

Introduction to the 4D Ajax Framework

Presented by: **Joseph Resuello**

SESSION INTRODUCTION

This session is for 4D Developers who have no prior experience with the Web 2.0 Pack. This introductory session will guide the developers through hands-on demos, which will lead them to gain an overall understanding of the framework. They will also learn how to develop Ajax applications using the 4D Ajax Framework. A sample database is provided with a demo of the 4D Ajax Framework (4DAF) component already installed.

This session will focus on building Custom Ajax applications and it will do so without much interaction in the Client. The example demos will target the Data Grid object and Dashboards.

SESSION REQUIREMENTS

In this session we will create Ajax applications, which will be previewed using Mozilla's Firefox browser. This is to take advantage of the Add-on extension, Firebug, which is an excellent web development tool.

The Firebox browser can be found at:

<http://www.mozilla.com/en-US/firefox/>

Firebug can be found at:

<http://www.getfirebug.com/>

Developers must also have an HTML editor.

The latest revision of the 4D Ajax Framework (4DAF) v11 can be run on 4D 2004 or 4D v11 SQL. Thus, the developer must have at least one of these versions of 4D to run the interpreted source database that is included in the session materials.

SECTION 1: PREFACE ON AJAX

Before we begin discussing what Ajax is, perhaps it is best to start discussing what Ajax isn't. Many 4D Developers may not have much experience with web server Ajax is a commonly misunderstood phenomenon, and now would be a good time to dispel some common misconceptions and myths about it.

What Ajax isn't

Here are some common misconceptions as to what many believe Ajax is.

A Technology

Ajax is not a new technology. In fact, Ajax uses common technologies that existed long before it came to be. Thus, there is no Ajax Plug-in. Users do not need special software such as browser plug-ins or desktop applications in order to use an Ajax application.

A Language

To the relief of many, Ajax is not a new programming language. Developers cannot *code* in Ajax so, fortunately, many do not have to learn a new technology. However, as will be explained further later, Ajax does use a scripting language and other languages and if developers are not already familiar with these existing technologies then building an Ajax application from scratch will require some time to get the hang of.

Proprietary

Although it is a highly popularized buzzword, Ajax is not the name of a company or a product. In this respect many confuse Ajax with a proprietary technology such as Flash, which can also be used to create interactive web applications. One particular downside for proprietary technologies is that users need to download additional software in order to view and use it. Fortunately for Ajax, this is not the case.

What Ajax is

With those misconceptions out of way, let's unravel the truth behind what Ajax is:

A Technique

Ajax is a way of using existing technologies to create interactive web applications. These existing technologies are JavaScript, XML, HTML, and CSS. As mentioned earlier, these technologies existed long before the Ajax phenomenon came to be. Thus, Ajax is more of a novel and effective technique of using these technologies to create interactive web applications.

It uses open standards

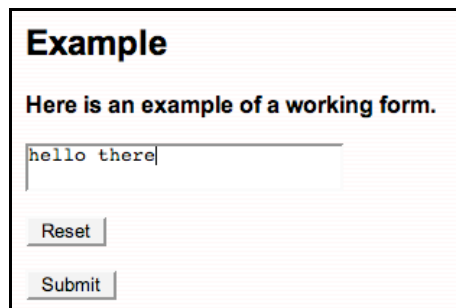
Without getting into too much detail about web standards and such, the point here is that Ajax uses open technologies. Thus, Ajax applications can be implemented in any browser, free of legal constraints. Any recent browser is Ajax-compatible (IE 5.0+, Mozilla 1.0+, Firefox 1.0+, Netscape 7.0+, and Apple added it to Safari 1.2+).

What are the characteristics of an Ajax application?

Ajax applications are interactive. In the world of web applications, this is a novel idea. Before Ajax came into the picture, the line between web applications and desktop applications was quite distinct. Now that line is becoming more and more grayed-out as Ajax web applications now have the GUI features with much of the seamlessness and responsiveness of desktop applications.

Few or No Page Reloads

Web applications from the old days are heavy on page reloading. Make any request to the web server, and the web page would have to be reloaded so that you can witness the changes.



The image shows a web form titled "Example". Below the title is the text "Here is an example of a working form." followed by a text input field containing "hello there". Below the input field are two buttons: "Reset" and "Submit".

The above picture is an example a web form that does not use Ajax. Once the user is done entering information they can hit the *Submit* button. Once they hit the *Submit* button, the page reloads to present something like the following:



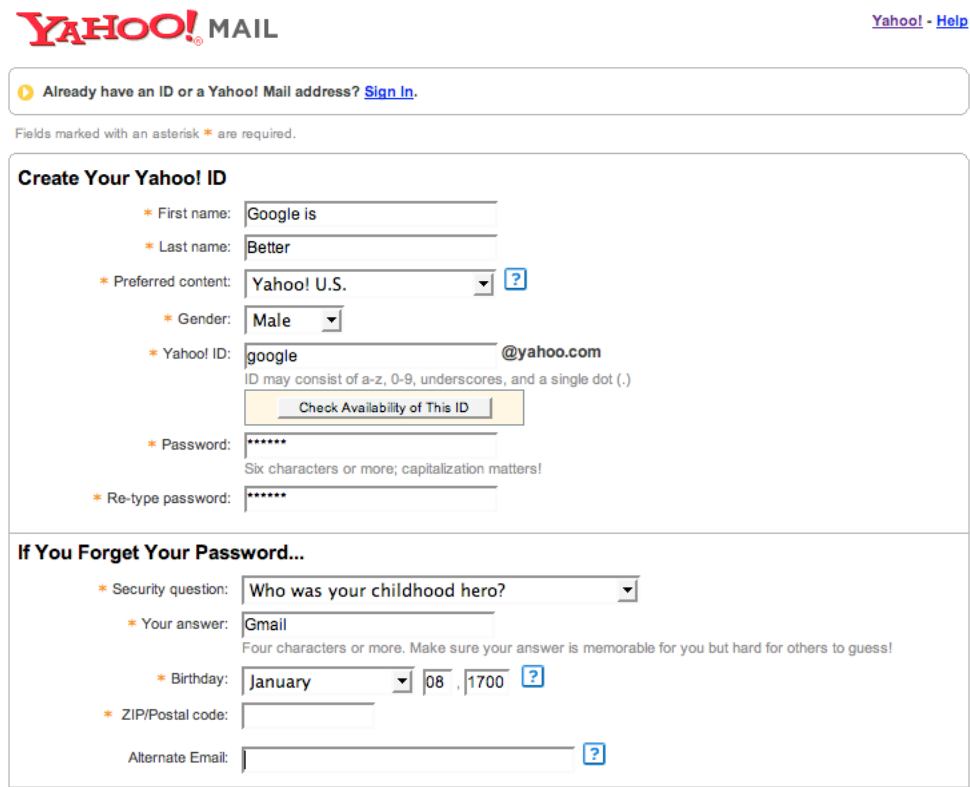
The image shows a thank you page with the heading "Thanks!". Below the heading is the text "Your sample form has been submitted." followed by a "Return to form" button.

The form is fine as it is, but it isn't great. There are some noticeable pitfalls to having a web application such as this one.

What if the user entered incorrect information?

Suppose the user entered too many digits for a phone number field. Suppose the email address they entered was badly formed. The user would not have been informed of their mistakes until after they hit the *Submit* button.

Take a look at the following example.



