4D Ajax Framework

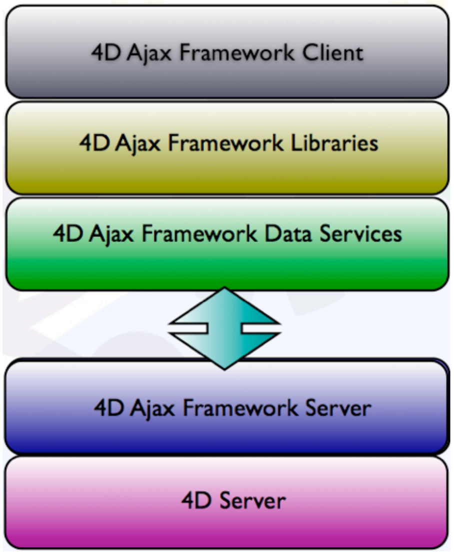
Developer’s Guide
Windows/Mac OS

Overview

The 4D Ajax Framework (4DAF) allows 4D developers to add web-browsing integration into their applications. Using Ajax technology, 4D developers can offer the end-user a rich web 2.0 interface that can browse, add, modify and delete records from the underlying database. 4D Ajax Framework includes a component and a full Javascript framework that is available for integration into existing HTML pages. Experienced 4D web developers will be able to use the DAX API (Javascript framework) to quickly add 4D Ajax Framework objects into their existing HTML/Javascript pages.

There are three main ways of handling the 4DAF:

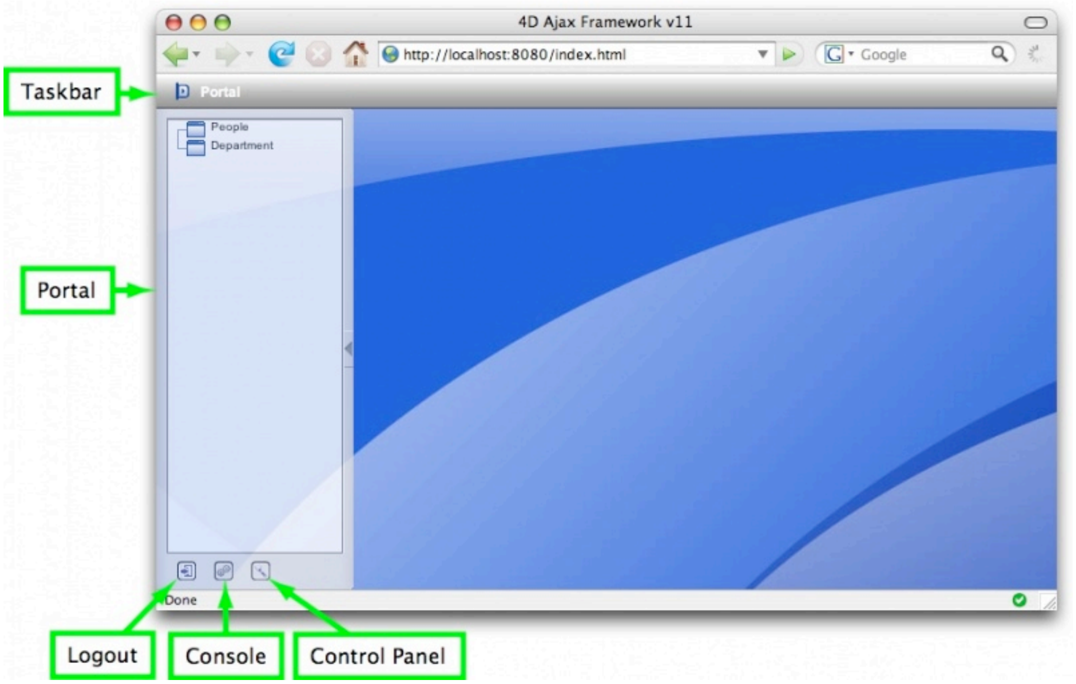
- The 4DAF Client
- The 4DAF Libraries
- The 4DAF Bridge



The 4DAF Client

The 4DAF Client is the top-most layer of the 4DAF, which means it a Rich Internet Application (RIA) right out of the box. Once the 4DAF is installed to your database, the 4DAF Client is automatically available by connecting to your webserver (for example, http://<webserverURL>/index.html).

For more information on the 4DAF Client see document "4DAF v11 Admin".

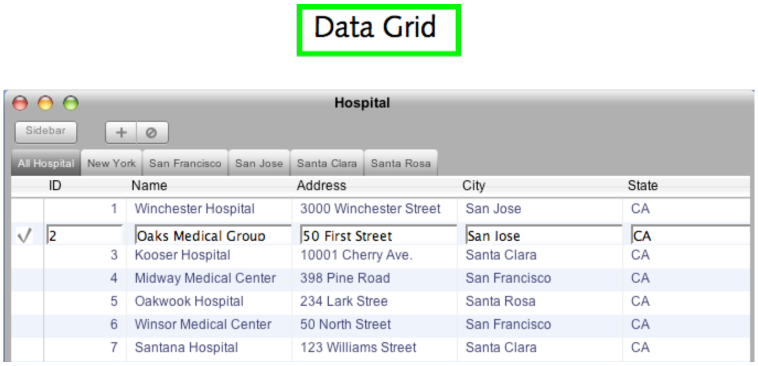


A look at the 4DAF Client

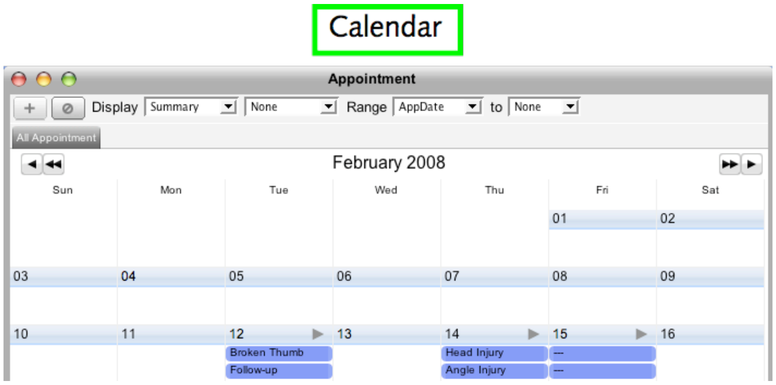
For developers who would like more customization with the 4DAF see sections “*The 4DAF Libraries”* and *“The 4DAF Bridge”* below.

The 4DAF Libraries

4D developers who want to create their own web applications supported by the 4DAF read further in this document. At this level 4D Developers can create their own web pages and embed their data structures using the 4DAF objects (examples seen below).



	ID	Name	Address	City	State
	1	Winchester Hospital	3000 Winchester Street	San Jose	CA
✓	2	Oaks Medical Group	50 First Street	San Jose	CA
	3	Kooser Hospital	10001 Cherry Ave.	Santa Clara	CA
	4	Midway Medical Center	398 Pine Road	San Francisco	CA
	5	Oakwook Hospital	234 Lark Stree	Santa Rosa	CA
	6	Winsor Medical Center	50 North Street	San Francisco	CA
	7	Santana Hospital	123 Williams Street	Santa Clara	CA



Display Summary None Range AppDate to None

February 2008

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					01	02
03	04	05	06	07	08	09
10	11	12 ▶ Broken Thumb Follow-up	13	14 ▶ Head Injury Angle Injury	15 ▶ — —	16

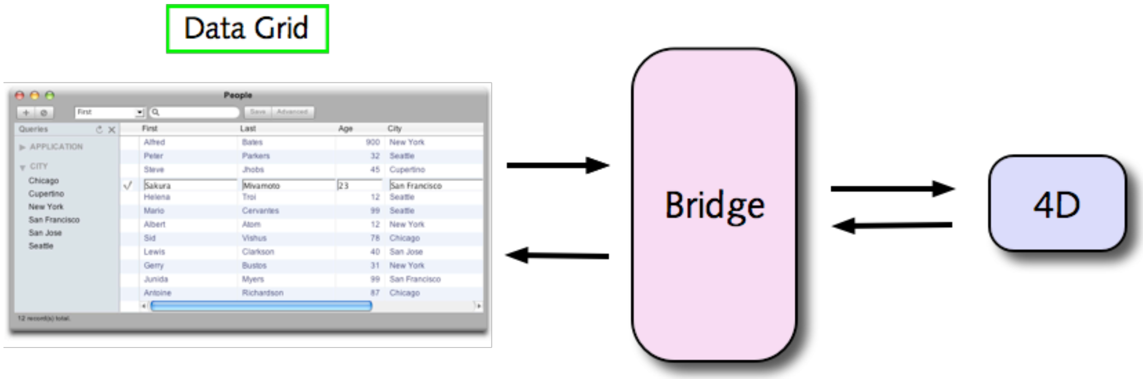
The 4DAF Libraries provide the 4D Developer with access to each object’s API thus granting much more customization potential. It is also at this level that the 4D Developer is allowed to take full advantage of the 4DAF’s Developer Hooks.

To get started on creating your own custom applications see chapter “My First Grid”. *It covers all you need to know to get jumpstarted on creating your own web applications with the 4DAF.*

The 4DAF Bridge

The lowest frontend layer of the 4DAF is the Bridge. It is at this level that 4D Developers can forego the 4DAF objects and libraries provided by the framework. Here, only the raw data from the 4D server is of consequence. Use the 4DAF at this level when you are creating your own interface objects or if you are opting for other user interface elements from other Ajax frameworks such as Dojo.

The primary chapter for this layer of development is “The 4DAF Bridge” *found later in this document.*



	First	Last	Age	City
	Alfred	Bates	900	New York
	Peter	Parkers	32	Seattle
	Steve	Jhobs	45	Cupertino
✓	Sakura	Mivamoto	23	San Francisco
	Helena	Troi	12	Seattle
	Mario	Cervantes	99	Seattle
	Albert	Atom	12	New York
	Sid	Vishus	78	Chicago
	Lewis	Clarkson	40	San Jose
	Gerry	Bustos	31	New York
	Junida	Myers	99	San Francisco
	Antoine	Richardson	87	Chicago

My First Grid

Welcome to chapter “My First Grid”, the first and only chapter you will need to get your feet wet with custom web page development using the 4D Ajax Framework. This chapter provides the steps necessary embed a live 4DAF Data Grid object into an HTML page.

Installation

The first step is to install the 4D Ajax Framework (4DAF) into your database.

> *For installation instructions please refer to the document “4DAF v11 Install & Upgrade”.*

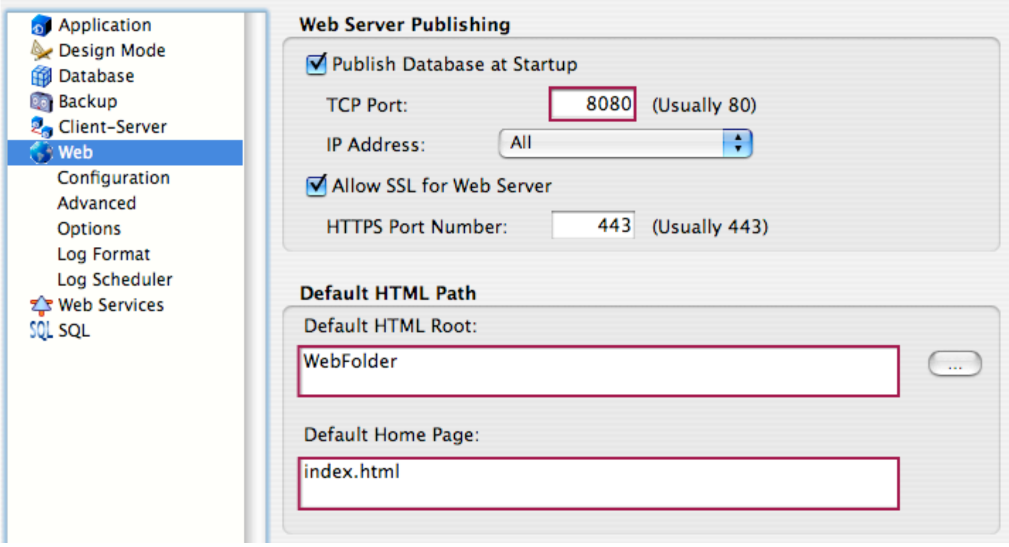
What’s So Different After Installation?

So you have your 4D database with the 4D Ajax Framework (4DAF) component installed. What does that mean exactly? Launch the 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.



Web Server Publishing

☑ Publish Database at Startup

TCP Port: 8080 (Usually 80)

IP Address: All

☑ Allow SSL for Web Server

HTTPS Port Number: 443 (Usually 443)

Default HTML Path

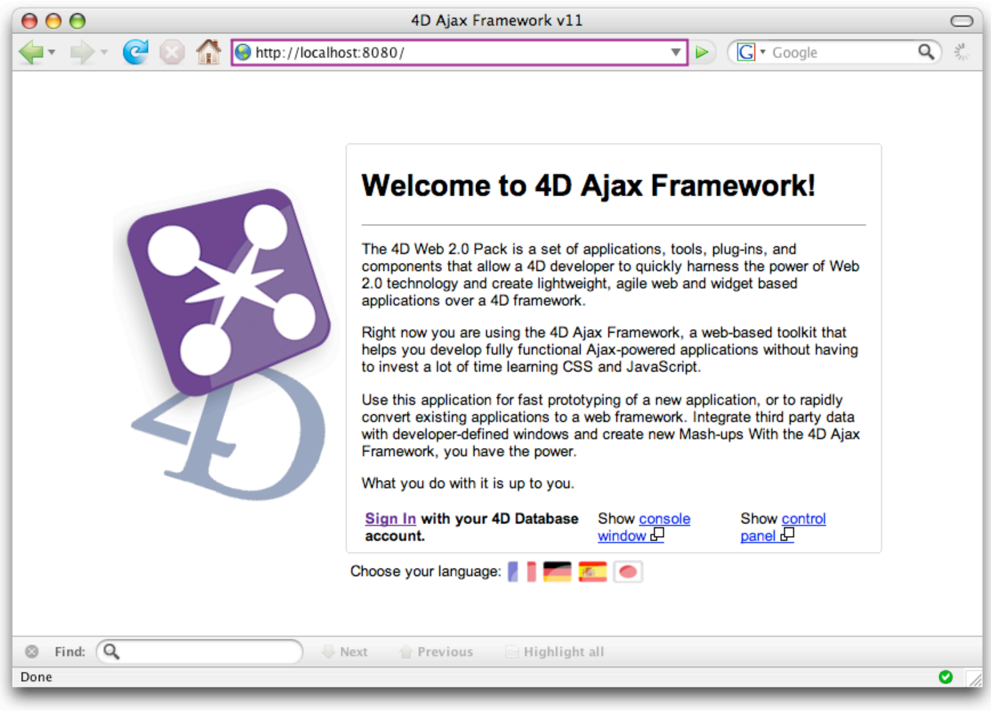
Default HTML Root: WebFolder

Default Home Page: index.html

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

On the same machine that is running the database, open up your web browser and connect to http://localhost:8080.

> *‘Localhost’ means the same machine you are currently using. ‘8080’ is the port that the web server is published on.*



Welcome to 4D Ajax Framework!

The 4D Web 2.0 Pack is a set of applications, tools, plug-ins, and components that allow a 4D developer to quickly harness the power of Web 2.0 technology and create lightweight, agile web and widget based applications over a 4D framework.

Right now you are using the 4D Ajax Framework, a web-based toolkit that helps you develop fully functional Ajax-powered applications without having to invest a lot of time learning CSS and JavaScript.

Use this application for fast prototyping of a new application, or to rapidly convert existing applications to a web framework. Integrate third party data with developer-defined windows and create new Mash-ups With the 4D Ajax Framework, you have the power.

What you do with it is up to you.

Sign In with your 4D Database account. Show console window Show control panel

Choose your language:

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.

If you see the above page then congratulations! 4D is running as a web server and the 4DAF is successfully installed.

