mxml` and find the `updateMainFormView()` method. The

switch is based on the caller IDs defined in the constants class. Create a case for each of your forms and copy the logic from existing examples.

Lastly, switch to the design view and add your forms to the viewstack, be sure to make the IDs the same as in the cases in the switch.

5. Dependencies

The screenshot displays a web application interface. At the top, there is a 'Region' form with two input fields: 'Region Name' containing 'Region 1' and 'Regional Manager' containing 'Regional Manager'. To the right of the 'Regional Manager' field is a 'Users' button with a red 'X' icon. Below the form is a tabbed interface with two tabs: 'Locations' (selected) and 'Internal Users'. The 'Locations' tab contains a table with the following data:

Location No	Location Name	Branch Manager	Region Name
00001	Location 1	Branch Manager	Region 1
00002	Location 2		Region 1
00003	Location 3		Region 1

Below the table, it says 'Locations (3 Records)'. To the right of the table is a pagination control showing '1 / 1' and a 'Records/Page' dropdown set to '10'.

Dependency views are much like forms, but only require an mxml class. Copy and existing dependency example and rename it. In the design view, add tabs for each dependency you wish to display, then find the `populateDependencyView()` method. Create a new case for each tab, editing the parameters with the correct database, table, `callerID` and method for the given list in the same way you did when setting up your navigation buttons.

You will also need to edit the `tabNavigatorChangeEventHandler()` method and add a case for each tab and assigning the correct `callerID` to the method calls.

After you have created your dependencies, return to `WebClient_FormView.mxml` and go to the `openDependencyView()` method. Here you need to add a case for each dependency, like you did for the forms. As with the forms, you also need to switch to the design view and add your dependencies to the dependency view stack.

6. Server Configuration

```
WebServer.ini - Notepad
File Edit Format View Help

[HTTP_Server]
HTMLDirectory=
Port=8080

[Security]
DefaultCompressAndEncrypt=yes

[Debugging]
Debug=yes

[Databases]
DatabaseCount=2

DatabaseType1=mysql-5
DatabaseServerIP1=localhost
DatabaseName1=comparatio_demo_users
DatabaseUser1=
DatabasePassword1=

DatabaseType2=sqlite-3
DatabaseServerIP2=localhost
DatabaseName2=d:\Sqlite\comparatio_demo_webclient.db
DatabaseUser2=
DatabasePassword2=

[Application User Tables]
ApplicationCount=1

ApplicationName1=demo
UserDatabase1=comparatio_demo_users
UserTable1=us1_user
UserNameField1=us1_login
UserPasswordField1=us1_pass

[Default User Tables]
UserDatabase=comparatio_demo_users
UserTable=us1_user
UserNameField=us1_login
UserPasswordField=us1_pass

[SMTP_Server]
SMTPServer=
SMTPUser=
SMTPPassword=
```

To configure the server, open `WebClient.ini`. Encryption should be set to yes, as well as Debug.

Next, define the number of databases. The example above shows two separate database connections: table `comparatio_demo_users` and `comparatio_demo_webclient`. As you can see, one of these connections is a sqlite3 database and the other is a MySQL database. Comparatio's WebServer is capable of utilizing a number of databases. Contact us for a full list. Fill in your username and password.

The Application User Tables section is used to define which tables contain user accounts for login. The default user table is defined, as well as individual tables for specific projects, identified by the `ApplicationName` parameter. This application name is set within the Constants file in your project. This flag allows multiple projects to use the same server and to share databases and tables.

Finally, you can enter the mail server settings for email functionality.