The screenshot shows the Yahoo! Mail registration page. At the top, the Yahoo! Mail logo is on the left and a "Yahoo! - Help" link is on the right. Below the logo is a link: "Already have an ID or a Yahoo! Mail address? [Sign In.](#)". A note states: "Fields marked with an asterisk * are required." The main section is titled "Create Your Yahoo! ID". It contains several fields with asterisks indicating they are required:

- * First name: "Google is"
- * Last name: "Better"
- * Preferred content: "Yahoo! U.S." (with a help icon)
- * Gender: "Male" (dropdown menu)
- * Yahoo! ID: "google" followed by "@yahoo.com". Below this, a note says: "ID may consist of a-z, 0-9, underscores, and a single dot (.)". There is a button "Check Availability of This ID".
- * Password: "*****" with a note: "Six characters or more; capitalization matters!"
- * Re-type password: "*****"

Below the "Create Your Yahoo! ID" section is a section titled "If You Forget Your Password...". It contains several fields with asterisks indicating they are required:

- * Security question: "Who was your childhood hero?" (dropdown menu)
- * Your answer: "Gmail" with a note: "Four characters or more. Make sure your answer is memorable for you but hard for others to guess!"
- * Birthday: "January" (dropdown menu), "08" (text input), "1700" (text input) with a help icon.
- * ZIP/Postal code: (empty text input)
- Alternate Email: (empty text input) with a help icon.

This is part of the registration form for creating a new *Yahoo! Mail* user account. Now suppose we had made mistakes in this form. The following is what we would see after a hit of the *Submit* button and a page reload.



Please correct the entries highlighted in yellow. We either had trouble understanding those fields, or need more information.

- Someone has already chosen that **Yahoo! ID**. Please choose another Yahoo! ID. For help, please click the Find an Available ID button below.
- You didn't specify a valid **Birthday**.
- You didn't specify an understandable **Zip or Postal Code** for United States(Please verify that you have selected the correct **Country**.)
- You need to enter the **code** shown to verify your registration.

Fields marked with an asterisk * are required.

Create Your Yahoo! ID

* First name: Google is

* Last name: Better

* Gender: Male

* Yahoo! ID: google @yahoo.com

ID may consist of a-z, 0-9, underscores, and a single dot (.)

Find an Available ID

* Password: [Not Shown for Your Protection]

If You Forget Your Password...

* Security question: Who was your childhood hero?

* Your answer: Gmail

* Birthday: [Select a Month] dd yyyy ?

* ZIP/Postal code:

* Country: United States

We would be presented with the same form but with indicators showing us where we messed up. The downside of such an application is that the user had wasted valuable time during the page reload. It would have been nice to have been informed of the mistakes as the fields were being entered. Also, this makes for an unpleasant and rather dry user experience. The 1-2-3 Step of 1) *Enter Information*, 2) *Submit Information*, and then 3) *Cross Fingers and Hope the Information was Well Received* shows that there is clearly a lack of interactivity going on. An Ajax application, on the other hand, could validate information as it was entered.

Ajax to the Rescue

Speaking of page reloads and email registration sites, let's take a look at the email registration form for *Gmail*. We will notice that it has some Ajax capabilities. Here is the field for entering your email password.

Choose a password: Password strength:
Minimum of 8 characters in length.

Right now no information is entered yet.

Choose a password: Password strength: Too short
Minimum of 8 characters in length.

However, as I type, a notification appears to the right. This one tells me my password length is too short.

Choose a password: Password strength: Weak
Minimum of 8 characters in length.

Here I am being told that my password is not that strong. Hackers may be able to figure it out quite easily.

Choose a password: **Password strength:** Strong
Minimum of 8 characters in length.

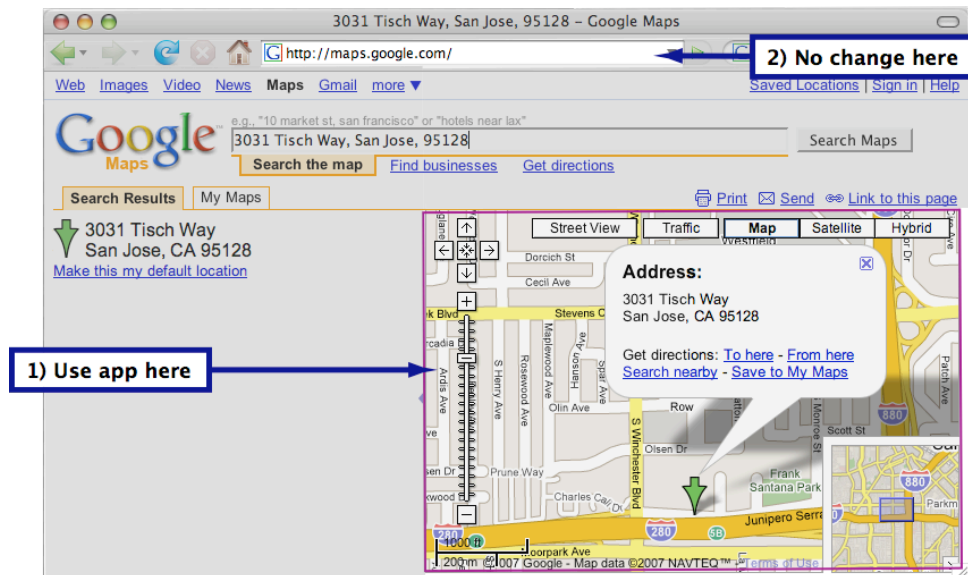
Now my password is satisfactory. I noticed that as I was typing in possible passwords, I did not have to wait for a page reload to see these notifications. These notifications appeared in real time, asynchronously as I was typing. This makes the experience interactive, and it is made possible by Ajax.

Graphical responsiveness

Indicators presented in the example above show how Ajax can create an interactive user experience. Those examples, however, represent only the *tip of the iceberg* when it comes to the graphical responsiveness that Ajax applications are capable of. Here now are more graphically rich examples of Ajax applications.

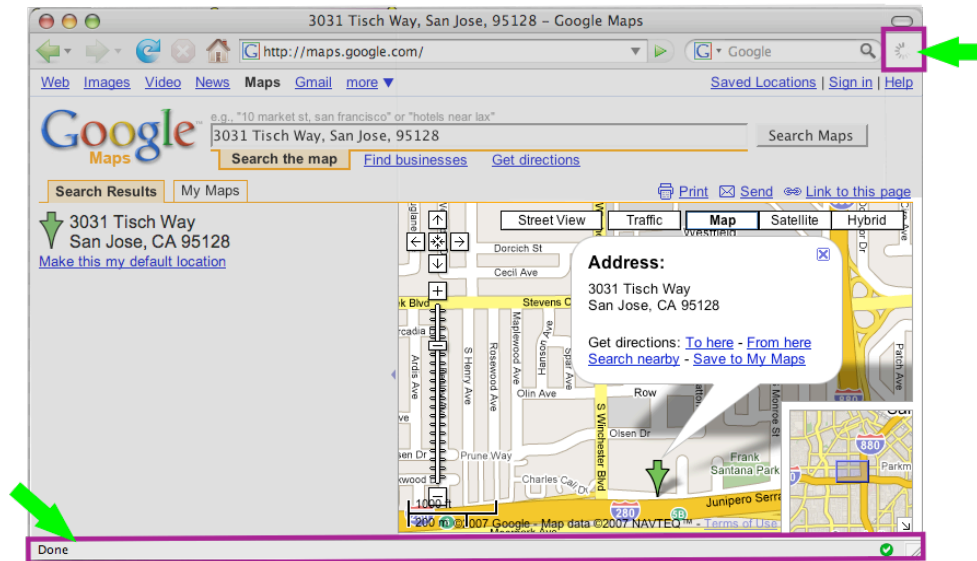
Ajax Characteristics

There are a couple of things that should be really noted when seeing an Ajax application in action.



The address bar does not change: Since only parts of the web page are being updated on the fly, there is no need for a new HTML page to appear. The power of Ajax means that only specific elements on the page are changed.

Status bar and Loading indicators hardly load



There is minor indication of loading while interacting with this the Ajax application. This makes the user experience asynchronous. Progress bars may indicate some activity between the web page and the web server, but that activity does not interrupt the user.

How Does Ajax do it?