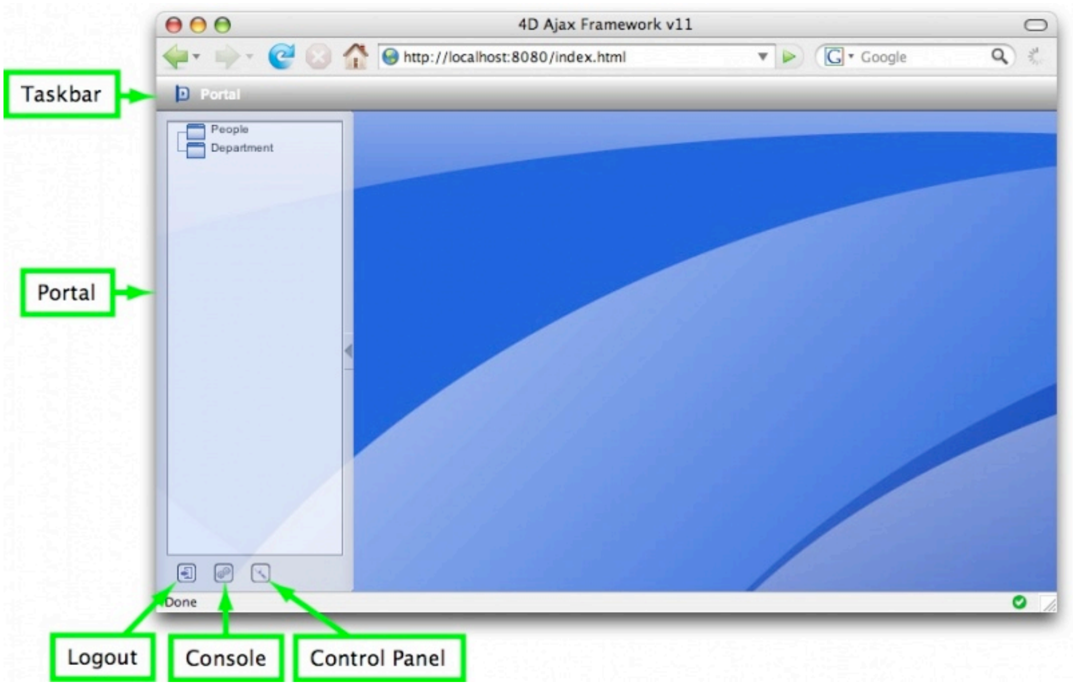
Web Folder files

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

Navigate to the 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 stylesheets, 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

As mentioned in chapter "*Overview*", you can use the 4DAF Client by connecting to http://localhost:8080/index.html in your web browser. The hit the Sign In link and enter username 'Administrator' with no password.



For more information on the 4DAF Client see document "4DAF v11 Admin".

Of course, this chapter focuses on creating your own web application by embedding your first Data Grid object onto the page. So, the next step is to create your HTML page and to place it into your Webfolder.

Creating your own Web Application

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.

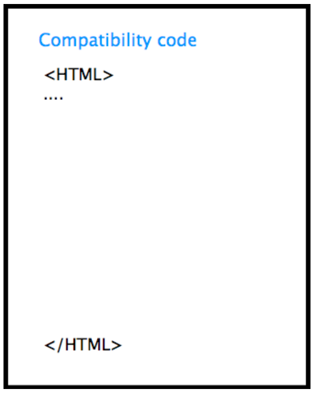
Let's take a closer look at the Template Page from top to bottom.

The Template Page

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.

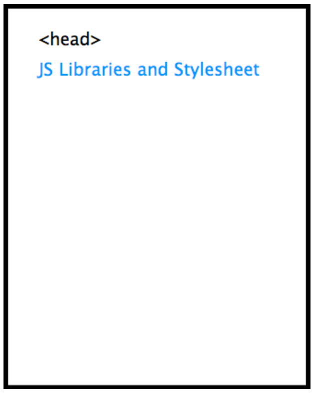
Note: *This code appears before the <HTML> tag.*

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" />
```



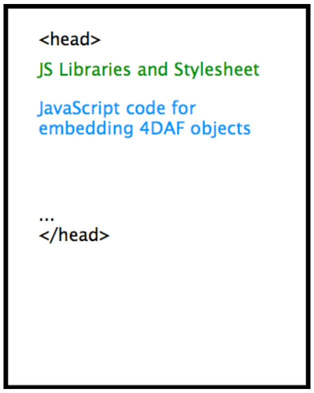
Note: *This code appears within the top of the <head> tag. Since the browser reads this information in a top to bottom manner, it is best to include these libraries early before any actual JavaScript code specific to the 4DAF is included soon below.*

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).


<head>
JS Libraries and Stylesheet
JavaScript code for embedding 4DAF objects
...
</head>

Login

In this Template page the 4DAF’s *Login* function is called within the <body> element tag.

```
<body onload="Login("Guest", "")">
```

Here, user ‘Guest’ is entered with no password. The same *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.

4D Ajax Framework Concepts

Although 4D Ajax Framework will provide full functionality without any modifications by the developer, many 4D developers will want to take advantage of some of the more powerful advanced options that are available.

Localization

Ajax Framework uses a localization technique that allows developers to deploy their applications in most languages. All messages that are displayed in the application interface and on the web front end are retrieved from XML files located in the \Extras\Support folder. By default these messages appear in English.

Localized_strings.xml

This file contains the progress, alert and confirm messages that are displayed in the 4D interface. Any time Ajax Framework needs to display a message, Ajax Framework will find an appropriate translation based on the value that was set in the method named DAX_DevHook_Preferences (set a language: DAX_Dev_SetPreference (1;"en")).

The developer can add new language elements by editing the XML file. For example, if you want the " Ajax Framework Initializing" startup message to appear in Spanish, simply add a <es> element as a child of the <DAXInitializing> element with the message in Spanish as the element value.

en_errors.xml

This file contains all of the error messages that are sent to the front end. By default, the name of the file matches the language value set in the variable <>DAX_Language_t ("en"). You can create an error file for another language by duplicating the en_errors.xml file and changing the contents of the <message> nodes.

To create an error file for a different language:

1. Duplicate the en_errors.xml file.
2. Rename the duplicate file to begin with the language that you set in DAX_DevHook_Preferences (fr_errors.xml for example).
3. Go through the new file and translate the contents of all of the <message> nodes to the language you are adding.

Developer Hooks

Developer Hooks allow the 4D developer to override certain default Ajax Framework behaviors and to trap various events. Use of the Developer Hooks is optional. The only editable methods within the Ajax Framework component are named DAX_DevHook.... The only callable methods in the Ajax Framework component are named DAX_Dev_.... Use of the various available developer hooks is discussed later in this document.

Views or Selections

A view or selection is a window that displays data or other content to the user. The simplest example of this is the window that opens and displays the list of records when a table name is clicked on in the Portals sidebar. You can create various custom views within the Ajax Framework component to be added to the list of tables that appears by default.

Developer Created Selections

Developer Created Selections (DCS) are views that are completely array driven. The developer defines the view through 4D code and provides arrays with the data that is used in the view. See Creating DCS Views later in this document for more information.

Developer Defined Windows

Developer Defined Windows (DDW) are views that can contain any HTML content. They are not generally for displaying a list of data but instead available to display a complete web page within the front-end user interface. A DDW can open a link to an external site such as http://www.4d.com or call back to the developer's existing web code with a link such as /4dcgi/mymethod. A DDW can also be sent an HTML blob directly. This allows the developer to create a web page in memory and deliver it to the front-end.

Callbacks

Callbacks act similar to 4D's form events. A callback can be assigned to a field within the input form and call 4D code when a particular event occurs (currently On Load and On Data Change). For example, an email address can be validated when a user tabs out of the email address field and the user can be given immediate feedback if the address is determined to be invalid. See Creating Callbacks later in this document.

Shared Methods

When using the 4D Ajax Framework v11 with 4D v11, methods that are called must be shared between the host and component. This is a common situation when methods are used in DDWs, Preset Queries or other callbacks. The method must be shared so that it can be correctly executed in 4D v11 SQL.

For example, let's say you are creating a DDW that will call a method named "MapIt." Make sure that under "Method Properties" the option for "Shared by components and host database" is selected.

Group Names

There is currently a known issue regarding Group Names that have spaces " " in between words. This is will generate an error when the framework attempts to generate the XML files (such as when preferences are saved on shutdown). This limitation is being worked on and will be corrected in a future release.

DAX_Dev_WaitForInitialization

4D Ajax Framework v11 introduces a method that delays the current process until the 4D Ajax Framework initialization process (process name = DAX_Initialize) is finished. This method should be called right after DAX_Dev_Initialize.

For every one minute that the initialization process is still running, the user will be prompt to confirm if he or she wish to continue to wait.

```
Passed:
$1        BOOL    True for displaying wait confirmation every min.
                  Omitted or False for Suppressing the confirmation.
```

Retrieve User Information

The 4D Ajax Framework v11 introduces two new methods that allow you retrieve more information about the current user. These methods are:

```
Dax_Dev_GetCurrentSessionID
Dax_Dev_GetCurrentUser
```

DAX_Dev_GetCurrentSessionID

This method returns the Session ID of the current 4D Ajax Framework connection.

```
Returned:
$0         TEXT   Current Session ID
```

DAX_Dev_GetCurrentUser

This method returns the name of the current 4D Ajax Framework Web user.

```
Returned:
$0         TEXT    Current Username
```

Setting Preferences

4D Ajax Framework has several global preferences that you can adjust to your specific needs. To change the default settings, open the project method DAX_DevHook_Preferences and edit the variable values defined below.

Language and Interface Settings

There are two 4D Ajax Framework language settings. The first setting determines what language 4D Ajax Framework should use when displaying information dialogs in the 4D interface during startup and shutdown. The second determines what language 4D Ajax Framework uses when displaying information to the front-end (web browser) user interface.

The language used in the 4D interface is determined by the value set through DAX_Dev_SetPreference. This value must correspond to a valid language in the Localized_strings.xml file (see Ajax Framework Concepts above). The default value is “en” for English.

Example

```
DAX_Dev_SetPreference(1;"en") ` set the 4D interface language to English
```

The language to use for the Ajax Framework front-end can be installed on a group by group basis. To install a language for a particular group, make a call to the project method DAX_Dev_InstallLanguage inside the DAX_DevHook_Preferences method. By default the front-end will use English for all groups.

Example

```
DAX_Dev_InstallLanguage ("Admins";"en") ` set the language for Admins to
English
DAX_Dev_InstallLanguage ("Group A";"fr") ` set the language for Group A to
French
```

To suppress the startup and shutdown status messages in 4D, call the method DAX_Dev_SetPreference(3,Boolean) in the DAX_DevHook_Preferences. Critical error alerts will still be displayed.

Example

```
DAX_Dev_SetPreference (3;"False") ` Turn off messages
DAX_Dev_SetPreference (3;"True") ` Turn on messages
```

General Settings

There are several general settings 4D Ajax Framework uses in conjunction with its various features.

User's sessions automatically expire after a given period of inactivity. By default, sessions will expire after one hour. Set the time out value to ?00:00:00? to have sessions never timeout.

Example

```
DAX_Dev_SetPreference (4;"01:00:00") ` expire sessions after 1 hour of
inactivity
```

The 4D Ajax Framework record locking system automatically releases a lock after a given period of inactivity. By default a lock will expire after five minutes. Set the lock value to ?00:00:00? to have locks never timeout .

Example

```
DAX_Dev_SetPreference (5;"01:00:00") ` expire locks after 5 minutes of
inactivity
```

When a new view is created from the Admin area of the 4D Ajax Framework front-end the related tables are displayed so data may be combined from multiple tables. By default this value is set to eight. It is recommended that this value not be increased.

Example

```
DAX_Dev_SetPreference (7;"8")
```

4D Ajax Framework has a daemon process that runs continuously in the background. This process periodically writes 4D Ajax Framework preferences to disk and cleans up memory being used for user sessions that have expired. By default this process wakes up and runs every thirty minutes.

Example

```
DAX_Dev_SetPreference (6;"00:30:00")
```

DAX_DevHook_Preferences

4D Ajax Framework has some of the default setting that can be modified by the developers. The modification can be done from a developer hooks method named DAX_DevHook_Preferences. This method is used to set several preferences:

```
1   Language
2   Encoding
3   Display Message
4   Lock Timeout
5   Connection Timeout
6   Daemon Delay
7   Relation Depth
8   Wildcard mode (v11 SQL component only)
```

Implementation

In the version 1.0 and 1.1, the 4DAF preferences are set with through an interprocess variable. This technique are still valid up to 4D 2004.x. We recommend that you begin using the new preference setting approach in 1.2. In version 1.2, a new utility method named DAX_Dev_SetPreference is added to help developer set the global preferences more efficiently.

Method: DAX_Dev_SetPreference

```
Call Syntax: DAX_Dev_SetPreference (Preference ID;Preference Value)

   Passed:
   $1        LONGINT   Preference ID of 1 to 7 (8 for v11 SQL
                       Component only)
   $2        TEXT      Preference Value

   1      Language "en, fr, de, etc."
   2      Encoding "iso-8859-1, Shift_JIS, etc."
   3      Display Message "True, False"
   4      Lock Timeout "00:05:00"
   5      Connection Timeout "01:00:00
   6      Daemon Delay "00:30:00
   7      Relation Depth "8"
   8      Wildcard mode (v11 SQL component only)
          "True" to consider @ as a character (not a wildcard) for query


Here are some of the example calls:
```

```
  ` Display Startup and shutdown status messages?
DAX_Dev_SetPreference (3;"True")  `◊DAX_DisplayMessages_b:=True
  ` period of inactivity allowed before a record lock times out
  ` locks will expire with no activity in the given time frame
  ` set to ?00:00:00? to never timeout
DAX_Dev_SetPreference (4;"00:05:00")  `◊DAX_LOCK_TIMEOUT_m:=†00:05:00†
  ` period of inactivity allowed for login
  ` Sessions will expire with no activity in the given time frame
  ` set to ?00:00:00? to never timeout
DAX_Dev_SetPreference (5;"01:00:00")  `◊DAX_CONN_TIMEOUT_m:=†01:00:00†
```

Dax_Dev_IsDaxCommand

For those already using 4D as a web server, 4D Ajax Framework version 1.2 introduced a new command that can be used to conveniently replace a test for 'dax' in the requested URL within the On Web Connection method.

Previously your On Web Connection code might look like this:

```
Case of
  : (($url_t="DAX@") | ($url_t="/DAX@"))
    DAX_Dev_OnWebConn
($url_t;$header_t;$clientIP_t;$serverIP_t;$user_t;$pass_t)
Else
  ` handle non-DAX requests in other cases or here
  `---------------------------------------------------
End case
```

With this new command you can now make a call like this:

```
Case of
  : (DAX_Dev_IsDAXCommand ($url_t))
    DAX_Dev_OnWebConn
($url_t;$header_t;$clientIP_t;$serverIP_t;$user_t;$pass_t)
Else
  ` handle non-DAX requests in other cases or here
  `---------------------------------------------------
End case
```

Managing User Authentication and Sessions

By default 4D Ajax Framework will use the built in 4D Users and Groups system for users login and group management. 4D Ajax Framework also has its own internal session management system. You can override the DAX login system, the 4D Ajax Framework login system and session management, or just let 4D Ajax Framework handle both for you. *You cannot override session management unless you are also overriding login.*

If you already have your own login system in place, for instance a Users table, you can hook into the 4D Ajax Framework implementation and authorize users yourself. If you also have your own session management system you can hook into the 4D Ajax Framework session implementation and validate the sessions yourself as well.

Authentication Developer Hooks

DAX_DevHook_Login

DAX_DevHook_Login is used by the Developer to override the 4D Ajax Framework login system. By default 4D Ajax Framework uses the 4D Users and Groups system to authenticate users when they login using the 4D Ajax Framework login object. If you have your own login system you can hook into the login process using this method and take over user validation. If you are using the 4D Ajax Framework login system (4D Users and Groups) you do not need to modify this method.

DAX_DevHook_Login receives six text parameters. These six parameters are identical to the ones that are received in the On Web Connection database method. Therefore you can call the same authentication code you would normally call from On Web Connection.