I know what you are thinking and the answer to your question is, “No, black magic is not what makes Ajax applications work the way they do”. In this section we will get into some detail about the technologies that make an Ajax application run.

XHTML and CSS

This is what makes the user interface of the web application. Why XHTML and CSS? Because they represent an interface that any browser can display. Being compatible on any browser (whether it be proprietary like Internet Explorer or open source like Firefox) is one of the core strengths of an Ajax application.

DOM

DOM stands for Document Object Model, and it is a widely accepted programming interface that allows you to update specific areas on the page.

XML

XML is a tag language, much like HTML but it allows for more explicit tags. XML is the format that information is passed between the web application and the web server.

XMLHttpRequest

This is an object supported by most browsers that allows the application to make requests to the server without having to reload the page in order to process the request.

JavaScript

JavaScript is the scripting language that acts as the glue that puts all of these pieces together. Ajax applications rely heavily on this technology. One of the downsides of JavaScript is that 4D developers may not be familiar enough with this language to create Ajax applications.

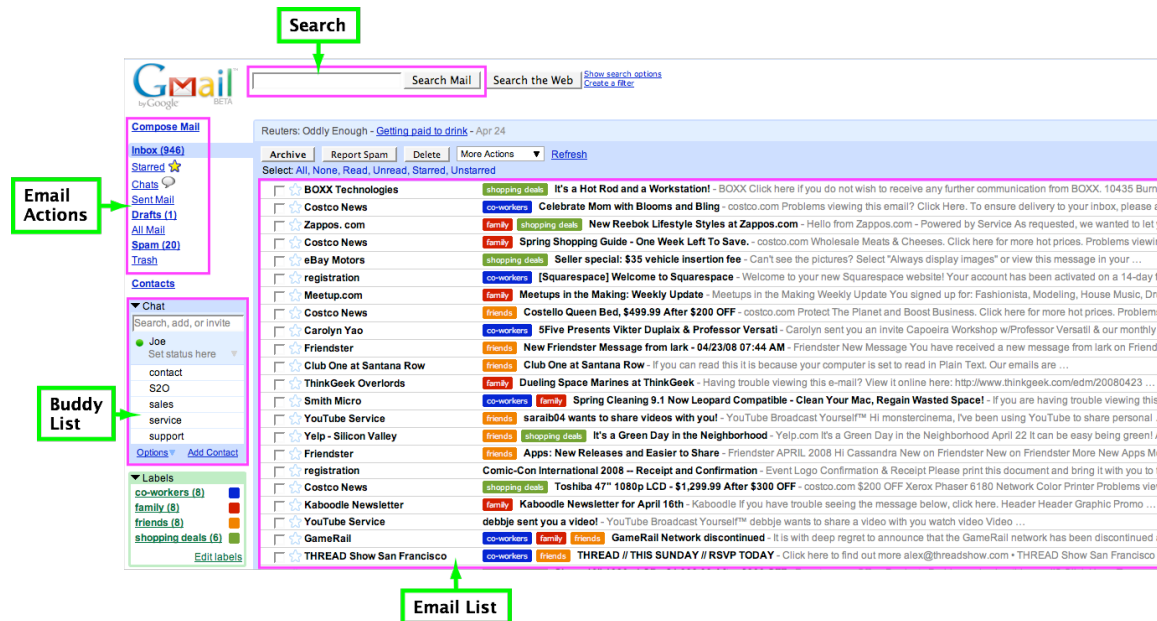
SECTION 2: RIA VS. WEBPAGE WITH AJAX ELEMENTS

One key concept to understand is that two types of web pages can be implemented with Ajax: 1) A full Rich Internet Application (RIA) or 2) A static webpage with Ajax objects on it.

Rich Internet Applications

Rich Internet Application's (RIA's) are web applications with interactive objects fully and seamlessly integrated together. A good example is an email application such as Gmail or Yahoo! Mail.

Once a user logs into the page they enter a web-based mail application that fully stands on its own. All objects are interactive and integrated with one another. All interactivity occurs on the same page. For example, the user can hit the 'Compose Email' link and a particular area of the page refreshes to compose a new email. *As mentioned in Chapter 1, an RIA's URL does not change even though there may be interactivity with the backend.* Ajax does all the heavy lifting by updating only certain areas on the page that need updating depending on the user's request.



A static webpage with Ajax objects on it

Conversely, an RIA may not be necessary for a developer. They may just need some static web pages with static content, and all the interactivity they may need would only reside within an interactive Ajax object embedded on the page. For example, maybe all that is needed is an interactive data grid. The grid itself would dynamically update if the data is changed from the backend, live searches can be performed on the content, and it could responsively support in-line editing.

This particular scenario fulfills the scope of this introductory Ajax training course. RIA's, on the other hand, are much too complex and advanced to be covered within this introductory course. However, being able to embed objects onto a static page is one gigantic leap forward on the road to creating Rich Internet Applications. In this course we will use template web pages and we will embed interactive Ajax objects onto them using the objects provided by the 4D Ajax Framework.

SECTION 3: GETTING ACQUAINTED WITH THE 4DAF

Installation

To get acquainted with the 4D Ajax Framework (4DAF), installation is probably the best place to start.

NOTE: For installation instructions please refer to the document "4DAF Install & Upgrade".

What's So Different After Installation?

We have a 4D database with the 4D Ajax Framework (4DAF) component installed. What does that mean exactly? Launch the sample database, and let's highlight some noticeable changes.

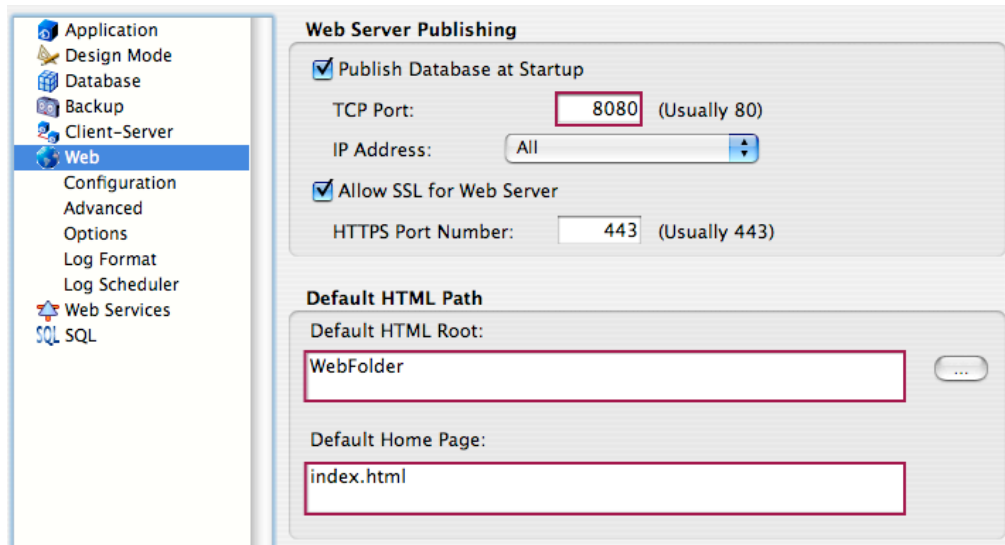
Web Server Preferences

In 4D, go to Preferences -> Web. Take note of the publishing port, HTML root folder, and the default home page file.

Publishing port – The port the web server publishes on.

HTML root folder – Root directory containing the web files for the web page.

Default Home Page – The default HTML page that is loaded when users connect to the web server.

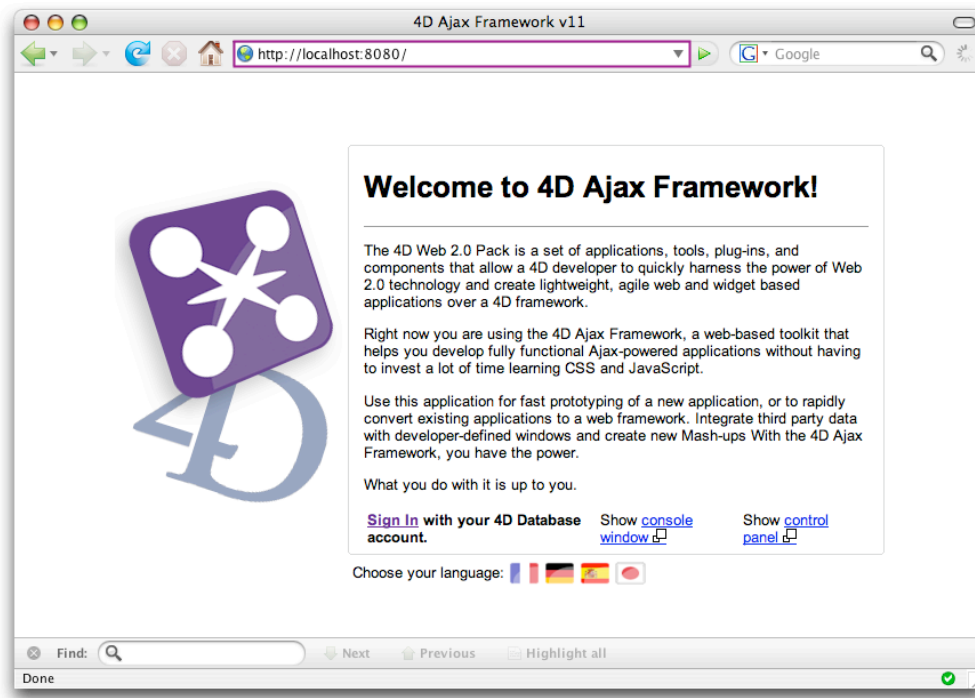


Let's test what happens when we connect to the web server.

On the same machine that is running the sample database, open up your Firefox web browser and connect to:

<http://localhost:8080>

NOTE: 'Localhost' means the same machine you are currently using. '8080' is the port that the web server is published on.



Notice that by connecting to the web server at *http://localhost:8080* we are directed to the default HTML page (*index.html*) specified in 4D Preferences.

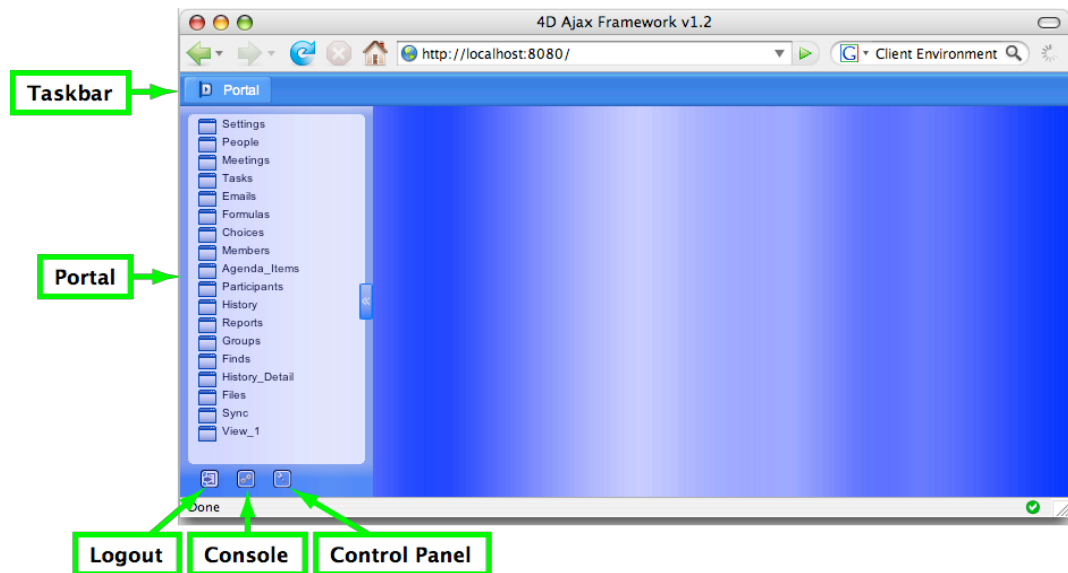
Web Folder files

Where is the default HTML page *index.html*?

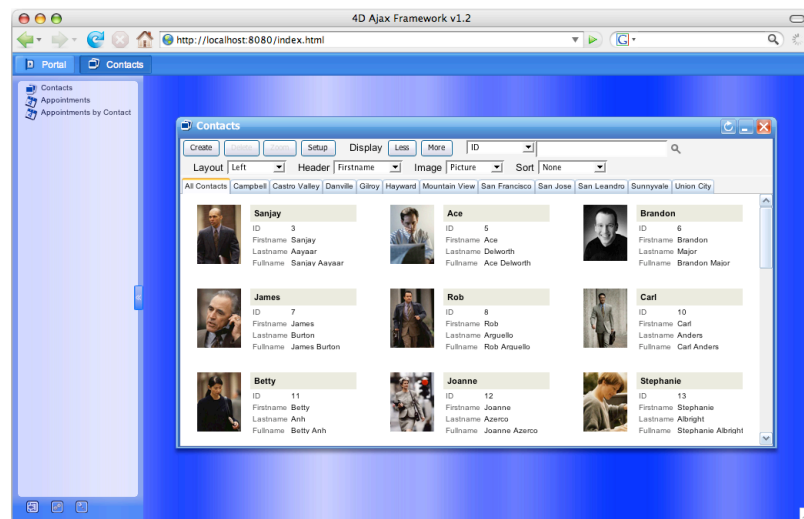
Navigate to the test database in your operating system and next to the structure file is the 'Webfolder' folder. The 4D Ajax Framework installs important files in this location such as JavaScript libraries, CSS themes, etc. However, we will focus on the HTML files located at the top level at the moment. These are the HTML pages the users see when they connect to your webserver.

The 4DAF Client

Let's take a look at the 4D Ajax Framework Client. The Client itself is an RIA that serves your database on the web. To see the Client, hit the Sign In link at <http://localhost:8080>. Enter username 'Administrator' with no password.

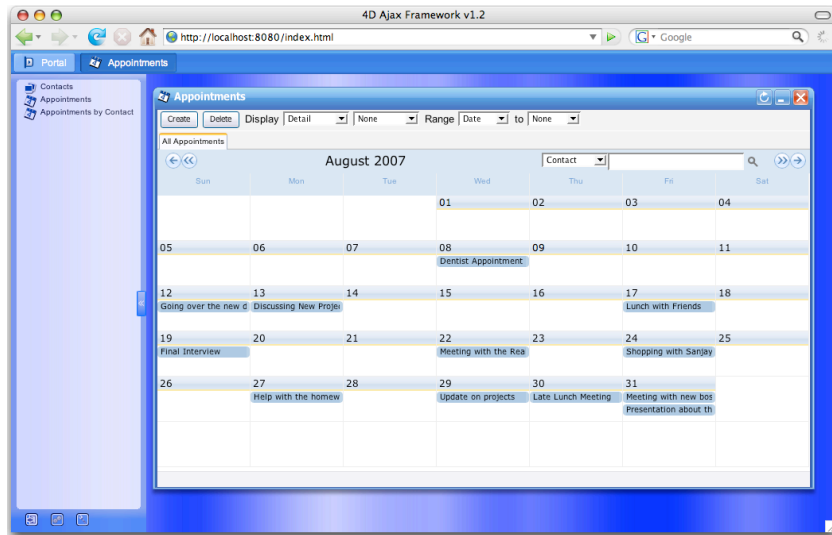


In this environment, 4D tables are listed in the Portal area to the left. Click on the table name to load it:



A table can be represented graphically, just like the Image Matrix style shown above, which represents your records visually. Now remember that this is an Ajax environment, so your users can interact with these objects asynchronously. They can perform live searches, they change styling and formatting, they can modify records, etc. Everything is asynchronous. Everything is updated live.