```
Passed:
$1        TEXT       URL that has been requested by the client
$2        TEXT       Header text from the request
$3        TEXT       Client IP Address
$4        TEXT       Server IP Address
$5        TEXT       User Name (from HTTP Authorization)
$6        TEXT       Password (from HTTP Authorization)

Returned:
$0         BOOLEAN    Did this user successfully log in?
```

Using the system you currently have in place you validate the user name and password. This most likely involves querying a table for the user name and password to authenticate the user. If you are using the 4D Ajax Framework login object on the web page you can retrieve the user name and password using the following calls:

```
$userName_t:=DAX_Dev_GetWebVar ("username")
$password_t:=DAX_Dev_GetWebVar ("password")
```

If the user has successfully logged in, you return True from this method and, if their login fails, you should return False.

When you are using the DAX_DevHook_Login method to validate users yourself, you must pass 4D Ajax Framework some information about the user.

Upon Successful Login

1. You must call DAX_Dev_Session_UserName and pass the user's user name.
2. If the user should have Admin privileges, call DAX_Dev_Session_Admin and pass True; otherwise, pass False or simply don't call the method.
   A user with Admin privileges can make changes to what tables and fields are displayed using the Admin object on the web page. Generally very few users should have this ability.
3. If you are also planning on handling sessions with your own session management system, you will need to provide the session id you will be using for this user by calling DAX_Dev_Session_SessionID and passing the session id you are using for this user.

Upon Failed Login

If the login is not successful, you must pass an error code to Ajax Framework. If you return False from this method and do not pass an error code, Ajax Framework will attempt to log the user in. To set a login error code call DAX_Dev_Session_Error and pass an appropriate error code.

Login error codes include:

```
LoginPasswordMissing
LoginUsernameMissing
UserNotFound
InvalidUsernameOrPassword
```

Successful Example:

```
DAX_Dev_Session_UserName($userName_t)
DAX_Dev_Session_Admin($isAdmin_b)
DAX_Dev_Session_SessionID($sessionID_t)
$0:=True
```

Failed Example:

```
DAX_Session_Error ("LoginPasswordMissing")
$0:=False
```

DAX_DevHook_Login - Portal

We need the current user name and password. We can retrieve it from those commands:

```
$username_t:=DAX_Util_GetWebVar ("username")
$password_t:=DAX_Util_GetWebVar ("password")
```

The current user may be an admin. We need to check that out:

```
Gen4D_PreferenceGet ("AdminPass")  `sets CGI4D_t_adminPass
Gen4D_PreferenceGet ("AdminUser")  `sets CGI4D_t_adminUser
$admin_b:=((CGI4D_t_adminPass=$password_t) &
(CGI4D_t_adminUser=$username_t))
```

We can now check if the user can login:

```
QUERY([CGI4D_Portal_Users];[CGI4D_Portal_Users]User_Name=$username_t;*)
QUERY([CGI4D_Portal_Users]; & [CGI4D_Portal_Users]Password=$password_t)
If (Records in selection([CGI4D_Portal_Users])=1)
   $loggedIn_b:=True
End if
```

DAX_DevHook_SessionValidate

DAX_DevHook_SessionValidate is used by the developer to override the default 4D Ajax Framework session management system. By default 4D Ajax Framework uses an internal session management system to authenticate users when they make various 4D Ajax Framework requests. If you have your own session management system, using a Sessions table for example, you can hook into the session process using this method and take over session validation. You must have first modified DAX_DevHook_Login and provided a session id before using this developer hook. If you are using the default 4D Ajax Framework session management system, you do not need to modify this method.

DAX_DevHook_SessionManagement receives six text parameters. These six parameters are identical to the ones that are received in the On Web Connection database method. Therefore you can call the same validation code you would normally call from On Web Connection.

```
Passed:
$1        TEXT    URL that has been requested by the client
$2        TEXT    Header text from the request
$3        TEXT    Client IP Address
$4        TEXT    Server IP Address
$5        TEXT    User Name (from HTTP Authorization)
$6        TEXT    Password (from HTTP Authorization)

Returned:
$0         BOOLEAN    Is this a valid user session?
```

Using whatever system you currently have in place, you validate the user's current session. This most likely involves querying a table for the session id. 4D Ajax Framework supplies the session id with every 4D Ajax Framework call. You can retrieve the supplied session id with the following call:

```
$sessionid_t:=DAX_Dev_GetWebVar ("sessionid")
```

If the session is valid, return True from this method. If the session is not valid, return False.

When you are using the DAX_DevHook_SessionValidate method to validate users yourself, you must pass 4D Ajax Framework some information about the user and the session.

Upon Valid Session

1. You must call DAX_Dev_Session_UserName and pass the user's user name.
2. If the user should have Admin privileges, call DAX_Dev_Session_Admin and pass True; otherwise, pass False or simply don't call the method.
   A user with Admin privileges can make changes to what tables and fields are displayed using the Admin object on the web page. Generally very few users should have this ability.
3. You must call DAX_Dev_Session_SessionID and pass the session id you are using for this user.

Upon Failed Login

If the session is not valid you must pass an error code to 4D Ajax Framework. If you return False from this method and do not pass an error code, 4D Ajax Framework will attempt to validate the session itself. To set a session error code call DAX_Dev_Session_Error and pass an appropriate error code.

Session error codes include:
`SessionExpired`

```
IllegalAccessPoint
InvalidSessionId
```

Successful Example:

```
DAX_Dev_Session_UserName($userName_t)
DAX_Dev_Session_Admin($isAdmin_b)
DAX_Dev_Session_SessionID($sessionID_t)
$0:=True
```

Failed Example:

```
DAX_Session_Error ("InvalidSessionId")
$0:=False
```

DAX_DevHook_SessionValidate - Portal

When logged as Admin, a second request will be performed to request the table list We have a session ID but we do not know whi is the current user. We still need to tell if the user is the Administrator or not.

```
Gen4D_PreferenceGet ("AdminPass")  `sets CGI4D_t_adminPass
Gen4D_PreferenceGet ("AdminUser")  `sets CGI4D_t_adminUser
$admin_b:=((CGI4D_t_adminPass=$pass_t) & (CGI4D_t_adminUser=$user_t))
```

We need a session ID. If we do not have any, we would need to create one:

```
$sessionID_t:=DAX_Session_New
($userName_t;$HTTPUserAgent_t;$HTTPAddress_t;$adminFlag_b)
```

DAX_DevHook_GetGroupsList

DAX_DevHook_GetGroupsList is provided for the Developer to override the default 4D Ajax Framework Groups system. By default 4D Ajax Framework uses the 4D Users and Groups system to authenticate users when they login using the 4D Ajax Framework login object. Groups are then used to determine what to display to particular users. For instance, users in Group A can be shown and allowed to edit the table 1 Contacts, while users in Group B will never see the table. If you have your own users and groups system in place, you can override the 4D Ajax Framework group system here. This method should be used in tandem with DAX_DevHook_UserInGroup. If you use DAX_DevHook_Login to override the login system and do not implement your own groups, all of the users of your system will be assigned to the "AllUsers" group and have equal access. You must return True from this method if you implement your own solution here.

DAX_DevHook_GetGroupsList receives two Pointer parameters. These two pointers point to arrays that the developer should populate with their group names and group numbers.

```
Passed:
$1       POINTER    Pointer to a Text Array to receive the group names
$2       POINTER    Pointer to a Longint Array to receive the group numbers


Returned:
$0        BOOLEAN  Flag indicating that you are using your own groups system
```

Example:

```
READ ONLY([User_Groups])
```

```
All Records([User_Groups])
SELECTION TO ARRAY([User_Groups]ID;$groupIDs_p>;
[User_Groups]Group_Name;$groupNames_p->)
UNLOAD RECORD([User_Groups])

$0:=True ` Developer is handling Users and Groups
```

DAX_DevHook_UserInGroup

DAX_DevHook_UserInGroup is provided for the Developer to override the default 4D Ajax Framework Groups system. By default 4D Ajax Framework uses the 4D Users and Groups system to authenticate users when they login using the 4D Ajax Framework Login object. Groups are then used to determine what to display to particular users. For instance, users in Group A can be shown and allowed to edit [Table 1] while users in Group B will never see the table. If you have your own users and groups system in place you can override the 4D Ajax Framework group system here. This should be used in tandem with DAX_DevHook_GetGroupsList. ***If you use DAX_DevHook_Login to override the login system and do not implement your own groups all of the users of your system will be assigned to the "AllUsers" group and have equal access***. You must return True in $0 if you implement your own solution here.

Parameters

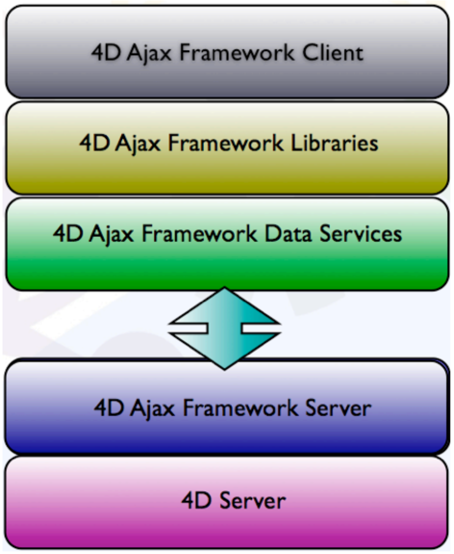
DAX_DevHook_UserInGroup receives 2 Text parameters and 1 Pointer parameter. The 2 text parameters contain the user name and the group name. The pointer points to a boolean variable that you set to TRUE if the passed user is in the passed group and FALSE if the user is not.

```
Passed:
$1            TEXT           User name we are checking
$2            TEXT           Group name we are checking
$3            POINTER        Pointer to a Boolean: TRUE this user
                             is in this group, FALSE otherwise

Returned:
$0         BOOLEAN  Flag indicating that you are using your own groups system
```

Use the passed parameters to check if the user is in the group. For example, let's assume you have a table named [User_Groups] that contains the fields [User_Groups]ID and [User_Groups]Group_Name and another table named [Users] that has a field that links to [User_Groups] named [Users]User_Groups_Link and a field [Users]User_Name. Your code might look like the following:

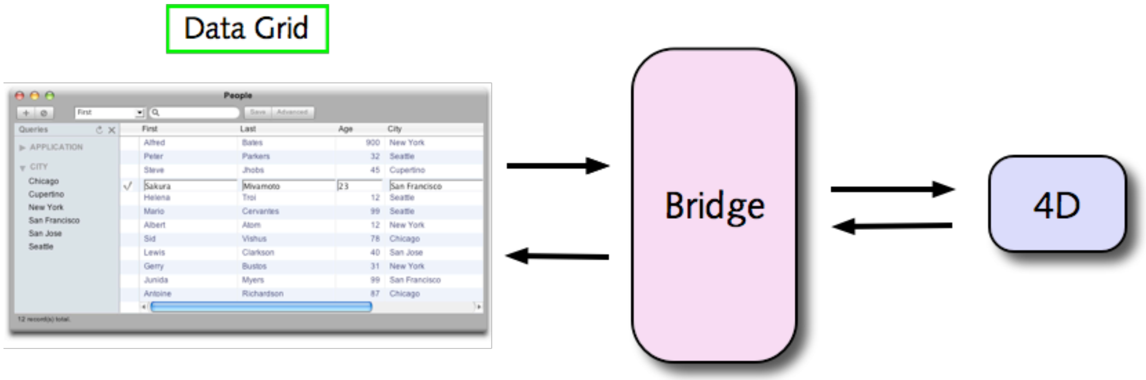
```
$userName_t:=$1
$groupName_t:=$2
$userInGroup_p:=$3

READ ONLY([Users])
READ ONLY([User_Groups])
QUERY([Users];[Users]User_Name=$userName_t)
RELATE ONE([Users]User_Groups_Link)
$userInGroup_p->:=([User_Groups]Group_Name=$groupName_t)
UNLOAD RECORD([Users])
UNLOAD RECORD([User_Groups])

$0:=True ` Developer is handling Users and Groups
```

The 4DAF Bridge



The 4D Ajax Framework (4DAF) comes in many layers, which allow for different levels of customization. At the top most level is the Client, which readily publishes a 4D database on the web out of the box. No JavaScript is needed at this level. Below that layer are the 4D Ajax Framework libraries. Here, *minimal* JavaScript is necessary to create HTML pages and to embed 4DAF objects onto a page.

The next layer is the Bridge where *much* JavaScript is necessary. Here, the 4D Developer essentially is given access to the data after it has arrived from 4D but before it is loaded in an object such as a Data Grid. A common scenario of using the Bridge is when developers are building their own custom forms, in which case the data from 4D is all that they require.



	First	Last	Age	City
	Alfred	Bates	900	New York
	Peter	Parkers	32	Seattle
	Steve	Jhobs	45	Cupertino
✓	Sakura	Mivamoto	23	San Francisco
	Helena	Troi	12	Seattle
	Mario	Cervantes	99	Seattle
	Albert	Atom	12	New York
	Sid	Vishus	78	Chicago
	Lewis	Clarkson	40	San Jose
	Gerry	Bustos	31	New York
	Junida	Myers	99	San Francisco
	Antoine	Richardson	87	Chicago

12 record(s) total.

The Bridge does the job of generating requests to 4D, receiving XML responses, parsing those responses, and then formatting them into easily readable JavaScript form for 4DAF Objects or custom developer objects alike. The Bridge also has the task of maintaining meta-data, session management, and user information.

For Desktop and iPhone

Desktop iPhone

All functionality of the Bridge is supported on the desktop, while a majority of it is still supported on the iPhone. Thus, each section is tagged “Desktop” or “iPhone” if it applies to that platform.

Sessions and Logging In

Before any requests can be made to 4D through the Bridge, the user must have a valid session ID justifying that they are allowed to make a request in the first place. Thus, this section focuses on how to log into the framework and it explains a bit on session management.

Session Management

Session management for the 4D Ajax Framework can be broken down into the following:

Authentication: The session is created after login information is successfully authenticated. Authentication is done on the 4D side by either the default 4DAF login system or via Developer Hooks (rolling own password systems via Dax_DevHook_Login). The user can only be logged in from one browser at a time.

Persistance: The session persists until the browser is closed, or if the same user logs in on another browser, or if a timeout occurs from prolonged inactivity.

Security: As a security measure, the session will only work for the same IP address and browser for the duration of the session.

Login

Desktop iPhone

To login to the framework use the following command:

```
dax_login(username, password);
```

Parameter	Type	Description	Optional
username	string	Username to be authenticated by the 4D login system or a custom password system.	
password	string	Password to be authenticated by the 4D login system or a custom password system.	

The users can be authenticated through the default 4DAF login system or, if the developer is rolling out their own custom login system, then it would be through Developer Hook Dax_DevHook_Login.

Login must be successful in order to be able to communicate through the Bridge. After *dax_login* is called one of two handlers take care of a successful or failed login attempt:

- dax_loginSuccess: To handle a successful login.

- dax_loginFail: To handle a failed login.

Both are explained further in the sections that follow.

Examples:

1) Log in with username 'John' and password 'Doe'. The *dax_login* command is called when the web page is loaded (denoted by *onload* within the <BODY> element). This technique is useful if your design allows the login information to be explicitly hard-coded into the page. This is can be handy when you do not require user input for authentication.

```
<body onload="dax_login('John', 'Doe ');" onunload="dax_logout();">>
```

2) Alternatively, you can use an input form and then call the *dax_login* command during the *onclick* of the submit button. Here is sample code within the <BODY> of an HTML page:

   BODY code:

```
<body onload="" onunload="dax_logout();">
<div ID="LoginArea">
       <input id="userID" />
       <input id="passID" type="password" />
       <button type="submit" id="loginbutton"
onclick="dax_login($('userID').value, $('passID').value)">Sign
in</button>
</div>
</body>
```

   JavaScript code:

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

       function dax_loginSuccess() {
              alert("Login Success!");
       }

</script>
```

In this example two input fields, identified as 'userID' and 'passID', retrieve the username and password. Since the password field is of type 'password', the data is masked when the user types in the field. During the *onclick* of the submit button the *dax_login* function is called to authenticate the user. The *value* properties of the input fields are passed into the *dax_login* function to be authenticated by the backend.

Note: Here the '$' character actually translates to 'document.getElementById'. The 4DAF conveniently shortens this for you.

On a successful login, the handler *dax_loginSuccess* is called, in which an alert message Appears. Handlers will be introduced in the next section.

dax_login Handlers

After *dax_login* is called one of two handlers take care of a successful or failed login attempt:

- **dax_loginSuccess**: To handle a successful login.
- **dax_loginFail**: To handle a failed login.

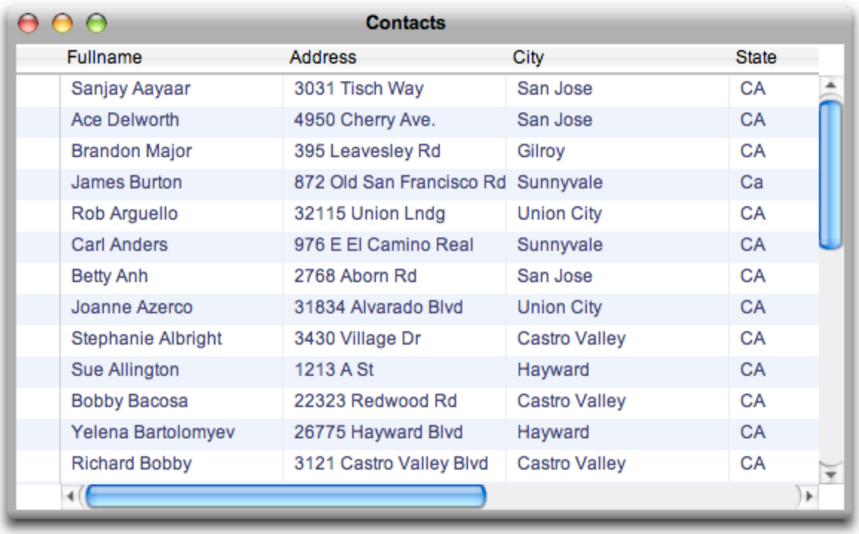
In most cases, the *dax_loginSuccess* handler will perform the rest of the 4DAF initialization code, such as creating data views.