Here is an example of another Ajax object, the Calendar:



As long as your table has date fields, they can be represented in this Calendar object. What is nice is that these objects (the Calendar, the Image Matrix, etc.) are already created for you. All that you have to do is assign what object style best represents your tables.

As wonderful as the Client is, let's move on to creating your own pages with the framework. Thus, we'll try to minimize time spent on the Client and focus on creating web pages with Ajax objects embedded in them.

SECTION 4: THE TEMPLATE PAGE

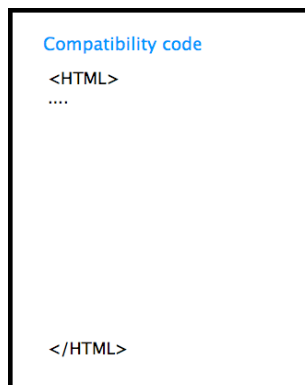
Creating a web page with Ajax objects embedded into it can be a daunting task if you are a 4D developer with not much web development experience. Have no fear; a Template Web Page is here. This page will set everything right so that the only thing you need to worry about is deciding what type of object to embed.

Let's take a closer look at the Template Page from top to bottom.

The First Lines are for Compatibility

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

The first couple of lines ensure the best compatibility with the 4D Ajax Framework objects. For instance, we are stating that Transitional XHTML mode is used.



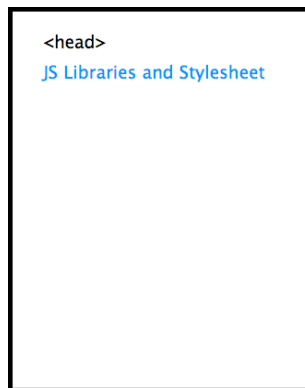
Not much has to be explained here. Just be sure to include these lines of code as soon as possible when creating a web page from scratch, which you intend to use for the 4DAF.

Note: This code appears before the <HTML> tag.

Framework Libraries and Stylesheets

The next set of lines specify which JavaScript libraries from the framework the page should refer to as well as the stylesheet which will determine the look of the 4DAF Objects on the page.

```
<script language="javascript" type="text/javascript"
src="dax/dev/callbacks.js"></script>
<script language="javascript" type="text/javascript"
src="dax/js/framework.js"></script>
<script language="javascript" type="text/javascript"
src="dax/js/dax.js"></script>
<link rel="stylesheet" href="dax/themes/leopard/leopard.css" media="all"
type="text/css" title="Leopard" />
```



Note: This code appears within the top of the <head> tag. Since the browser reads this information in a top to bottom manner, it is best to include these libraries early before any actual JavaScript code specific to the 4DAF is included soon below.

JavaScript code

4D Developers can add their JavaScript code within the following:

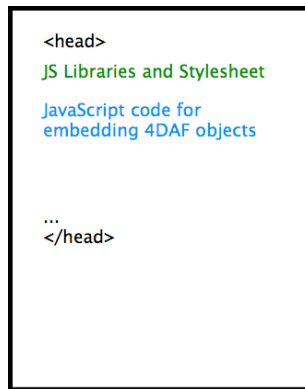
```
<script language="javascript" type="text/javascript">

dax_loginSuccess = function()
{

    // Insert code to embed 4DAF objects here

}
</script>
```

Developers can actually include JavaScript anywhere within the <script> tag. However, it is recommended to insert code within the *dax_loginSuccess function* because it is a good place for beginners to start coding. The 4D Ajax Framework provides the *dax_loginSuccess* function, and it is called soon after the page has been validated through the login process (explained soon).



Login

In this Template page the 4DAF's *dax_login* function is called within the `<body>` element tag.

```
<body onload="dax_login('Guest', '')">
```

Here, user 'Guest' is entered with no password. The same *dax_login* function checks if the user can be validated by the 4D Users and Groups password system or if the developer is rolling out their own password system.

In this particular case, we are rolling out our own pass system. The username 'Guest' is being allowed permission to see the 4D Ajax Framework objects that will be embedded on the page.

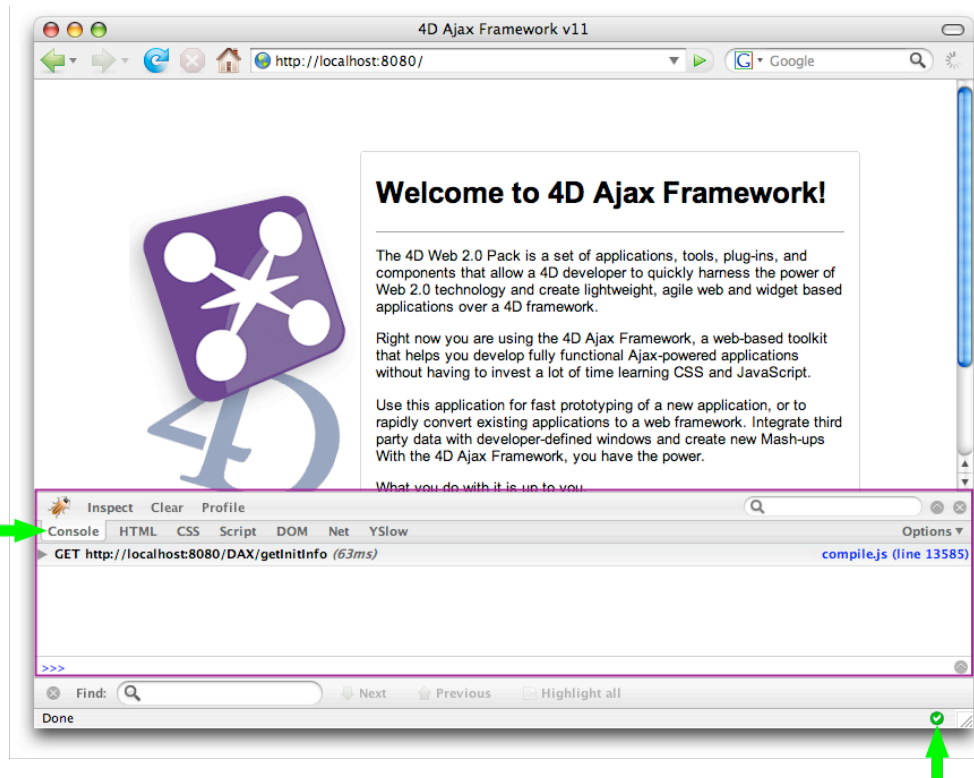
Firebug

It is recommended that the developers install Firebug to the Mozilla Firefox web browser. This is a powerful tool for debugging the front end side of things during web development.

Although Firefox and Firebug is a great setup for debugging purposes, it is best practice to constantly test your Ajax applications in all browsers (and all platforms) during development.

Firebug has some important features that are essential to web development:

Console: See the XML requests that are received from the web server. Any errors that occur will be reported here.



Open Firebug by hitting the green checkmark icon in the bottom right of Firefox. Once it's loaded, hit the Console tab to see what requests the Console has logged.

Inspect CSS: This is a great tool for understanding the layout of elements on your page. See what CSS styles are applied for certain elements on the page.

SECTION 5: DEVELOPER HOOKS

Developer Hooks are there to give the developer as many ways as possible to implement their own requirements and designs into their solution. They are deemed “hooks” because they are regarded as situations where developers can hook-in to insert or inject their own custom code. Each hook has its own defined set of parameters and specified result type. The 4DAF Developer Hooks can be categorized in the following manner:

- Callback installation and event response
- DCS creation and record handling
- DDW installation and assignment
- Choice List installation, assignment, and manipulation
- User Groups
- Login and Session control
- Query control
- Record control
- Global preferences

The following tables give a general description of each Developer Hook:

Callback installation and event response

Method Names	Description
<i>DAX_DevHook_CB_EventFired</i>	[Callbacks] This method is called anytime a field receives an enabled event. Supported events are On Load or On Data Change (applicable only to input fields).
<i>DAX_DevHook_CB_Install</i>	[Callbacks] This method is called at startup to allow the developer to enable events for specific fields (applicable only to input fields).
<i>DAX_DevHook_InstallCallBack</i>	[Callbacks – Backward compatibility for pre 1.2 version] This method was the previous way of adding a Callback to handle an event. It should no longer be used, but is kept for compatibility with version 1.1 (applicable only to input fields).

DCS creation and record handling

Method Names	Description
<i>DAX_DevHook_DCS_RecordDelete</i>	[DCS] This method is called whenever 4DAF is asked to delete a DCS record. It is up to the developer to take appropriate action.
<i>DAX_DevHook_DCS_RecordSave</i>	[DCS] This method is called whenever 4DAF is asked to save a DCS record. It is up to the developer to take appropriate action.
<i>DAX_DevHook_DCS_SetSelection</i>	[DCS] This method is called whenever 4DAF needs the selection for a DCS View added through the <i>DAX_DevHook_DCS_ViewAdd</i> method. The developer populates their arrays and returns them.
<i>DAX_DevHook_DCS_ViewAdd</i>	[DCS] This method is provided for the Developer to add custom Views to the 4DAF web interface using arrays as the data source.

DDW installation and assignment

Method Names	Description
<i>DAX_DevHook_DDW_Install</i>	[DDW] This method is provided for the Developer to add DDW Views to the 4DAF web interface.

Choice List installations, assignments and manipulations

Method Names	Description
<i>DAX_DevHook_InstallChoiceList</i>	[Choice List] This method allows developers to install or overwrite a choice list that will be used for a field (applicable only to input fields).
<i>DAX_DevHook_ListContents</i>	[Choice List] This method is provided for the Developer to override a list before it is sent to the front-end.

Users and Groups

Method Names	Description
<i>DAX_DevHook_GetGroupsList</i>	[Users/Groups] This method is provided for the Developer to override the default 4DAF Groups system. If you have a users/groups system in place you can override the 4DAF group system here. This should be used in tandem with DAX_DevHook_UserInGroup.
<i>DAX_DevHook_UserInGroup</i>	[Users/Groups] This method is provided for the Developer to override the default 4DAF Groups system. If you have a users/groups system in place you can override the 4DAF group system here. This should be used in tandem with DAX_DevHook_GetGroupsList.

Login and Session control

Method Names	Description
<i>DAX_DevHook_Login</i>	[Login] This method is provided for the Developer to override the default 4DAF login system. The default is to use the built-in Users and Groups system.
<i>DAX_DevHook_SessionValidate</i>	[Session] This method is provided for the Developer to override the default 4DAF session management system. The developer must also override DAX_DevHook_Login if they are taking control of session management.

Query control

Method Names	Description
<i>DAX_DevHook_OnQuery</i>	[Query] This method is called whenever 4DAF is about to perform a query. Developers can perform their own query in this method. This may be useful when performing queries on a separate lookup table or if you wish to implement 'fuzzy matching' or Hash based queries.
<i>DAX_DevHook_QueryAdd</i>	[Query] This method is provided for the Developer to add custom queries to the 4DAF web interface. All custom queries will be added to the standard list of queries available on the front-end admin area. They appear to the user as tabs in the selection window.
<i>DAX_DevHook_QueryFilter</i>	[Query] This method is called right before the XML that is sent to the front-end is constructed. If you need to remove any records from the selection that was created you can do so here.

Record control

Method Names	Description
--------------	-------------

<i>DAX_DevHook_DeleteRecord</i>	[Record Control] This method is called whenever 4DAF is about to delete a record. The developer can accept or reject the action and also handle any other actions, such as logging, that is required.
<i>DAX_DevHook_SaveRecord</i>	[Record Control] This method is called whenever 4DAF is about to save a record. The developer can accept or reject the action and also handle any other actions, such as logging or deleting related records.

Global Preferences

Method Names	Description
<i>DAX_DevHook_Preferences</i>	[Global Preferences] This method is provided for the Developer to override the default preferences of the 4DAF system

SECTION 6: TABLES, DCS, AND VIEWS

There are 3 types of data structures that can be represented in the 4D Ajax Framework.

- Tables
- Developer Created Selections (DCS)
- Views

Tables


Tables and fields from your application's data structure translate automatically to the front end. No work has to be done to be able to see the tables from the 4D table structure in the Client.

Developer Created Selections

Developer Created Selections (DCS) allow the developer to create a custom table based tables and fields in the existing table structure. This gives the developer freedom in creating a data selection in any way and from any data source they want.

Unlike physical tables or Views, a DCS structure and selection must be created programmatically. The developer must compose the DCS structure using arrays. The content of the arrays will be read into the 4D Ajax Framework structure during the startup of the application.

Given the following 4D table, we can duplicate the table with a DCS by composing the arrays as follows:

	A1 {1}:="FirstName", {2}:="LastName", {3}:="DOB" A2 {1}:="Is String Var", {2}:="Is String Var", {3}:="Is Date" A3 {1}:=False, {2}:=False, {3}:=False A4 {1}:=False, {2}:=False, {3}:=False A5 {1}:=False, {2}:=False, {3}:=False A6 {1}:=False, {2}:=False, {3}:=False
---	---

Note that a DCS table with a specific name will be created only once. Each DCS must have a unique name.

Create a DCS Table

Based on the above example we are going to see how to create the DCS structure. There are two Developer Hook methods that are involved in the DCS creation process:

DAX_DevHook_DCS_ViewAdd: Create the arrays.

DAX_DevHook_DCS_SetSelection: Fill the arrays with data.

To create the Contacts table as a DCS, we must first edit the method

DAX_DevHook_DCS_ViewAdd.

```

` Method: DAX_DevHook_DCS_ViewAdd
ARRAY TEXT ($A1;3)
ARRAY LONGINT ($A2;3)
ARRAY BOOLEAN ($A3;3) ` False by default
ARRAY BOOLEAN ($A4;3) ` False by default
ARRAY BOOLEAN ($A5;3) ` False by default
ARRAY BOOLEAN ($A6;3) ` False by default

` set the names of the columns
$A1{1}:="FirstName"
$A1{2}:="LastName"
$A1{3}:="DOB"

` set the data types of the columns
$A2{1}:=Is String Var
$A2{2}:=Is String Var
$A2{3}:=Is Date

` Create DCS View named myContacts
$added_b:=DAX_Dev_DCS_AddCustomView ("myContacts";->$A1;->$A2;->$A3;->$A4;->$A5;->$A6)

```

The method *DAX_Dev_DCS_AddCustomView* is a protected component method that is used to add the composed DCS View into the 4D Ajax Framework structure.

Syntax: DAX_Dev_DCS_AddCustomView

- \$1 Text DCS Name (Must be a unique name)
- \$2 Pointer Pointer to a Text array for field names
- \$3 Pointer Pointer to a Longint array for the field types
- \$4 Pointer Pointer to a Boolean array for the unique property
- \$5 Pointer Pointer to a Boolean array for the mandatory property
- \$6 Pointer Pointer to a Boolean array for the non-enterable property
- \$7 Pointer Pointer to a Boolean array for the non-modifiable property

At this point the DCS table does not have any data in it yet. It is the developer's responsibility to supply it with a set of data. To do this, the developer must compose a data set inside the method

DAX_DevHook_DCS_SetSelection.

```
` Method: DAX_DevHook_DCS_SetSelection
Case of
    : ($1="myContacts")
ARRAY LONGINT (recIDs_al;0)
ARRAY TEXT (fNames_at;0)
ARRAY TEXT (lNames_at;0)
ARRAY DATE (dobs_ad;0)

    ` Populate data into arrays
ALL RECORDS ([Contacts])
SELECTION TO ARRAY ([Contacts];recIDs_al;[Contacts]FirstName;fNames_at;[
Contacts]LastName;lNames_at;[Contacts]DOB;dobs_ad)

    `Set the selection in DAX in the order defined
    `when we created the View
DAX_Dev_DCS_SetSelection(->recIDs_al;->fNames_at;->lNames_at;->dobs_ad)
End Case
```

In this method, we are not only populating the data into the fNames_at, lNames_at and dobs_ad arrays, we also need to generate the record id for each row. The record id will be used to identify a specific record during record modification and deletion. When a query request is made for any DCS record this method will be executed first to setup the initial selection. This selection will then be queried on with the criteria posted by the Web request.