Example:

1) Here's an example that will create a data grid if user is logged in successfully, or prompt an alert message if login failed.

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

// called if user is logged in successfully
function dax_loginSuccess() {

   // create a floating data grid window
   myDataGrid = new dax_dataGrid('Contacts');
   myDataGrid.go();
}

// login failed, alert the user
function dax_loginFail() {

   alert('Login unsuccessful. Please check your username and password and
try again.');
}

</script>
```

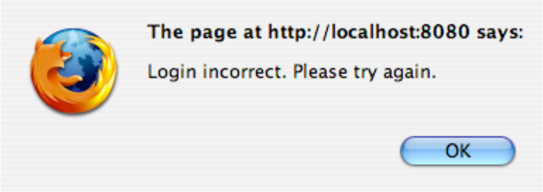
Here, a successful login displays a floating window of a data grid from the Contacts table. The data grid is one of the built-in 4DAF objects you can use to display data.



Contacts

Fullname	Address	City	State
Sanjay Aayaar	3031 Tisch Way	San Jose	CA
Ace Delworth	4950 Cherry Ave.	San Jose	CA
Brandon Major	395 Leavesley Rd	Gilroy	CA
James Burton	872 Old San Francisco Rd	Sunnyvale	Ca
Rob Arguello	32115 Union Lndg	Union City	CA
Carl Anders	976 E El Camino Real	Sunnyvale	CA
Betty Anh	2768 Aborn Rd	San Jose	CA
Joanne Azerco	31834 Alvarado Blvd	Union City	CA
Stephanie Albright	3430 Village Dr	Castro Valley	CA
Sue Allington	1213 A St	Hayward	CA
Bobby Bacosa	22323 Redwood Rd	Castro Valley	CA
Yelena Bartolomyev	26775 Hayward Blvd	Hayward	CA
Richard Bobby	3121 Castro Valley Blvd	Castro Valley	CA

Otherwise, an alert displays an error message for a failed login.



The page at http://localhost:8080 says:

Login incorrect. Please try again.

Log out Desktop iPhone

Use the logout command to sign the current user out and to end the current session.

```
dax_logout();
```

Example:

1) A good location to place the *dax_logout* command is within the <BODY> element tag.

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

Typically a web page is unloaded when the window is closed so this would be an appropriate time to log out the user.

dax_logout Handlers

Just like the *dax_login()* command, *dax_logout()* will trigger one of the two developer-defined functions below:

- **dax_logoutSuccess:** To handle a successful logout.
- **dax_logoutFail:** To handle a failed logout attempt.

Example:

1) In this example an alert is prompted when the logout is a success.

```
// if user is logged out successfully, display the message
function dax_logoutSuccess() {
   alert ('Sign out was successful');
}
```

Localization Desktop

If more than one language library is loaded, the *.localize* call lets the developer choose which language will be used by the 4DAF.

```
dax_bridge.localize(languageAbbreviation);
```

Parameter	Type	Description	Optional

languageAbbreviation	string	Two letter language abbreviation. **Example values:** • en - English • es - Spanish • de - German • fr - French • ja - Japanese	

Example:

1) This example sets the language of the 4DAF user interface to French.

```
// sets language as French
dax_bridge.localize('fr');
```

Language libraries are loaded in the <head> tag of the web page. Here's an example that loads English and French libraries:

```
<head>

<script language="javascript" type="text/javascript" charset="utf-8"
src="/dax/js/localization/resources_en.js"></script>
<script language="javascript" type="text/javascript" charset="utf-8"
src="/dax/js/localization/resources_fr.js"></script>

   ...

</head>
```

Login Properties Desktop iPhone

Once a user is logged in, some information can be obtained via the dax_bridge properties. Current available properties are:

- dax_bridge.language - Two letter abbreviation of the language set for the current user as specified by 4D.
- dax_bridge.sessionId - Unique session id.
- dax_bridge.username - Username of the current user.

Example:

1) This example displays the current user.

```
// show the name of the currently logged in user
alert ('Current user is ' + dax_bridge.username);
```

Tables and fields

This section focuses on getting table and field references. Once you have those references you can then access the properties of those references.

Retrieving table and field references do not initiate requests to 4D. They only reference tables and fields that were loaded during initialization. You can determine which tables and fields are initially loaded when you make them available or not via the Control Panel in the Admin Client.

Note: The terms table and selection are used interchangeably, and refer to Tables, Views, or Developer Created Selections (DCS).

Tables Desktop iPhone

To receive a reference to a particular selection use the following command:

```
var myTable = dax_getTable(selectionName);
```

This command will return the reference to the selection object if the named selection exists.

Example:

1) In this example we are getting a reference to the Contacts table.

```
// myTable variable is assigned reference to Contacts table
var myTable = dax_getTable('Contacts');
```

Table Properties

Once you get a reference to a table, you can access the following properties of that table.

Properties available:

- fields - Array of fields
- fieldsDetail - Array of fields visible in detail forms
- fieldsList - Array of fields visible in list form
- tablealias - Alias for this selection, visible to user
- tablename - Real table name

Example:

1) This example accesses the table name, alias, and fields properties and then displays them in an alert message:

```
// get reference to the table
var myTable = dax_getTable('Contacts');

// get table name and alias
var myTableName = myTable.tablename;
var myTableAlias = myTable.tablealias;
```

```
// get number of fields (.length is javascript array property)
var myTableFieldsCount = myTable.fields.length;

// show table name, alias, and field count
alert ('Selection ' +myTableName + ' has alias ' + myTableAlias + ' and
has ' + myTableFieldsCount + ' fields in it.');
```

The alert can look something like: 'Selection myEmployees has alias Employees and has 5 fields in it.'

Fields Desktop iPhone

To receive a reference to a particular field use the following command:

```
var myField = dax_getField(tableName, fieldName);
```

Note: Remember that the field must somehow be made accessible via the Control Panel (ie. The 'Input' checkbox is selected.

Parameter	Type	Description
tableName	string	Name of the selection (Table, View, or DCS)
fieldName	string	Name of the field

Example:

1) This example gets a reference to the [Employees]FirstName field.

```
// gets reference to [Employees]FirstName field
var myField = dax_getField('Employees', 'FirstName');
```

Field Properties

Once you get a reference to a field, you can access the following properties of that field.

Field properties available:

- fieldalias - field name visible to users
- fieldname - real field name
- fieldtype - field type, such as 'boolean'

Following properties have 'true or 'false' value:

- fieldmandatory - is field mandatory?
- fieldnonEnterable - is field enterable?
- fieldnonModifiable - is field modifiable?
- fieldsearchable - is field visible in search dropdowns?
- fieldunique - is field unique?

All properties are of type string.

Example

1) This example gets a reference to the field [Employees]LastName and then displays several of its properties via an alert.

```
// get reference to the field
var myField = dax_getField('Employee','LastName');

// get field name and alias
var myFieldName = myField.fieldname;
var myFieldAlias = myField.fieldalias;

// get field type
var myFieldType = myField.fieldtype;

// display field information
alert(myFieldName + ' has alias ' + myFieldAlias + ' and has field type '
+ myFieldType);
```

Record Manipulation

The following commands in this section "Record Manipulation" set a precedent in this document because these are the first set of commands that actually send requests to 4D and acquire responses via the Bridge.

This section focuses on record manipulation tasks such as:

- * Adding a record
- * Modifying a record
- * Deleting a record
- Adding a batch of records

Note: The tasks above denoted by the '*' receive responses from 4D in raw XML form. The 4DAF Bridge, at this time, is unable to parse and format the responses for these commands. Thus, the examples for these commands below show sample code on how to parse the XML response via JavaScript in your own handler.

Fortunately, the XML responses are brief since all that is essentially received is a 'succes' or 'fail' in these instances.

Add a Single Record Desktop iPhone

This command adds a record to a specific selection. By selection we include Tables, Views, and Developer Created Selections (DCS).

```
dax_bridge.addRecord(selectionName, fieldArray, valueArray, handler, variable);
```

Parameter	Type	Description	Optional
selectionName	string	Name of the selection (table, view, DCS)	
fieldArray	array	Array of field names to add	
valueArray	array	Array of values to add	
handler	function	Developer's method that's called to handle the XML response from 4D	x
variable	Any JavaScript object (variable, objects, arrays, strings, functions, etc.)	Variable that's passed to the handler (for example, to uniquely identify caller command)	x

The handler is your own method that is called once the XML response from 4D is received.

Example

1) This example will add the following record:

   - [myTable]myField1 = 'myValue1'
   - [myTable]myField2 = 'myValue2'

```
// add the record
dax_bridge.addRecord('myTable', ['myField1', 'myField2'], ['myValue1',
'myValue2']);
```

2) The following example will add a record with only one value for one field:

   - [StoreItems]itemID = '948#0934'

```
// add the record
dax_bridge.addRecord('StoreItems', ['itemID'], ['948#0934']);
```

3) This example will call the developer's handler function *myRecordAdded* after the record is saved successfully, and will then pass a variable named 'StoreItems' to uniquely identify the caller to the *myRecordAdded* handler.

   - [StoreItems]itemID = '948#0934'

```
// add the record
dax_bridge.addRecord('StoreItems', ['itemID'], ['948#0934'],
myRecordAdded, 'StoreItems');
```

Handler definition:

Here you define your own handler to deal with the response of the *addRecord* request to 4D.

Example

1) This example shows a handler named *myRecordAdded*.

Note: The 4DAF Bridge, at this time, is unable to parse and format the responses for the *addRecord* command. Thus, you will receive real Ajax XML responses, which will trigger multiple times for each 'readyState' change. For more information see http://en.wikipedia.org/wiki/XMLHttpRequest#Properties. Consequently, that means your handler function will trigger multiple times as well.

Thus, the sample code below (at *http_response.readyState !=4*) tells the handler to disregard all previous states until state 4 occurs, which means communication with the backend has finished.

```
// developer handler for .addRecord call, must be defined before
// .addRecord is called
// http_response is usual AJAX XML reply, and 'variable' is specified
// as .addRecord parameter

function myRecordAdded(http_response, passedVariable) {

// this portion is required for proper AJAX response handling

if (http_response.readyState != 4)
   return;

// check if passedVariable is 'StoreItems', if it is, notify the user
if (passedVariable == 'StoreItems')
   alert('This record was saved to StoreItems table.');
}
```

Add a Batch of Records

Desktop iPhone

Adding a batch of records adds several records at the same time. The command *addRecordBatch* is much different from the other commands in section “Record Manipulation” because the Bridge actually parses the XML response and formats it into a more presentable manner.

```
dax_bridge.addRecordBatch(selectionName, fieldArray, valueArray, handler,
variable);
```

Parameter	Type	Description	Optional
selectionName	string	Name of the selection (table, view, DCS)	
fieldArray	array	Array of field names inside record array to add	
valueArray	array	Array of values inside record array to add	
handler	function	Developer’s method that's called to handle the XML response from 4D	x
variable	Any JavaScript object (variable, objects, arrays, strings, functions, etc.)	JavaScript objects that's passed to the handler (for example, to uniquely identify the caller command)	x

Examples:

1) Example that will add two records with same fields:

- First record
    - [Employee]firstName = 'John'
    - [Employee]lastName = 'Adams'

- Second record
    - [Employee]firstName = 'Valerie'
    - [Employee]lastName = 'Wellington'

```
// add the record
dax_bridge.addRecordBatch('Employee', [['firstName', 'lastName'],
['firstName', 'lastName']], [['John', 'Adams'], ['Valerie',
'Wellington']]);
```

2) Following example will add a record with only one value:

- [StoreItems]itemID = '948#0934'

```
// add the record
dax_bridge.addRecordBatch('StoreItems', ['itemID'], ['948#0934']);
```

This example will call myRecordAdded handler function after record is saved successfully, and will pass variable named 'StoreItems' to uniquely identify the caller to the myRecordAdded handler.

- [StoreItems]itemID = '948#0934'

Command:

```
// add the record
dax_bridge.addRecordBatch('StoreItems', ['itemID'], ['948#0934'],
myAddRecordBatchHandler, 'StoreItems');
```

Handler definition:

```
function myAddRecordBatchHandler(handlerResponse, passedVariable) {
       // handler code
}
```

The Batch Add Handler Response

The *addRecordBatch* command gets a response from the Bridge called *handlerResponse*, which contains an array of responses for each record that should have been added during the batch. Here are the properties of the *handlerResponse* object for each response in the array:

- **.error** - If the record was not added successfully, a developer defined error message will be included here.
- **.id** - If the record was added successfully, this will contain added record id
- **.success** - Boolean value that indicates if the record was added successfully.

Examples:

Handler code example:

```
// developer handler for .addRecordBatch call, must be defined before
// .addRecordBatch is executed
function myRecordAddedHandler(handlerResponse, passedVariable) {

// loop through handlerResponse to check what's the status of added
// records
for (var recordCount = 0; recordCount < handlerResponse.length;
recordCount ++) {

      // get reference to added record for cleaner code
      var record = handlerResponse[recordCount];

      // check if record was added successfully
      if (record.success) {

         // added successfully, alert added record id
         alert ('Added record id is ' + record.id);
      } else {

         // adding record failed, show error message
         alert ('Record not added, error message is: ' + record.error);

      }
   }
}

// call that will trigger myRecordAddedHandler when done, and send string
'my custom variable' to the handler function
dax_bridge.addRecordBatch('Employee', [['LastName', 'DeptID'],
['LastName', 'DeptID']], [['testXZ', '123'], ['testYZ', '123']],
myRecordAddedHandler, 'my custom variable');
```

Modify Desktop iPhone

This command is used to modify a record. This command has one extra parameter compared to the other commands – recordId.

```
dax_bridge.modifyRecord(selectionName, fieldArray, valueArray, recordId,
handler, passedVariable);
```

Parameter	Type	Description	Optional
selectionName	string	Name of the selection (table, view, DCS)	
fieldArray	array	Array of field names to modify	
valueArray	array	Array of values to modify	
recordId	string	Record number of the record you want changed. This value is formatted in an ‘[x][y]’ manner.	
handler	function	Developer’s method that's called to handle the XML response from 4D.	x
variable	Any JavaScript object (ie. variable, objects, arrays, strings, functions, etc.)	Variable that's passed to the handler (for example, to uniquely identify the caller command)	x

Note: Only fields with modified values need to be passed.

Example:

1) This example modifies itemID and itemName fields for record with id [3][4]. Other fields remain unchanged.

- [StoreItems]itemID = '948#0934'
- [StoreItems]itemName = 'Wooden Chair'

```
// modify the record
dax_bridge.modifyRecord('StoreItems', ['itemID', 'itemName'],
['948#0934', 'Wooden Chair'], '[3][4]');
```

2) This example modifies itemColor field for record with id [3][6]. Other fields remain unchanged. Handler named myRecordModified is called, with table name as a passedVariable.

   - [StoreItems]itemColor = 'Orange'

   Command:

```
// modify the record
dax_bridge.modifyRecord('StoreItems', ['itemColor'], ['Orange'],
'[3][6]', myRecordModified, 'StoreItems');
```

   Handler definition:

Note: The 4DAF Bridge, at this time, is unable to parse and format the responses for the *modifyRecord* command. Thus, you will receive real Ajax XML responses, which will trigger multiple times for each 'readyState' change. For more information see http://en.wikipedia.org/wiki/XMLHttpRequest#Properties. Consequently, that means your handler function will trigger multiple times as well.

Thus, the sample code below (at *http_response.readyState !=4*) tells the handler to disregard all previous states until state 4 occurs, which means communication with the backend has finished.

```
// developer handler for .modifyRecord call, must be defined before
.modifyRecord is called
// http_response is usual AJAX XML reply, and passedVariable is specified
as .modifyRecord parameter

function myRecordModified(http_response, passedVariable) {

// this portion is required for proper AJAX response handling

if (http_response.readyState != 4)
   return;

// check if passedVariable is 'StoreItems', if it is, notify the user
if (passedVariable == 'StoreItems')
   alert('This record was saved to StoreItems table.');

}
```

Delete Desktop iPhone

The command allows you to delete a single record or an array of records.

```
dax_bridge.deleteRecords(selectionName, recordId(s), handler, passedVariable);
```

Parameter	Type	Description	Optional
selectionName	string	Name of the selection (table, view, DCS)	

recordId(s)	array	Array of record ID's to delete records	
handler	function	Method that's called with XML response from 4D	x
variable	Any JavaScript object (ie. variable, objects, arrays, strings, functions, etc.)	Variable that's passed to the handler (for example, to uniquely identify caller command)	x

Examples:

1) Example that deletes a single record with record id [6][0]:

```
dax_bridge.deleteRecords('myTable', '[6][0]');
```

2) Example that deletes multiple records with record ids [6][0] and [6][3]:

```
dax_bridge.deleteRecords('myTable', ['[6][0]', '[6][3]']);
```

3) This example deletes the record with id [6][7]. The developer's handler named *myRecordDeleted* is called, with table name as a passedVariable to uniquely identify from which table the records were deleted.

Command:

```
// modify the record
dax_bridge.deleteRecords('myTable', '[6][7]', myRecordDeleted,
'StoreItems');
```

Handler definition:

Note: The 4DAF Bridge, at this time, is unable to parse and format the responses for the *deleteRecords* command. Thus, you will receive real Ajax XML responses, which will trigger multiple times for each 'readyState' change. For more information see http://en.wikipedia.org/wiki/XMLHttpRequest#Properties. Consequently, that means your handler function will trigger multiple times as well.

Thus, the sample code below (at *http_response.readyState !=4*) tells the handler to disregard all previous states until state 4 occurs, which means communication with the backend has finished.

```
// developer handler for .deleteRecords call, must be defined before
.deleteRecords is called
// http_response is usual AJAX XML reply, and passedVariable is specified
as .deleteRecords parameter

function myRecordDeleted(http_response, passedVariable) {

// this portion is required for proper AJAX response handling
if (http_response.readyState != 4)
   return;

// check if passedVariable is 'StoreItems', if it is, notify the user
if (passedVariable == 'StoreItems')
   alert('Record was deleted from StoreItems table.');

}
```

Data Access

The commands in this section receive much more complex responses than other commands in this document. Here, queries are made to 4D and the responses can contain choice lists, single records, or a list of query results, depending on your situation.

Unlike most of the commands in section "Record Manipulation", the Bridge here parses and formats the responses from 4D (instead of leaving the responses untouched in XML).

Choice lists Desktop iPhone

Use this command to request a 4D choice list by name and then have its values returned as an array. When the array is returned from 4D, the developer must execute his or her own handler to handle it.

```
dax_bridge.getChoiceList(choiceList, handler, passedValue);
```

Parameter	Type	Description	Optional
choiceList	string	Name of the choice list	
handler	function	Developer's method that's called when the list of elements is obtained from 4D.	
passedValue	Any JavaScript object (ie. variable, objects, arrays, strings, functions, etc.)	JavaScript object, variable, array, etc. that is passed to handler	

Examples:

1) This example asks for values of a choice list named 'myList'. Notice that the handler *myListHandler* is the developer's own method.

   Command:

```
dax_bridge.getChoiceList('myList', myListHandler);
```

   Handler:

```
// myList will be an array
function myListHandler(myList) {

// display number of list elements
alert ('There are ' + myList.length + ' list elements.');

}
```

2) This example uses the passedVariable parameter of Example that asks for values of choice list named 'anotherList', and passes a variable named 'anotherList' to indicate the list name to the handler. The name of the choice can be a useful piece of information. Even though ‘anotherList’ is specified to 4D in the first parameter, it is not returned with the response that is sent from 4D.:

   Command:

```
dax_bridge.getChoiceList('anotherList', myListHandler, 'anotherList');
```

   Handler:

```
// myList will be an array
function myListHandler(myList, passedValue) {

// display number of list elements
alert ('There are ' + myList.length + ' list elements. List name is ' +
passedValue + '.');

}
```

Request a Single record Desktop iPhone

This command requests a single record from 4D based on the record ID. Only fields set as input in control panel are passed.

```
dax_bridge.getRecord(selectionName, recordId, handler, passedValue);
```

Parameter	Type	Description	Optional
selectionName	string	Name of the selection (table, view, DCS)	
recordId	string	Record ID	
handler	string	Developer's method that's called when the reply from 4D arrives	
passedValue	Any JavaScript object (ie. variable, objects, arrays, strings, functions, etc.)	Javascript object, variable, array, etc. that is passed to handler	x

Handler will receive parsed record as an object. For the detailed information on the record structure, check the handler information for the query object. Structures are identical, with the exception that parsed record for getRecord will always have only one record.

Example

1) This example will request a record with id [5][6] from table [myTable]:

Handler:

```
function myGetRecordHandler(parsedRecord) {

   // handler code

}
```

Command:

```
dax_bridge.getRecord('myTable', '[5][6]', myGetRecordHandler);
```

Query

Desktop iPhone

Query object makes query calls to 4D, passes and retrieves custom values, and holds the query response from 4D.

Following command will create query object, which is then used for further operations with the provided table.

```
myQuery = new dax_query(selectionName);
```

- selectionName - name of the selection (table, view, DCS)

Important part in creating the query is assigning the handler function. Handler function provides developer with result of the query, and fires when response is received and parsed. Here a basic format for assigning handler function:

```
// create handler function
function myQueryHandler(parsedRecords) {

   // code that handles query reply goes here

}

// assign the handler to the query
myQuery.handler = myQueryHandler;
```

Handler examples are provided below.

Properties

Query properties are updated every time after query is ran. These properties let developer know what's the number of records in selection, number of returned records, and other query-related information. Accessed via myQuery.*propertyName*:

Parameter	Type	Description
size	string	number or records returned by 4D
recordsInSelection	string	total records in selection

lastRecord	boolean	set to true if there are no more records after ones received

New, add, and run query

.newQuery, .addQuery, and .runQuery commands contain basic workflow of performing the query. .newQuery clears the search parameters, .addQuery adds one search parameters, and finally .runQuery executes the query. If query doesn't include search parameters, .runQuery is the only command required.

This command clears the search options that were set before, and resets position to first record in selection.

```
myQuery.newQuery();
```

This commands adds a search parameter. Call this command to add several search options.

```
myQuery.addQuery(field name, operator, value, and/or flag);
```

Parameters

Parameter	Type	Description	Optional
Field name	string	Name of the queried field	
Operator	string	Search operator **Possible values:** ○ = ○ # ○ < ○ > ○ <= ○ >=	
Value	string	Value that search is based on	
And/or flag	string	Indicates if this search option is added as 'and' or 'or' in relation to other search options. **Default value:** ○ 'and'	x

This command executes the query:

```
myQuery.runQuery(start record, length);
```

Parameters:

Parameter	Type	Description	Optional
start record	integer	First record sent, integer that starts from 1	x
length	integer	Number of records requested, integer that starts from 0	x

Examples:

1) Example for query that will request first 10 records:

```
myQuery.runQuery(1, 10);
```

2) Example for query that will request 5 records starting from 15th record:

```
myQuery.runQuery(15, 5);
```

3) Example that will request 15 records with last name Jones and where Jones is not a manager:

```
myQuery.newQuery();
myQuery.addQuery('FirstName', '=', 'Jones');
myQuery.addQuery('isManager', '=', 'false');
myQuery.runQuery(1, 15);
```

4) Example where 10 records are requested where person is either a manager or executive:

```
myQuery.newQuery();
myQuery.addQuery('isExecutive', '=', 'true', 'and');
myQuery.addQuery('isManager', '=', 'true', 'or');
myQuery.runQuery(1, 10);
```

Query response and handler

Query handler will receive records as a first parameter, as shown in handler template below.

```
// create handler function
function myQueryHandler(parsedRecords) {

   // code that handles query reply goes here

}
```

Parsed Records

An object named *parsedRecords* is returned from the *runQuery* and *getRecord* requests to 4D. At the highest level, the parsedRecords object is an array of records:

- parsedRecords - array of records
    - record 1, first array element, accessed as .parsedRecords[0]
    - record 2, second element, .parsedRecords[1]
    - record 3, third element, .parsedRecords[2]

Note: Using the *getRecord* command returns only one record not an array of records, as is the case when using the *runQuery* command.

Structure for record elements, accessed as .parsedRecord[recordNumber], where recordNumber starts from 0:

- locked - is record locked?, ex: 'false'
- record id - record id, ex: '[4][5]'
- selection id - record number in current selection, ex: '4'
- fields - array of fields

    - field 1 - first field, accessed as .parsedRecords[recordNumber].fields[0]
    - field 2 - second field, .parsedRecords[recordNumber].fields[1]
    - field 3 - third field, .parsedRecords[recordNumber].fields[2]

Structure for fields within the record, accessed as .parsedRecord[recordNumber].fields[fieldNumber], where fieldNumber starts from 0:

- id - field id, ex: '[3][4]'
- originalValue - unformatted value, ex: 2008-04-20
- value - formatted value, ex: 20-Apr-2008
- height - if image, holds image height in pixels
- width - if image, holds image width in pixels

Examples:

1) Get number of returned records

```
var numberOfRecords = parsedRecords.length;
```

2) Check if second record is locked

```
var secondRecordLocked = parsedRecords[1].locked;
```

3) Get number of fields returned for second record

```
var numberOfFields = parsedRecords[1].fields.length;
```

4) Get field and name for second record, third field

```
var fieldName = dax_getField(parsedRecords[1].fields[2].id);
var value = parsedRecords[1].fields[2].value;
```

5) Handler code that goes through the parsedRecords and displays values for every record:

```
// first, check if any records were returned
if (parsedRecords.length > 0) {

   // start looping through received records
   for (var recordCount = 0; recordCount < queryResult.length;
recordCount++) {

      // create new variable that will hold field name and record value
      var myRecord = '';

      // loop through fields
      for (var fieldCount = 0; fieldCount <
queryResult[recordCount].fields.length; fieldCount++) {

         // for current field, get field name and value
         var fieldName =
dax_getField(parsedRecords[recordCount].fields[fieldCount].id);
         var value = parsedRecords[recordCount].fields[fieldCount].value;
         // append those values to the myRecord variable so we can
display all values later
         myRecord += fieldName + ': ' + value + ', ';

      }

   // now that we have looped through all fields, display the current
record before going to next one
   // example output: "firstName: Annie, lastName: Johnson"
   alert (myRecord);

   }
}
```

Send custom values to 4D

Send custom value names and pairs to the back end using the following:

```
myQuery.addCustomValue(name, value);
```

- **name –** String, name of the custom value.
- **value –** Value of custom value.

Clear custom values using the following:

```
myQuery.clearCustomValues();
```

These values are retrieved by the GET WEB FORM VALUES command in 4D.

Example:

1) Sent two values to the 4D when the query is made. Both of these are retrieved from two input fields present on the HTML page.

   Javascript:

```
// clear all existing values
myQuery.clearCustomValues();

// get values from input fields
var myCity = $('myCity').value;
var myState = $('myState').value;

// add values to the query
myQuery.addCustomValue('myCity', myCity);
myQuery.addCustomValue('myState', myState);

// run query with stored custom values to 4D
myQuery.runQuery(1, 0);
```

   HTML:

```
My City: <input type="text" id="myCity" />
My State: <input type="text" id="myState" />
```

Get custom values from 4D

This command retrieves custom values that were sent from 4D via the last query request.

To get the custom values object:

```
myCustomValues = myQuery.getCustomValuesFrom4D();
```

On the front end side, the object received has the following properties:

* **myCustomValues.length** -> number of returned values

Name/value pair, where valueNumber is an integer starting from 0 and above:

* **myCustomValues[valueNumber].name**
* **myCustomValues[valueNumber].value**

Example

1) This example will make a query call to 4D and retrieve custom values in the handler:

   Here is sample code for the developer’s own handler:

```
function myQueryHandler() {

   // get the custom values
   var myCustomValues = myQuery.getCustomValuesFrom4D();

   // tell user how many custom values came in
   alert (myCustomValues.length + ' values returned from 4D.');

}
```

   Query code:

```
// create query object
var myQuery = new dax_query('myTable');

// assign the above created handler function
myQuery.handler = myQueryHandler;

// since we're interested in custom values only, we can run query now
without setting any other parameters
myQuery.runQuery(1, 0);
```

File Linking and Dependencies

The iPhone and iPod Touch

iPhone

The iPhone bridge requires the resource *iphone_bridge.js* and at least one language localization. Make sure that the Bridge is linked before localization is applied. Here's an example linking with English localization:

Example

1) This example links the iPhone Bridge with English localization:

```
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/iphone/common/iphone_bridge.js"></script>
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/iphone/common/iphone_loc_en.js"></script>
```

Miscellaneous

4DAF version

Desktop

4DAF version is stored in the variable:

```
dax_bridge.version
```

Force synchronous calls

Desktop iPhone

4DAF bridge calls are by default asynchronous, meaning that your code doesn't have to wait while 4D receives the request and replies to it. If you would like to force the bridge calls to be synchronous, use the following call:

```
dax_bridge.alwaysUseSyncCall(boolean);
```

Parameters

Parameter	Type	Description	Optional
boolean	boolean	Set to ‘true’ to force sync calls, false to use async calls	

Example:

1) In this example sets the calls to 4D to be synchronous.

```
// force sync calls
```

```
dax_bridge.alwaysUseSyncCall(true);
```

With a synchronous call, your code will wait until the request is processed before continuing.

Format Conversion

Desktop iPhone

This command converts the value from one specific format to another.

```
dax_bridge.applyFieldFormat(value, originalFormat, newFormat);
```

Parameter	Type	Description	Optional
Value	string	Value that's formatted	
originalFormat	string	Current format the value is in	
newFormat	string	Format in which new value is returned	

This table displays allowed original and new formats:

* Date - available as original and new formats
  o '4daf_default' - value 4D expects, YYYY-MM-DD format
  o 'dMM-DD-YYYY'
  o 'dDD-MM-YYYY'
  o 'dDD-MMM-YYYY'
* Time - use only as new formats, pass 'null' as old format
  o 'tHH:MM:SS'
  o 'tHH:MM:SS am/pm'
  o 'tHH:MM'
  o 'tHH:MM am/pm'

Managing Data Modification

Whenever 4D Ajax Framework is about to save or delete a record, the developer is given the opportunity to deny the action or handle other related tasks as necessary via two data modification developer hooks.

DAX_DevHook_SaveRecord

DAX_DevHook_SaveRecord is called anytime 4D Ajax Framework is about to save a record. It allows the developer to add any necessary internal field modifications, such as date created or modified fields, or to handle other necessary actions such as updating related records or a system log.

This method is always called whenever 4D Ajax Framework is about to save a record. A pointer to the table and the record number for the record to be saved are passed in. Developers can add any necessary internal field modifications here. For example, if you have a Date_Created field in your table you can set that value here. To deny the save record return False from this method.

```
Passed:
$1       POINTER    Pointer to the table where the record is being saved
$2       LONGINT    Record number for the record that is being saved

Returned:
$0         BOOLEAN  True=OK to save, False=Do not save and send an error
```

The record that is about to be saved is already loaded in read write mode and the changes to its fields have already been made. Any further processing is up to your specific needs.

The record is loaded in READ WRITE mode when this method is called, but the record has not been saved yet. 4D Ajax Framework will expect the record to be loaded still in READ WRITE mode after this call. If you need to modify the selection within the table, you must first save the record and then restore the selection in READ WRITE mode when your code completes.

Any code you have in Triggers will execute normally.

DAX_DevHook_SaveRecord Updated For Version 11.2

This has been updated with additional comments about how to add your own custom error message when a record is rejected. The execution point for this method has also been modified to run for every record that is being saved for the DAX/AddRecordBatch url (Batch Saving).

DAX_DevHook_DeleteRecord

DAX_DevHook_DeleteRecord is called anytime 4D Ajax Framework is about to delete a record. It allows the developer to deny the deletion or to handle other necessary actions such as deleting related records or updating a system log.

This method is always called whenever 4D Ajax Framework is about to delete a record. A pointer to the table and the record number for the record to be deleted are passed in. Developers can add any necessary handling here. For instance, if you need to have related records deleted when this record is deleted, you can do so here. To deny the delete record return False from this method.