This method has one parameter. This parameter contains the name of the requested DCS. In the above example, the Case-of is set up to check if the name of the requested DCS is "myContacts." If so, it will generate the initial selection for the DCS. The same approach will be used for record saving and deleting of the DCS as well.

IMPORTANT! Only process or inter-process arrays can be used to create a selection in method *DAX_DevHook_DCS_SetSelection*

The last call in the method is ***DAX_Dev_DCS_SetSelection.*** This method is another protected component method that should be used specifically in ***DAX_DevHook_DCS_SetSelection*** method.

Syntax: DAX_Dev_DCS_AddCustomView

\$1 Pointer to a Longint array containing unique record IDs
(Required)

\$_{2 to N} Pointer to array of DCS field data (where N=number of fields) in

DAX_DevHook_DCS_ViewAdd

Views

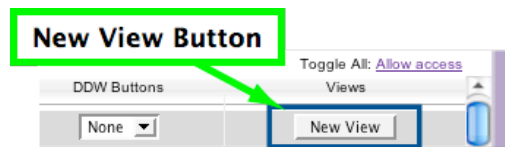
Views are the last type of table structure that can be represented in the framework. They are related tables represented as a single table, and they can be created quite easily in the Client.

Creating a View

Views can be thought of as virtual structures. They can be one-to-one copies of tables in your structure, or they can contain all fields from many tables in a relationship.

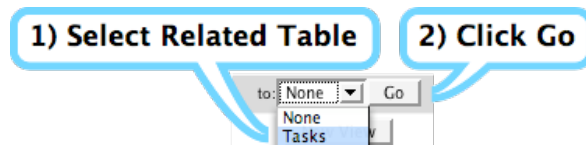
In the 4D Meetings Demo database, the Meetings table is the One table and the Tasks table is the Many table in a One-to-Many relationship.

To create a View you must first go to the One table in the relationship (in this case, the Meetings table). Click the 'New View' button.

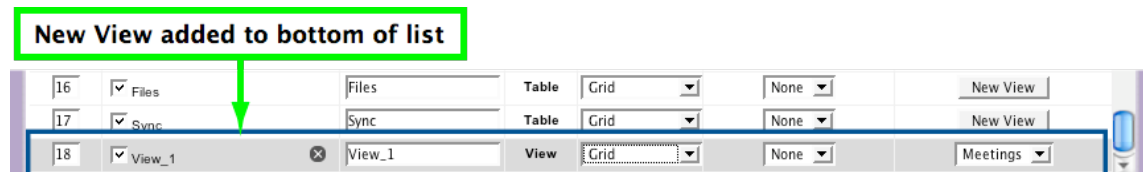


The current table selection (ie. Meetings table) can be thought of as a *starting point*. The table you select from the View pull-down menu (ie. Tasks table) can be thought of as an *ending point*. Any fields encompassed between the *starting* and *ending point* will be available in the new View.

Select the *ending point* from the pull-down list, then click the 'Go' button.



A new View will appear at the bottom of the list in the Portlets area.



The View can be modified like any other Selection in the Portlets area. Once you select it, you will see all the fields this virtual structure.

Below is a screenshot of the fields from a View in the 4D Meetings Demo database, where Meetings is the One table and Tasks is the Many table.

Fields are from Meetings and Tasks table

Properties			
Position	Real Name		DAX Alias
16	O: <input checked="" type="checkbox"/> I: <input checked="" type="checkbox"/>	[Meetings]Owner_ID	Owner_ID
17	O: <input checked="" type="checkbox"/> I: <input checked="" type="checkbox"/>	[Meetings]Version	Version
18	O: <input checked="" type="checkbox"/> I: <input checked="" type="checkbox"/>	[Tasks]Task_ID	Task_ID
19	O: <input checked="" type="checkbox"/> I: <input checked="" type="checkbox"/>	[Tasks]Name	Name
20	O: <input checked="" type="checkbox"/> I: <input checked="" type="checkbox"/>	[Tasks]Assigned_To	Assigned_To

CHAPTER 7: CALLBACKS

Callbacks are like form events in 4D. They allow for the developer to make certain actions to take place when a particular event occurs. A typical use of a callback would be to perform some type of data manipulation in a field. In the 4DAF the back-end callback can be triggered by two events: **On Load** and **On Data Change**. Note that the callback execution will happen at the field level only and it is supported from within the input layout only.

Developers should use the method **DAX_DevHook_CB_Install** to install all programmatic callbacks, and **DAX_DevHook_CB_EventFired** to handle all callback events.

DAX_DevHook_CB_Install is executed at database startup. Within this method, the developer calls the method **DAX_Dev_CB_Install** to enable an event for a specific field. Here are some installation examples:

```
` Method: DAX_DevHook_CB_Install
DAX_Dev_CB_Install(On Load;"Invoices";"ID") ` [Invoices]ID
DAX_Dev_CB_Install(On Load;"Customers";"Name") ` [Customers]Name
DAX_Dev_CB_Install(On Data Change;"Customers";"Prefix") ` [Customers]Prefix
```

To handle a callback, the developer must implement a trap inside the method named **DAX_DevHook_CB_EventFired**. This method will be triggered for every callback event. The developer is responsible for handling each event for each field via their own custom code. The following is the information that is available to the developer within this hook:

- Event that has fired
- Name of the selection that is being edited
- ID of the selection that is being edited
- Name of the field that is being edited
- ID of the field that is being edited
- Current value for the field that is being edited
- Record number that is being edited

This information can be obtained by calling the method **DAX_Dev_CB_GetInfo**. Here is an example:

```
` Method: DAX_DevHook_CB_EventFired
$eventID_1:=Num(DAX_Dev_CB_GetInfo("event id"))
$selectionName_t:=DAX_Dev_CB_GetInfo("selection name")
$selectionID_1:=Num(DAX_Dev_CB_GetInfo("selection id"))
$fieldID_1:=Num(DAX_Dev_CB_GetInfo("field id"))
$fieldValue_t:=DAX_Dev_CB_GetInfo("field value")
$recordID_1:=Num(DAX_Dev_CB_GetInfo("record id"))
```

This information can be used to determine what would be the appropriate action for the given callback. The following is example code based on the callbacks that we installed in the method ***DAX_DevHook_CB_Install***.

```
` Method: DAX_DevHook_CB_EventFired
Case of
: ($eventID_l=On Load)
  Case of
    : ($selectionName_t="Invoices")
      If ($recordID_l=New record) & ($fieldName_t="ID")
        ` Perform Data Manipulation here
      End if
    : ($selectionName_t="Customers")
      If ($recordID_l#New record) & ($fieldName_t="Name")
        ` Perform Data Manipulation here
      End if
  End case
  DAX_Dev_CB_SetStatus(1)

: ($eventID_l=On Data Change)
  Case of
    : ($selectionName_t="Customers")
      If ($fieldName_t="Prefix")
        ` Perform Data Manipulation here
      End if
  End case
  DAX_Dev_CB_SetStatus(1)

End case
```