```
Passed:
$1       POINTER    Pointer to the table where the record is being deleted
```

```
$2       LONGINT    Record number for the record that is being deleted

Returned:
$0         BOOLEAN  True=OK to delete, False=Do not delete and send an
                    error
```

The record that is about to be deleted is already loaded in read write mode. Any further processing is up to your specific needs.

The record is loaded in READ WRITE mode when this method is called. 4D Ajax Framework will expect the record to be loaded still in READ WRITE mode after this call. If you modify the selection within the table, you must restore the selection in READ WRITE mode when your code completes. Any code you have in Triggers will execute normally.

Preset Queries

DAX_DevHook_QueryAdd

4D Ajax Framework version 1.1 also introduces a new Developer Hook to programmatically create a new preset query. This hook is `DAX_DevHook_QueryAdd`.

`DAX_DevHook_QueryAdd` is used by the developer to create preset queries and assign them to a selection. These saved queries then appear as tabs in a Grid view. For instance you could create a saved query for each letter of the alphabet and assign it to the last name field of a Contacts table. Then the user would see 26 tabs that each return the results for the associated letter.

Adding a custom query is done by calling `DAX_Dev_QueryCreate` and passing nine (9) parameters:

```
Passed:
$1       TEXT       Name of the Query to Add
$2       TEXT       Name of the selection (table) the Query is for
$3       TEXT       Name of the field to use for sorting
$4       TEXT       Sort order to use. It must be either 'asc' or 'desc'
                    (for ascending or descending)
$5       POINTER    Pointer to a Text Array to set the line links.
                    Determines if multiple lines are processed as AND or
                    OR. The values must be either 'and' or 'or'
$6       POINTER    Pointer to a Text Array to set the query field names.
                    This is an array of field names that are to be used
                    for the query
$7       POINTER    Array of operators that are to be used for the query
$8       POINTER    Pointer to a Text Array to set the query conditions.
                    This is an array of conditions (value to search for)
                    that are to be used for the query
$9       POINTER    Pointer to a Boolean Array to set the condition in
                    method flag. This is a Boolean array that indicates if
                    the condition is to be treated as text or is the name
                    of a method that is to be executed. If it is a method,
                    the method must return a text value
```

Example

```
C_TEXT($queryName_t;$tableName_t;$sortFieldName_t;$sortOrder_t)
C_BOOLEAN($added_b)
C_LONGINT($size_l)

$queryName_t:="ATestQuery"
$tableName_t:="Album" ` Table name(->[Album])
$sortFieldName_t:="Album_Name" ` Field name(->[Album]Album_Name)
$sortOrder_t:="asc"

$size_l:=3
ARRAY TEXT($links_at;$size_l)
ARRAY TEXT($fields_at;$size_l)
ARRAY TEXT($operators_at;$size_l)
ARRAY TEXT($conditions_at;$size_l)
ARRAY BOOLEAN($flags_ab;$size_l)
```

```
$links_at{1}:="or"
$fields_at{1}:="Album_Name"
$operators_at{1}:="Starts With"
$conditions_at{1}:="A"
$flags_ab{1}:=False

$links_at{2}:="or"
$fields_at{2}:="Album_Name"
$operators_at{2}:="Starts With"
$conditions_at{2}:="B"
$flags_ab{2}:=False

$links_at{3}:="or"
$fields_at{3}:="Album_Name"
$operators_at{3}:="Starts With"
$conditions_at{3}:="C"
$flags_ab{3}:=False
$added_b:=DAX_Dev_QueryCreate($queryName_t;$tableName_t;
     $sortFieldName_t;$sortOrder_t;->$links_at;->$fields_at;
->$operators_at;->$conditions_at;->$flags_ab)
```

Valid Operators

The following are the valid operators you may use in the operators array:

- "equal" (same as =)
- "notequal" (same as #)
- "less" (same as <)
- "greater" (same as >)
- "lesseq" (same as <=)
- "greatereq" (same as >=)
- "ends with" (same as = and appending an @)
- "starts with" (same as = and prepending an @)

Method Parameters

If the condition you pass is a method to be executed the method will receive the following parameters when it is run.

- $1 TEXT - Query Name
- $2 TEXT - Selection (table) Name
- $3 TEXT - Field Name

Example

You can add as many custom queries as you like. Just make one call to DAX_Dev_QueryCreate for each query you are adding. This example creates a 3 line query to find album names ([Album]Album_Name) that start with 'A', 'B', or 'C' in the table [Album].

```
C_TEXT($queryName_t;$tableName_t;$sortFieldName_t;$sortOrder_t)
C_BOOLEAN($added_b)
C_LONGINT($size_l)

$queryName_t:="ATestQuery"
$tableName_t:="Album"   ` Table name(->[Album])
$sortFieldName_t:="Album_Name"   ` Field name(->[Album]Album_Name)
$sortOrder_t:="asc"
```

```
$size_l:=3
ARRAY TEXT($links_at;$size_l)
ARRAY TEXT($fields_at;$size_l)
ARRAY TEXT($operators_at;$size_l)
ARRAY TEXT($conditions_at;$size_l)
ARRAY BOOLEAN($flags_ab;$size_l)

$links_at{1}:="or"
$fields_at{1}:="Album_Name"
$operators_at{1}:="Starts With"
$conditions_at{1}:="A"
$flags_ab{1}:=False

$links_at{2}:="or"
$fields_at{2}:="Album_Name"
$operators_at{2}:="Starts With"
$conditions_at{2}:="B"
$flags_ab{2}:=False

$links_at{3}:="or"
$fields_at{3}:="Album_Name"
$operators_at{3}:="Starts With"
$conditions_at{3}:="C"
$flags_ab{3}:=False

$added_b:=DAX_Dev_QueryCreate($queryName_t;$tableName_t;$sortFieldName_t;
$sortOrder_t;->$links_at;->$fields_at;->$operators_at;->$conditions_at;-
>$flags_ab)
```

DAX_DevHook_QueryFilter

DAX_DevHook_QueryFilter is used by the developer to change the selection when it is about to be sent to the browser. This method is always called whenever 4D Ajax Framework is about to display a selection. A pointer to a longint array of table numbers and a pointer to a text array of set names are passed. The developer can then remove records that are not valid. For instance, if you have a 'Deleted' flag field, you could remove all of the records that are flagged as deleted.

DAX_DevHook_QueryFilter receives two pointer parameters:

```
Passed:
$1       POINTER    Pointer to a longint array of Table Numbers
$2       POINTER    Pointer to a Text array of Set Names
```

By default $1 is assigned to the local variable $setTableIDs_p and $2 is assigned to the local variable $setNames_p. For every table in the $setTableIDs_p array there is a corresponding Set in the $setNames_p array.

If you need to modify the selection you should:

```
1 Use Set ($setNames_p{$index}->)
2 Modify the selection
3 Update the set by clearing and recreating it (or using Union, Intersection,
etc.)
```

Note: The passed sets must still exist after this call. If you use CLEAR SET, be sure to use CREATE SET with an identical set name.

Example

```
For($i;1;Size of array($setTableIDs_p->))
```

```
  Case of
  : ($setTableIDs_p->{$i}=Table(->[My_Table]))
    USE SET($setTableIDs_p->{$i})
    CLEAR SET($setNames_p->{$i})
    QUERY SELECTION([My_Table];[My_Table]My_Field="My_Value")
    CREATE SET([My_Table];$setNames_p->{$i})
  End case
End for
```

DAX_DevHook_OnQuery

4D Ajax Framework version 1.2 introduces a new Developer Hook that is triggered every time the 4D Ajax Framework queries the data file. This hook is DAX_DevHook_OnQuery. *(Note: This hook is not triggered when a query is made on a DCS, however)*

This hook can be used to override the queries that are done by the 4DAF to implement your own query code. This is a very powerful feature that allows you to catch the query event and perform further queries and calculations if need be. The advantage is that your application can now have powerful and complex queries that can seem simple and transparent to the end user. This feature is only recommended if you are comfortable with building complex queries using pointers and text values.

Parameters

```
Passed:
$1       LONGINT    Table number of the table that gets the Selection
$2       POINTER    Pointer to a Longint Array of Table Numbers
$3       POINTER    Pointer to a Longint Array of Field Numbers
$4       POINTER    Pointer to a Text Array of Values to Query For
$5       POINTER    Pointer to a Text Array of Query Comparators
$6       POINTER    Pointer to a Text Array of Query Line Links
$7       BOOLEAN    Flag: Query in Current Selection?

Returned:
$0         BOOLEAN  Flag: Developer Performed the Query
```

By default the parameters are assigned to local variables as follows:

```
$selectionNumber_l:=$1
$tableNumbers_p:=$2
$fieldNumbers_p:=$3
$queryValues_p:=$4
$queryComparators_p:=$5
$queryLineLinks_p:=$6
$queryInSelection_b:=$7
```

If you perform the query yourself you must return True from this method. By default the local variable $queryDone_b is returned in $0. If this method returns False, 4DAF will perform the query normally.

The first parameter is the table number for the table that should end up with a selection. When your code has finished executing the records should remain in selection in this table so the 4DAF code can continue with the response to the front-end.

You can retrieve a pointer to this table with the following line of code:

```
$table_p:=Table($selectionNumber_l) `get a pointer to the selection table
```

Example

The following is a simple example of how to use the passed in parameters to build your own queries.

```
  ` get the name of the selection from the passed ID
$selectionName_t:=Dax_Dev_Struct_GetNameFromID ($selectionNumber_l)

Case of

  : ($selectionName_t="Album")
  ` Handle the query here ourselves
    $queryDone_b:=True

    $table_p:=Table($selectionNumber_l)  ` get a pointer to the selection
table
    $size_l:=Size of array($tableNumbers_p->)

    If ($size_l=1)  ` single line query
      $field_p:=Field($tableNumbers_p->{1};$fieldNumbers_p->{1})  ` get a
pointer to the field
      $operator_t:=$queryComparators_p->{1}
      $value_t:=$queryValues_p->{1}

      If ($queryInSelection_b)
        QUERY SELECTION($table_p->;$field_p->;$operator_t;$value_t)
      Else
        QUERY($table_p->;$field_p->;$operator_t;$value_t)
      End if

    Else   ` multi - line query

      For ($i;1;$size_l)

        ` You can build queries as separate parameters as done below.
        ` However you can not pass the pointers for operator and value
directly, they must be
        ` copied into stand-alone variables.
        ` Note that you can also pass a text value to query on to any
standard field type
        `(not Pics, Blob, Subtable)

        $field_p:=Field($tableNumbers_p->{$i};$fieldNumbers_p->{$i})  `
get a pointer to the field
        $operator_t:=$queryComparators_p->{$i}  ` copy the operator  to a
local text variable
        $value_t:=$queryValues_p->{$i}  ` copy the value to a local text
variable

        Case of

          :($i=1)  ` first query call don't need AND or OR
            If ($queryInSelection_b)
              QUERY SELECTION($table_p->;$field_p-
>;$operator_t;$value_t;*)
            Else
              QUERY($table_p->;$field_p->;$operator_t;$value_t;*)
            End if

          :($i=$size_l)  ` Last Query call don't add the continue ( * )
parameter
```

```
            If ($queryInSelection_b)
              QUERY SELECTION($table_p->;$field_p->;$operator_t;$value_t)
            Else
              QUERY($table_p->;$field_p->;$operator_t;$value_t)
            End if

          Else   ` Somewhere in the middle of things, need AND / OR and
continuation *

            If ($queryLineLinks_p->{$i}="And")

              If ($queryInSelection_b)
                QUERY SELECTION($table_p->; & ;$field_p-
>;$operator_t;$value_t;*)
              Else
                QUERY($table_p->; & ;$field_p->;$operator_t;$value_t;*)
              End if

            Else

              If ($queryInSelection_b)
                QUERY SELECTION($table_p->; | ;$field_p-
>;$operator_t;$value_t;*)
              Else
                QUERY($table_p->; | ;$field_p->;$operator_t;$value_t;*)
              End if

          End if

        End case

      End for

    End if

Else
  ` Let 4DAF do the query
  $queryDone_b:=False

End case


  ` *** Return True if you have performed the query
  `       Return False to have 4DAF perform the query
$0:=$queryDone_b
```

Query Templates

If you are comfortable with XML, you can now create your own Query Templates. Query Templates are stored in the folder \Extras\Support\Query Templates, and each template is an XML file. You can create or edit these files to create templates of your own. Here is the XML definition:

XML Definition

The root element for a query template is <queries>

Within the <queries> element 1-n separate queries can be defined. Each of these queries creates a tab in the data window display.

Each query is defined within a <query> element. The <query> element must always have 5 attributes:

- name – this is the name that is displayed on the tab that is created by the query
- sortfield – this is a place holder that should always be empty ""
- sortorder – this is a place holder that should always be "Asc"
- tableid – this is a place holder that should always be empty ""
- maxnb – this is a place holder that should always be "-1"

Within each <query> element 1-n separate query conditions can be defined. Each definition is evaluated in turn to determine the selection of records for this query.

Each condition is defined within a <queryline> element. The <queryline> element must have 5 attributes:

- condition – this is the value to query for
- field – this is a place holder that should always be empty ""
- link – this must always be either "And" or "Or" and determines how multiple querylines are evaluated together
- operator – this must be a textual representation of the operator to use for this condition. Valid operators are:
  - "equal"
  - "notequal"
  - "less"
  - "greater"
  - "lesseq"
  - "greatereq"
  - "ends with"
  - "starts with"
- method – this must always be either "True" or "False" and determines if the condition should be treated as a condition to query on or as a method to be executed which will return the condition to query on

See the included query template files for examples of this structure.

Choice Lists

DAX DevHook InstallChoiceList

In several UI implementations, a choice list is an ideal way to allow the user to select a predefined item from the list. This type of implementation is often used to avoid human error. In version 1.2, a new developer hook method is created to allow a choice list to be used in any enterable object (frontend) even if the bound field doesn't have a choice list assigned to it.

The installation of the choice list to a field must be done within a method called DAX_DevHook_InstallChoiceList. Before you start installing a choice list to a field, you need to make sure that the list exists or will exist in your database before a request to the field (or table that the field belongs to) is made by the Web frontend.

In the method DAX_DevHook_InstallChoiceList, you need to make a call to a method named DAX_Dev_SetChoiceList. This method will set a list (based on the name) to the field (in 4D Ajax Framework structure).

DAX_Dev_SetChoiceList

```
Call Syntax:  DAX_Dev_SetChoiceList (TableName;FieldName;ListName{;Overwrite})
```

```
   Passed:
   $1       TEXT     Name of the Table/View/DCS
   $2       TEXT     Name of the Field/View/DCS
   $3       TEXT     Name of the list
   $4       BOOLEAN  Optional: True is to use the list even if the field
                     already has a choice list installed in the database
                     structure. Omitted or False is to set a list to the
                     field only if the field has no choice list assigned to.
```

Note: List name must only consist of alphanumeric characters and the _ (underscore).

Installation

Suppose you want to allow a choice list to be used in a field that is accepting the state name in the US. You can call the DAX_Dev_SetChoiceList in one of the following ways:

Set a choice list using IDs.

```
 DAX_Dev_SetChoiceList (Table name(2);Field name(2;1);"States")
```

2. Set a choice list using names

```
 DAX_Dev_SetChoiceList ("Location";"States";"States")
```

3. Force 4DAF to use this list named "States" even if the field already has a choicelist assigned to it.

```
 DAX_Dev_SetChoiceList (Table name(2);Field name(2;1);"States";True)
```

Please note that the choice list installation happens during the database startup.

DAX_DevHook_ListContents

DAX_DevHook_ListContents is provided for the Developer to override a list before it is sent to the browser. Every time the 4D Ajax Framework is about to send a list to the browser the developer is given the opportunity to modify the list. This allows you to remove items from the list or create the list completely in code.

The list itself is assigned to a field using the structure editor and the field properties inspector. Simply assign a list defined in the Lists editor to a field using the Choices & Help tab. Note that you can create an empty list in the List editor and assign it to the field and then create

the list in code using this hook. DAX_DevHook_ListContents receives two parameters.

```
Passed:
$1        TEXT      Name of the list that has been requested
$2        POINTER   Pointer to a Text array containing the list items
```

By default $1 is assigned to the local variable $list_t and $2 is assigned to the local variable $listItems_p. This method receives the name of the list and the contents that will be returned to the browser. For every list you wish to modify, create a case to catch it and then add the code that is necessary for your situation. This might consist of removing items from the list or creating the complete list from scratch.

Example

```
Case of
  ` EXAMPLE - remove a value from the list
  :($list_t="My_List")
    C_LONGINT($find_l)
    $find_l:=Find in array($listItems_p->;"Remove_Value")
    If($find_l#-1)
      DELETE ELEMENT($listItems_p->;$find_l;1)
    End if

  ` EXAMPLE - Completely create the list
  :($list_t="Another_List")
    READ ONLY([My_Table])
    ALL RECORDS([My_Table])
    DISTINCT VALUES([My_Table]List_Items;$listItems_p->)
    UNLOAD RECORD([My_Table])
End case
```

Creating DCS Views

Developer Created Selections allow the developer to add a new item to the list of tables (Views) that are displayed on the front-end using arrays as the data source. The arrays can then be populated with data from any data source the developer has access to. To the end user a DCS view appears and acts no differently than any other table.

Since DCS does a lot of array manipulation it is highly recommended that DCS be used only in systems that will be deployed compiled.

DAX_DevHook_DCS_ViewAdd

DAX_DevHook_DCS_ViewAdd is provided for the Developer to add custom Views to the 4D Ajax Framework web interface using arrays as the data source. Views added with this method will be added to the standard list of tables available on the front-end and will appear to the end user as just another table. 4D Ajax Framework will automatically add any Views defined in this method when the database starts up. Using DAX_DevHook_DCS_ViewAdd is the only supported way to add a DCS view.

When using custom views the developer must handle all of the other DCS hooks as well:

```
DAX_DevHook_DCS_SetSelection
DAX_DevHook_DCS_RecordSave
DAX_DevHook_DCS_RecordDelete
```

Adding a custom view is done by calling DAX_Dev_DCS_AddCustomView and passing seven parameters.

1. Name of the view to add as text. This is the name that will be displayed to users on the web page. It is, in essence, a table name.
2. Pointer to text array of names for each column of data. This is a pointer to a text array that contains the names of the columns for the view. For instance, "Name", "Address", "Phone Number", etc. It is, in essence, the field names.
3. Pointer to longint array of types for each column of data. This is a pointer to a longint array that contains the data types for the columns in the view. You use the standard 4D data types; for instance, Is Text, Is Longint, etc.
4. Pointer to a boolean array to set the unique attribute.
5. Pointer to a boolean array to set the mandatory attribute.
6. Pointer to a boolean array to set the non-enterable attribute.
7. Pointer to a boolean array to set the non-modifiable attribute.

You can add as many custom views as you like. Just make one call to DAX_Dev_DCS_AddCustomView for each view you are adding.

Valid Data Types

The following is a list of the valid data types you may use in the types array. The types serve two purposes. The first is to let 4D Ajax Framework know what type of value it is working with so it can make the proper conversions when necessary (using String, Num, Date, etc.). The second is to let the 4D Ajax Framework web front-end know what type of display to use for field input. For example, the type 'Is Text' will cause a multi-line text area input field to be displayed, while the type 'Is String Var' will cause a single line input field. 'Is Date' will cause a date entry calendar to be displayed. Numerical types will cause the 4D Ajax Framework front-end to allow only numerical values to be entered.

The type is for an individual item, so pass standard types not array types (i.e., don't use 'Text array' instead use 'Is Text').

```
Is Text
Is String Var
Is Real
Is Integer
Is LongInt
Is Date
Is Time
Is Boolean
Is Picture
```

Example:
This example creates a view to display Customer names and the totals from their Invoices. It is assumed we have a Customer table and a related Invoice table. This example continues in the other DCS method definitions.

```
C_TEXT($viewName_t)
C_BOOLEAN($added_b)
C_LONGINT($i)

ARRAY TEXT($names_at;3)
ARRAY LONGINT($types_al;3)
ARRAY BOOLEAN($unique_ab;3)
ARRAY BOOLEAN($mandatory_ab;3)
ARRAY BOOLEAN($nonEnterable_ab;3)
ARRAY BOOLEAN($nonModifiable_ab;3)

$viewName_t:="Customer Invoices" ` set the name of the view you are
adding

 ` set the names of the columns
$names_at{1}:="First Name"
$names_at{2}:="Last Name"
$names_at{3}:="Total"

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

 ` set the attributes of the columns
 ` for simplicity the example is setting all to false
For ($i;1;Size of array($unique_ab))
   $unique_ab{$i}:=False
   $mandatory_ab{$i}:=False
   $nonEnterable_ab{$i}:=False
   $nonModifiable_ab{$i}:=False
End for

$added_b:=DAX_Dev_DCS_AddCustomView ($viewName_t;->$names_at;-
>$types_al;->$unique_ab;->$mandatory_ab;
->$nonEnterable_ab;->$nonModifiable_ab)
```

DAX_DevHook_DCS_SetSelection

DAX_DevHook_DCS_SetSelection is called whenever 4D Ajax Framework needs the selection for a developer created view added through the DAX_DevHook_DCS_ViewAdd method. The name of the view that was set in DAX_DevHook_DCS_ViewAdd is passed in so the developer knows which records to load.

```
Passed:
$1       TEXT     Name of the DCS view to load the selection for
```

When referring to a "record," we mean any single set of data and not necessarily a record from a 4D table. A record can come from any data source as long as you can identify it uniquely.

Setting a custom view selection is done by calling DAX_Dev_DCS_SetSelection and passing two-n pointer parameters:

The first parameter to DAX_DCS_SetSelection must be a pointer to a longint array of record IDs. These IDs can be record numbers from a table, an ID or Key field's values created with Sequence Number or a similar function or any other method you have for creating a longint identifier.

Whatever method you use, the record ID must be a value that can find and load a particular record (data set) regardless of the current selection. Using a simple array element number system is not recommended and would most likely cause serious problems in a multi-user database.

The rest of the parameters to DAX_Dev_DCS_SetSelection are pointers to the arrays you will be using for the data. These arrays must be passed in the same order as the column names that were defined in DAX_DevHook_DCS_ViewAdd.

In other words, if the first element in the column names was "First Name" the first data array passed should be the one that contains first names. The type of arrays must be compatible with the types you set for the columns in DAX_DevHook_DCS_ViewAdd. Compatible means direct assignment between values is possible. So if you passed a type of 'Is Real' you cannot now pass a text array, but you can pass a longint or integer array. similarly, if you set the column type to 'Is Text' or 'Is String var' you can pass a text or string array interchangeably (assuming the string is long enough to hold your values).

Example:
This example continues from the example view that was created in DAX_DevHook_DCS_ViewAdd to display Customer names and the totals from their Invoices. This example continues in the other DCS method definitions.

```
Case of
   :($viewName_t="Customer Invoices")
        ` Declare the arrays
      ARRAY LONGINT(recordIDs_al;0)
      ARRAY REAL(totals_ar;0)
      ARRAY TEXT(firstNames_at;0)
      ARRAY TEXT(lastNames_at;0)

        ` Put the data into our arrays
      ALL RECORDS([Invoice])
      SELECTION TO ARRAY([Invoice];recordIDs_al;[Invoice]Total;totals_ar;
[Customer]FirstName;firstNames_at; [Customer]LastName;lastNames_at)

        ` Set the selection in DAX in the order defined when we created
the View
      DAX_Dev_DCS_SetSelection(->recordIDs_al;->firstNames_at;-
>lastNames_at;->totals_ar)

End case
```

DAX_DevHook_DCS_RecordSave

DAX_DevHook_DCS_RecordSave is called whenever 4D Ajax Framework is asked to save a DCS record. The name of the view that you set in DAX_DevHook_DCS_ViewAdd and the record id for the record to be saved are passed in. For modified records the record id is the id you passed in as the record id in DAX_DevHook_DCS_SetSelection; for new records the constant 'New record' is passed in.

If you save the record, return the record id, either the same one that was passed in (for modifying) or a new record id (for new records). If you do not save the record, return the constant 'No current record'.

```
Passed:
$1       TEXT       Name of the DCS View where the record is being saved
$2       LONGINT    Record ID for the record that is being saved ('New
                    record' constant for new records)
$3       POINTER    Pointer to text array of field names for the record
                    that is being saved
$4       POINTER    Pointer to text array of field values for the record
                    that is being saved

Returned:
$0         LONGINT  Flag: Developer Performed the Query
```

In the case of the ID being a record number, you can use Goto Record to load the record that needs to be updated. In the case of the ID being a unique sequence number from an ID field, you can query for the record that needs to be updated. In other cases it will depend on what the ID means to you as the developer.

Example:
This example continues from the example view that was created in DAX_DevHook_DCS_ViewAdd to display Customer names and the totals from their Invoices. This example continues in the other DCS methods.

```
Case of

   :($viewName_t="Customer Invoices")
      If($recordNumber_l=New record)
         CREATE RECORD([Invoice])

         $find_l:=Find in array($addRecFieldName_p->;"First Name")
         $firstName_t:=$addRecFieldValue_p->{$find_l}

         $find_l:=Find in array($addRecFieldName_p->;"Last Name")
         $lastName_t:=$addRecFieldValue_p->{$find_l}

         QUERY([Customer];[Customer]FirstName=$firstName_t;*)
         QUERY([Customer]; & ;[Customer]LastName=$lastName_t)
         [Invoice]CustomerID:=[Customer]ID

      Else
         READ WRITE([Invoice])
         GOTO RECORD([Invoice];$recordNumber_l)

         If(Records in selection([Invoice])#1)
            $recordNumber_l:=No current record
            UNLOAD RECORD([Invoice])
         End if
      End if

      If($recordNumber_l#No current record )
         $find_l:=Find in array($addRecFieldName_p->;"Total")
         [Invoice]Total:=$addRecFieldValue_p->{$find_l}
         SAVE RECORD([Invoice])
         $recordNumber_l:=Record number([Invoice])
         UNLOAD RECORD([Invoice])
      End if

End case

$0:=$recordNumber_l
```

DAX_DevHook_DCS_RecordDelete

DAX_DevHook_DCS_RecordDelete is called whenever 4D Ajax Framework is asked to delete a DCS record. The name of the view that you set in DAX_DevHook_DCS_ViewAdd and the record id for the record to be deleted are passed in. The record id is the id you passed in as the record id in DAX_DevHook_DCS_SetSelection. If you delete the record, return True. If the record is not deleted (denied), return False.

```
Passed:
$1       TEXT       Name of the DCS View where the record is being deleted
$2       LONGINT    Record ID for the record that is being deleted

Returned:
$0         BOOLEAN  True=Deleted, False=Not deleted and send an error
```

The record id is the id you passed in as the record id in DAX_DevHook_DCS_SetSelection. The developer must use this ID to load the data that it represents and act accordingly.

In the case of the id being a record number, you can use Goto Record and delete the record. In the case of the id being a unique sequence number from an id field, you can query for the record and delete the record. In other cases it will depend on what the id means to you as the developer.

Example:
This example continues from the example view that was created in DAX_DevHook_DCS_ViewAdd to display Customer names and the totals from their Invoices. This example continues in the other DCS methods.

```
$deleteRecordAccepted_b:=True

Case of

   :($viewName_t="Customer Invoices")
      If ($recordID_l>No current record )
         READ WRITE([Invoice])
         GOTO RECORD([Invoice];$recordID_l)
      End if

      If (Records in selection([Invoice])=1)
         DELETE RECORD([Invoice])
      Else
         $deleteRecordAccepted_b:=False
         UNLOAD RECORD([Invoice])
      End if

End case
```

```
$0:=$deleteRecordAccepted_b
```

DAX_Dev_DCS_Delete

DAX_Dev_DCS_Delete should be called any time you wish to remove a DCS that was previously added in DAX_DevHook_DCS_ViewAdd. The name of the DCS View you wish to delete is passed in. The DCS is removed from the XML files on disk and from memory. **Note:** If you no longer want the DCS to exist at all be sure to first disable your code that creates the DCS. Otherwise the DCS will be recreated at the next launch.

Call Syntax: `DAX_Dev_DCS_Delete(DCS Name)`

```
Passed:
$1        TEXT        DCS Name
```

Implementation

Simply place a call to the method within any other method and execute it.

Example

```
` Method: Temporary Method
` Executed within 4D to remove a DCS
DAX_Dev_DCS_Delete("MyDCSView")
```

Developer Defined Windows (DDW)

Developer Defined Windows (DDW) give developers the ability to add a window with any HTML content they want on the front-end. The developer can have the window display an HTML blob or content from any valid URL. A DDW can display anything that can be displayed in a web browser.

A DDW can be created at the portal level, selection level or field level. A portal level DDW is displayed in the left sidebar of the main window. A selection level DDW is displayed at the top of a window which displays a selection of records with the Create, Delete, etc. buttons. For selection level a button is also added to the detail view next to the Save and Cancel buttons. A field level DDW causes the field data to display as a link in a list of records and as a button next to the field value in the detail views

Adding a DDW view is done by calling DAX_Dev_DDW_Create and passing four parameters.

1. Object Type as Text. This is the type of object that will be displayed to users on the web page. Valid values are either "button" or "link".
2. Object Title as Text. This is the title that will be displayed on the button or as the link.
3. Method or URL as Text. This can be one of two things. The first use is the name of the method to execute when the user clicks on the button or link. In this case the method must return a blob that contains HTML content to be displayed. The second use is a valid URL that points to a resource to be loaded into the window (i.e., "http://www.4d.com" or "/4dcgi/MyWebMethod&parameter=value").
4. Associated To as Text. This is the type of object with which the DDW will be used. Valid values are "Portal" or "Other". This parameter is optional and will default to "Other" if it is not provided.

Examples:
You can add as many DDW views as you like. Just make one call to DAX_Dev_DDW_Create for each DDW view you are adding.

Adding a Portal

```
$ddwID_l:=DAX_Dev_DDW_Create("Link";"4D Home Page";
"http://www.4d.com/";"Portal")
```

Creating a DDW and assigning it to multiple fields

```
$ddwID_l:=DAX_Dev_DDW_Create("Link";"Map It";"MyMapMethod";"Other")
DAX_Dev_DDW_AssignToObject($ddwID_l;"Field";Table name(->[Contact]);Field
name(->[Contact]Address))
DAX_Dev_DDW_AssignToObject($ddwID_l;"Field";Table name(->[Company]);
Field name(->[Company]Address))
```

Creating a DDW and assigning it to a Selection

```
$ddwID_l:=DAX_Dev_DDW_Create("Link";"Help";"MyHelpMethod")
DAX_Dev_DDW_AssignToObject($ddwID_l;"Selection";Table name(->[Contact]))
```

In the above examples, MyHelpMethod might load a local HTML help file into a blob and then return the blob. Or, MyMapMethod could use the TCP commands to download map information from another site and then return the HTML in a blob.

Note: A URL can be specified instead of method. The specified URL must either be absolute (http://www.domain.com/...) or relative (/4DCGI/Commands...).

Once a DDW is created, it can be used in one of the following:

Selection

1. The DDW option appears at the Toolbar level in the Grid (List) window)
2. The DDW option appears at the Button Set (Save and Cancel) level.

Field

The DDW option appears next to the field object.

ALL DDW ID HAVE A STARTING RANGE OF 100001 AND UP.

DAX_Dev_DDW_GetID

This method allows the developer to programmatically retrieve a DDW ID using a DDW Name or DDW Title.

Passed:

$1	TEXT	DDW Title or Name (e.g. DDW_1, DDW_2, ..., DDW_[n])
$2	LONGINT	Mode (0 or Omitted for Get-By-Title, 1 or more for Get-By-Name)

Returned:

$0	LONGINT	DDW ID

DAX_Dev_DDW_GetTitle

This method allows the developer to programmatically retrieve a DDW Title using a DDW ID.

Passed:

$1	LONGINT	DDW ID

Returned:

$0	TEXT	DDW Title

DAX_Dev_DDW_GetState

This method allows the developer to programmatically retrieve the state of a DDW.

Here is a list of possible states that can be returned.

- State = 1 (The DDW exists and can be executed)
- State = 0 (The DDW does not exist)
- State = -1 (The DDW was deleted)

DAX_Dev_DDW_GetList

This method retrieves the information for all DDWs and populates the information into the arrays.

Passed:

$1	LONG	Pointer to an array for DDW ID
$2	TEXT	Pointer to an array for DDW Name
$3	TEXT	Pointer to an array for DDW Type
$4	TEXT	Pointer to an array for DDW Title
$5	TEXT	Pointer to an array for DDW Content
$6	TEXT	Pointer to an array for DDW Assigned To
$7	LONG	Pointer to an array for DDW State (1 for exists and enabled, -1 for deleted)

DAX_Dev_DDW_GetAttributes

Retrieve one or more DDW attributes and related information about the current DDW call.