The developer can manipulate the data in any of the fields in the input layout. This is done by calling the method ***DAX_Dev_CB_SetFieldValues***. This method takes two parameters: a pointer to a Text array containing field names and a pointer to a Text array containing field values. Here is an example based on the callbacks that we installed in the method ***DAX_DevHook_CB_Install***.

```
Case of
: ($eventID_l=On Load)
  Case of
    : ($selectionName_t="Invoices")
      If ($recordID_l=New record) & ($fieldName_t="ID")
        ARRAY TEXT($fieldNames_at;1)
        ARRAY TEXT($fieldValues_at;1)
        $fieldNames_at{1}:=$fieldName_t
        $fieldValues_at{1}:=Sequence number([Invoices])
        DAX_Dev_CB_SetFieldValues(->$fieldNames_at;->$fieldValues_at)
      End if
    : ($selectionName_t="Customers")
      If ($recordID_l#New record) & ($fieldName_t="Name")
        ARRAY TEXT($fieldNames_at;1)
        ARRAY TEXT($fieldValues_at;1)
        $fieldNames_at{1}:=$fieldName_t
        $fieldValues_at{1}:=Uppercase($fieldValue_t)
        DAX_Dev_CB_SetFieldValues(->$fieldNames_at;->$fieldValues_at)
      End if
  End case
  DAX_Dev_CB_SetStatus(1)

: ($eventID_l=On Data Change)
  Case of
    : ($selectionName_t="Customers")
      If ($fieldName_t="Prefix")
        ARRAY TEXT($fieldNames_at;1)
```

```

        ARRAY TEXT($fieldValues_at;1)
        $fieldNames_at{1}:="Sex"
        If ($fieldValue_t="Mr.")
            $fieldValues_at{1}:="Male"
        Else
            $fieldValues_at{1}:="Female"
        End if
        DAX_Dev_CB_SetFieldValues(->$fieldNames_at;->$fieldValues_at)
    End if
End case
DAX_Dev_CB_SetStatus(1)

```

End case

Callback status is set in the method ***DAX_Dev_CB_SetStatus***. This method tells the Web front-end whether the callback execution is successful or not. A failed callback is indicated by changing the background color (Yellow) of the field that triggered the callback.

DAX_Dev_CB_SetStatus(1) → Success

DAX_Dev_CB_SetStatus(0) → Fail

Along with the method ***DAX_Dev_CB_SetStatus***, the developer can add a meaningful message to the front-end by calling the method ***DAX_Dev_CB_SetMessage***. For example:

```

DAX_Dev_CB_SetStatus(0) ` Callback fail
DAX_Dev_CB_SetMessage("Invalid Zip Code") ` Give the reason why

```

CONCLUSION

This session introduces several key concepts employed by the 4D Ajax Framework. The developer was guided from the stage of installation all the way through to the development of their own custom pages. The hands-on demos should provide the developer with a solid foundation and the necessary tools for Ajax development with the 4D Ajax Framework.

DEMOS

Here are the demos for this session:

Demo 1: Install the Framework

- Set the port to 8080
- Connect to <http://localhost:8080> in your web browser
- Copy the contents of Webfolder from Session into your Webfolder

This demo is to get comfortable with the installation process. Instructions can be found in document “4DAF Install & Upgrade”.

Demo 2: Install Firebug to Firefox

- Firebug:
- www.getfirebug.com
- Firefox:
- www.getfirefox.com

This is a good demo to get everyone ready to go with Firebug. Firebug plays a big role in web development and it will play a big role in the demos that follow.

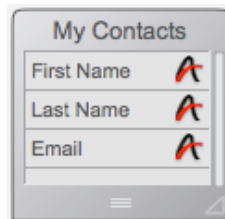
Demo 3: The Firebug Console

- Connect to localhost:8080/index.html in Firefox.
- Login to the framework with:
 - A valid user (Administrator, no password)
 - An unknown user
- Use Firebug to find out what XML information is returned.

This demo gets the developer more familiar with the Firebug interface. They will be introduced to the Console and will inspect XML responses via Firebug.

Demo 4: Create a DCS

- Follow the structure of this table:



My Contacts	
First Name	A
Last Name	A
Email	A

- Hints:
 - Dax_DevHook_DCS_ViewAdd: To create the arrays.
 - Dax_DevHook_DCS_SetSelection: To populate the array with data.
 - Use 'First Name, Last Name, and Email' fields from the [Contacts] table.

Here 4DAF data structures have been introduced. There are three types of 4DAF data structures: 1) Tables, 2) Developer Created Selections, and 3) Views. Developer Created Selections (DCS) are essentially arrays that are built by the developer and then populated with data from the existing tables in the structure. This example asks the developer to build their first DCS. This is the first time that the developer is introduced to a Developer Hook, and they will need to use two (2) to fully create a DCS.

Demo 5: Create a View

- Create a View using the tables [Contacts] and [Company].
- Hint
 - Start from the 'One' table.

- Extra Credit:
 - Display the View as a Data Tree

In this demo the developer is asked to create the last type of 4DAF data structure, the View. This is much easier than the previous example since Views can be created via the Client. DCS's, on the other hand, are created programmatically via (2) Developer Hooks.

Demo 6a: Login

- Use Developer Hook `Dax_DevHook_Login` to have a user named 'Guest' with password '4d' log in to the page.
- 2) Create a new 4D user. Login with that user.

This demo introduces the developers to their first 4DAF JavaScript method, `dax_login`. Here, they are experimenting with the different ways they can validate a user (4D User and Groups, or their own password using Developer Hook `Dax_DevHook_Login`).

Demo 6b: Login advanced

- Do not call `dax_login()` during the `OnClick` of the submit button. Instead, call your own function which will make the `dax_login()` call.
- Extra Credit:
 - Hide the input form on successful login.

By the request of many developers, here is an example where log in is not directly hard-coded onto the page. This example takes input from the user, and then calls the `dax_login` function after the click of the submit button.

Demo 7: Embed Data Grid

- Embed it within the "GridDiv" `<div>`
- Explore other possibilities:
 - Lock 1 left column.
 - Float the grid in a window instead of embedding it on the page.
 - Create multiple header rows.
 - Disable the Control Column.
- Hint: Enter this code within `dax_loginSuccess`

With the log in procedure taken care of, the developer should have enough know-how to handle embedding a Data Grid to the HTML page (with the help of the Data Grid API at hand). Here they explore the API and see how easily it is (2 lines of JavaScript code) to embed one of their own tables to a custom page.

Demo 8: Preset Queries

- Use the Query Manager to create query tabs for the Contacts table.
 - Make tabs so that the First Name field is organized in alphabetical order.
- Embed the Contacts table on the page within the 'GridDiv' `<div>`.
- Try displaying the Preset Queries in the Sidebar and then as Tabs.

Preset Queries are a great way to present and organize information for the end user once they see a Data Grid. Here, the developer defines the Preset Queries in the 4DAF Client and then uses the Data Grid API to display the queries as Tabs or in the Sidebar.

Demo 9: Drag Drop Grid

- Modify 'Exercise_9.html' so that:
 - The background color for CSS class 'dragdrop_style' is set to green.
 - All rows and columns are draggable (instead of just the 2nd column).
- Extra Credit:
 - Display an Alert message when a cell is dropped into a target area. Make sure the Alert displays the row, column, and value of the original cell and its target destination

This example may be a bit robust for beginners, but it's simple enough to for them to analyze should they need more time outside the session. They start out with a working Drag and Drop example and are asked to make slight modifications.

CREDITS

Some material can be credited to:

- Joe Resuello's 4D Summit 2007 Conference Session "*Ajax Primer*"
- Add Komoncharoensiri's 4D Summit 2007 Conference Session "*Developer Hooks for 4D Ajax Framework*"
- *The 4D Ajax Framework Admin Ref*