```
Syntax:
DAX_Dev_DDW_GetAttributes(AttrName;PointerToVar{;PointerToVar{;PointerToVar...}
})
```

```
Passed:
$1        TEXT       Name of the attribute
$2        POINTERS   Pointer to the following variables
                     1. Pointer to a TEXT variable for AttrName="Type"
                     2. Pointer to a TEXT variable for AttrName="TableName"
                     3. Pointer to a LONGINT variable for
                     AttrName="FieldType"
                     4. Pointer to a TEXT variable for AttrName="FieldData"
                     5. Pointer to a LONGINT ARRAY variable for
                     AttrName="RecordNumbers"

Returned:
$0          POINTER  Returns Pointer to the posted table only if AttrName is
                     "TablePointer". Otherwise NULL.
```

For example:

```
Retrieve the DDW type:
DAX_Dev_DDW_GetAttributes("Type";->$Type_t)

Retrieve the table name:
DAX_Dev_DDW_GetAttributes("TableName";->$TableName_t)

Retrieve the pointer to the physical table:
$TablePointer_p:=DAX_Dev_DDW_GetAttributes("TablePointer")

Retrieve the type of the posted field:
DAX_Dev_DDW_GetAttributes("FieldType";->$FieldType_l)

Retrieve the data of the posted field
DAX_Dev_DDW_GetAttributes("FieldData";->$FieldData_t)

Populate the all record number into the arrays:
DAX_Dev_DDW_GetAttributes("RecordNumbers";->$Recordnumber_al)
```

The "RecordNumbers" tag returns an array that contains record numbers for highlighted records in the selection. If no record has been selected, the selection is considered the records they are viewing. This array will contain all record numbers for that selection.

DAX_Dev_MimeType

Get or Set the mime type for an http return. It should generally only be used within a DDW return method.

```
Passed:
$1        TEXT       Set the mime type for the http return

Returned:
$0          TEXT     Get the mime type for the http return
```

DAX_Dev_DDW_Delete

New for version 11.2, this method deletes a DDW based on the given DDW ID

```
Call Syntax:   DAX_Dev_DDW_Delete (DDW_ID{;CompletelyDeleted)
```

```
Passed:
```

$1	LONGINT	DDW ID of 100001 or higher
$2	BOOLEAN	TRUE is to delete the DDW if it is active or marked as deleted False is to delete the DDW if it is active.

Please note that this method should only be executed anytime after the method DAX_Dev_Initialize has been executed.

Tips: How to Ignore Backend Errors

In some cases, you may need to ignore the backend error (such as InternalError) to avoid an interuption of the frontend. This can be done by completing the following steps:

1. Assign 0 to a variable named "Error"
   e.g.   Error:=0

2. Assign an empty string ("") to the internal DAX error
   DAX_Dev_Session_Error("")

Using a DDW with Portal view, Selection and field

Portal View: This process happens when a you create a DDW. Once you have set a DDW to be associated to "Portal", it will automatically become a new portlet view in the sidebar.

Selection: Using a DDW in a selection means that the DDW option will be available at the top of the Grid (List) Window and at the bottom of the Editor (Detail) Window as a button next to the Save and Cancel buttons. To use a DDW in a selection, you must pass an appropriate information to the backend. Some are mandatory and some aren't. Here they are:

DDW for	Virtual Table ID	Virtual Field ID
Physical table	Mandatory	Optional
View	Mandatory	Mandatory
DCS	Mandatory	Optional

Important: If a Virtual Field ID is selected for Physical or DCS, you can obtain all values from that field when the DDW method is executed by calling the command:

`DAX_Dev_DDW_GetAttributes.`

Field: Setting a DDW to a field requires the Virtual Table ID and Field ID to be passed to the backend. The format is the following:

```
Virtual Table ID   (+/-) Numeric
Field ID           [Physical Table ID][Physical Field ID], e.g. [2][4]
```

Once a DDW is set to a field, the DDW option will appear in 2 locations:

1. In the Grid (List) window as the link in the field column. The text that appears to in the Grid will appear as a clickable-link.
2. In the Editor (Detail) window as a button next to the field.

Defining DDWs From Administration Dialog

From the Administration dialog, you can create a DDW and choose different options:

You can display a static HTML page or refer a static link to another web site:

> Select 'New Window - Link' and enter the URL in the Contents area.
> Choose 'Portal' if you want the URL to be available from the Portal list.
> Choose 'Others' if you want the URL to be available from a selection or a field.

You can display a dynamic HTML page or dynamic link to another web site, returned by a method:

> Select 'New Window - HTML method' and enter the name of the method in the Contents area.
> This method must return the HTML code in $0. The type of $0 will be Blob.
> Choose 'Portal' if you want the URL to be available from the Portal list.
> Choose 'Others' if you want the URL to be available from a selection or a field.

You can display a message at the bottom of the current selection or record window:

> Select 'Status Message - Text Method' and define a method name in the Contents area.
> This method must return a text message in $0.
> You can only choose 'Others'. The DDW will be available for objects (table, field..).
> This feature cannot be used with 'Portal'.

You can now assign your DDWs with the 'Others' setting to your selections or your fields. DDWs created with 'Portal' will automatically be displayed in the Portal window.

Backend Responsibility

When the command GetDDW is called, the backend will attempt to load all posted and related attributes before executing the method set to the DDW. Here are some of the attributes that can be access by the developers:

Type	TEXT	The type of the DDW that the frontend posted to the backend. (Selection, Field)
TableName	TEXT	Name of the Table
TablePointer	POINTER	Pointer to the table; Null if DCS
FieldData	TEXT	The posted data in a given field (Always returned in TEXT)
RecordNumbers	POINTER	Pointer to arrays to be populated with data record number
ViewArray	POINTER	(OBSOLETE in 1.1) Pointer to an array to get populate with the data from the field set in DDW. You should only attempt to populate this array when the Type is "Selection" and the current access table is a View (Virtual Table).

In the DDW method, the above attributes can be access by calling the command DAX_Dev_DDW_GetAttributes.

About Your DDW Method

When a DDW is executed, the method that associated with DDW will be executed. This method is called DDW method. The method will be executed only when the /DAX/GetDDW... request is sent to the backend. It is up to the developer to perform appropriate action inside the DDW method. Often you will use DDW method to create a dynamic Web content or URL. Here are some examples of what you can do inside a DDW method:

Create Dynamic Web Content

```
` DDW Method: Customer_info
` Description: Base on the customer id, find the customer and compose an
`   HTML layout with his or her information and send it to the Web
Frontend
C_BLOB($0)
C_TEXT($info_t)
ARRAY LONGINT($RecordNumbers_al;0)
DAX_Dev_DDW_GetAttributes ("RecordNumbers";->$RecordNumbers_al)
If (Size of array($RecordNumbers_al)=1)
  GOTO RECORD([Company];$RecordNumbers_al{1})
  $Info_t:="<b>"+[Company]Name+"<b><br>"
  $Info_t:=$Info_t+"  <i>"+[Company]Address+"<br>"
  $Info_t:=$Info_t+"  <i>"+[Company]City+", "
  $Info_t:=$Info_t+[Company]State+" "+[Company]ZipCode+"</i>"
Else
  $Info_t:="<b>The requested company does not exist in the database.</b>"
End if
TEXT TO BLOB($Info_t;$0;Text without length )
```

In this example, the DDW method named Customer_info is installed to the field named [Company]Name.

```
` Component method: DAX_DevHook_DDW_Install
$ddwID_l:=DAX_Dev_DDW_Create ("LinkStatic";"More
Info";"Customer_info";"Other")
DAX_Dev_DDW_AssignToObject ($ddwID_l;"Selection";Table name(-
>[Company]);Field name(->[Company]Name))
```

When the Web frontend display the selection, the customer name will appear as a link.

When the user clicks on the link, the /DAX/GetDDW is made to the backend. As the result, the method Customer_info is executed.

Here is an example of the result:

James Williams
9128 Benton Street
Santa Clara, CA 98172

Create a Dynamic Web URL

```
` DDW Method: Map_Address
` Description: Base on the customer id, find the customer and compose a
Google
`   map URL for the customer's address and return to the frontend.
C_TEXT($0;$saddr_t;$daddr_t)
ARRAY LONGINT($RecordNumbers_al;0)
DAX_Dev_DDW_GetAttributes ("RecordNumbers";->$RecordNumbers_al)
If (Size of array($RecordNumbers_al)=1)
   GOTO RECORD([Company];$RecordNumbers_al{1})
   $saddr_t:="saddr=3031+Tisch+Way,+San+Jose,+CA+95128"

$daddr_t:="daddr="+[Company]Address+","+[Company]City+","+[Company]State+
[Company]ZipCode
End if
$0:="http://maps.google.com/maps?"+$saddr_t+"&"+$daddr_t

In this example, the DDW method named Map_Address is installed to the
field named [Company]Name.

` Component method: DAX_DevHook_DDW_Install
$ddwID_l:=DAX_Dev_DDW_Create
("LinkDynamic";"MapIt";"Map_Address";"Other")
DAX_Dev_DDW_AssignToObject ($ddwID_l;"Selection";Table name(-
>[Company]);Field name(->[Company]Name))
```

The frontend will received the Google Map URL where it can be loaded anywhere in the frontend.

Frontend Commands

Command: AdminCreateDDWInfo

/DAX/AdminCreateDDWInfo

This command allows a DDW to be created by the frontend. This command should be called by the Admin connection only.

Parameters

Name	Type	Example Value
Type	TEXT	"LinkDynamic" or "LinkStatic"
Title	TEXT	Title/Label that will appear when displaying the DDW object.
Method	TEXT	Method name or URL (Absolute or Relative)
AssociatedTo	TEXT	"Portal" for displaying the DDW as a new portlet, "Other" or omitted for allowing the DDW to be assigned to other objects.

More about Type

If Frontend set:	**Type=Link, Content=URL**

Must send to Backend:	Type=LinkStatic&Method=MethodName&Title=MyTitle&
	  AssociatedTo=Other or
	Type=LinkStatic&Method=MethodName&Title=MyTitle&
	  AssociatedTo=Portal

If Frontend set:	Type=Link, Content=MethodName (a method that generates a URL)

```
Must send to  Type=LinkDynamic&Method=MethodName&Title=MyTitle&
Backend:        AssociatedTo=Other or
              Type=LinkDynamic&Method=MethodName&Title=MyTitle&
                AssociatedTo=Portal


If Frontend   Type=Link, Content=MethodName (a method that generates an HTML
set:          content/blob)
Must send to  Type=LinkStatic&Method=MethodName&Title=MyTitle&
Backend:        AssociatedTo=Other or
              Type=LinkStatic&Method=MethodName&Title=MyTitle&
                AssociatedTo=Portal


If Frontend   Type=Web Content, Content=MethodName (a method that generates
set:          an HTML content/blob)
Must send to  Type=LinkStatic&Method=MethodName&Title=MyTitle&
Backend:        AssociatedTo=Other or
              Type=LinkStatic&Method=MethodName&Title=MyTitle&
                AssociatedTo=Portal


If Frontend   Type=Message, Content=MethodName (a method returns a text
set:          message)
Must send to  Type=Message&Method=MethodName&Title=MyTitle&
Backend:        AssociatedTo=Other or
              Type=Message&Method=MethodName&Title=MyTitle&
                AssociatedTo=Portal
```

Important!!! All specified URL in a DDW for a Dynamic type is automatically converted to static. The reason is that the backend needs a method to compose a Dynamic URL. If a URL is specified in the Method parameter, then there will not be a method to generate URL.

For example

1. Create a DDW as a new portlet

```
Type              "LinkDynamic"
Title             "My New Portlet"
Method            "My4DMethod"
AssociatedTo      "Portal"
```

```
http://localhost/DAX/AdminCreateDDWInfo?sessionId=S9222006150444CCWB&Type=LinkD
ynamic&Title=My New Portlet&Method=My4DMethod&AssociatedTo=Portal
```

2. Create a DDW to be assign for other objects

```
 Type          =
 Title         =
 Method        =
 AssociatedTo  =
Type              "LinkStatic"
Title             "Report"
Method            "mGenReport"
AssociatedTo      "Other"
```

```
http://localhost/DAX/AdminCreateDDWInfo?sessionId=S9222006150444CCWB&Type=LinkS
tatic&Title=Report&Method=mGenReport&AssociatedTo=Other
```

Reply

```
<CreateDDWInfo>
   <DDW ddwid="100001" name="DDW_1" type="LinkStatic" title="Report"
method="mGenReport" associatedto="Other" />
</CreateDDWInfo>
```

Command: AdminModifyDDWInfo

/DAX/AdminModifyDDWInfo

This command allows the frontend to modify one or more information in a specific DDW. This command should be called by the Admin connection only.

Parameters

Name	Type	Example Value
ddwid	NUMERIC	DDW ID (Required)
name	TEXT	Name of the DDW
Type	TEXT	"LinkDynamic" or "LinkStatic"
Title	TEXT	Title/Label that will appear when displaying the DDW object.
Method	TEXT	Method name or URL (Absolute or Relative)
AssociatedTo	TEXT	"Portal" for displaying the DDW as a new portlet, "Other" or omitted for allowing the DDW to be assigned to other objects.

For example

1. Modify the name of a DDW and its object title.

ddwid	**100001**
Name	"Special Report"
Title	"Company Report"

```
http://localhost/DAX/AdminModifyDDWInfo?sessionId=S9222006150444CCWB&ddwid=1000
01&name=Special Report&Title=Company Report
```

Reply

```
<CreateDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkStatic"
title="Company Report" method="mGenReport" associatedto="Other" />
</CreateDDWInfo>
```

Command: AdminDeleteDDW

/DAX/AdminDeleteDDW

This command allows the frontend to delete a DDW.

Parameters

Name	Type	Example Value
ddwid	NUMERIC	DDW ID (Required)

For example

```
http://localhost/DAX/AdminDeleteDDW?sessionId=S9222006150444CCWB&ddwid=100001
```

Reply

```
<AdminDeleteDDW>
   <Status Success="True" />
</AdminDeleteDDW>
```

Command: AdminSetDDWToObject

/DAX/AdminSetDDWToObject

This command allows the frontnd to assign a DDW to an object. Objects can either be "Selection" or "Field" level (type).

Parameters

Name	Type	Example Value
ddwid	TEXT	DDW ID (Required)
type	TEXT	"Selection" or "Field"
objectref	NUMERIC	Object ID
subobjectref	[NUMERIC][NUMERIC]	[Table ID][Field ID] (required when assigning a DDW to a field or a View Selection)

When sending only the object id, the backend will assume that the frontend wants to assign the DDW to a Selection level of the given object. If both object and subobject id (in the format of [Object ID][Subobject ID]) are sent, the backend will assign the DDW to the field level of the given object ids.

Note: [Object ID] can be a physical table, view or DCS id

```
[Subobject ID] is a field id of physical table, view or DCS
```

For example

1. Set a DDW to a Selection level of a Physical Table (ID = 1)

ddwid	**100001**
type	"Selection"
objectref	1

```
http://localhost/DAX/AdminSetDDWToObject?sessionId=S9222006150444CCWB&ddwid=100001&type=Selection&objectref=1
```

Reply

```
<SetDDWToObject>
   <Status Success="True" />
</SetDDWToObject>
```

2. Set a DDW to a Selection level of a View (ID = 32001)

ddwid	**100001**
type	"Selection"
objectref	32001

```
http://localhost/DAX/AdminSetDDWToObject?sessionId=S9222006150444CCWB&ddwid=100001&type=Selection&objectref=32001
```

Reply

```
<SetDDWToObject>
```

```
   <Status Success="True" />
</SetDDWToObject>
```

3. Set a DDW to a Selection level of a View (ID = -1).

Because the structure of the view consists of fields from multiple tables, a field must be sent with the request. The field id will be used later to load all values of the field into a text array when the GetDDW is called. Please the GetDDW section for more information.

ddwid	**100001**
type	"Selection"
objectref	-1
subobjectref	[3][4]

```
http://localhost/DAX/AdminSetDDWToObject?sessionId=S9222006150444CCWB&ddwid=100
001&type=Selection&objectref=-1&subobjectref=[3][4]
```

Reply

```
<SetDDWToObject>
   <Status Success="True" />
</SetDDWToObject>
```

4. Set a DDW to a Field level

ddwid	**100001**
type	"Field"
objectref	1
subobjectref	[1][4]

```
http://localhost/DAX/AdminSetDDWToObject?sessionId=S9222006150444CCWB&ddwid=100
001&type=Field&objectref=1&subobjectref=[1][4]
```

Reply

```
<SetDDWToObject>
   <Status Success="True" />
</SetDDWToObject>
```

Command: GetDDWInfo

/DAX/GetDDWInfo

This command returns the Information about a DDW or DDW that is assigned to a selection or field.

Parameter

Name	Type	Example Value
type	TEXT	"DDW", "Portal", "Other", "Selection", "Field" or "All" (Required)
objectid	NUMERIC	DDW or Physical/View/DCS (Not required when type is Portal)
subobjectid	[NUMERIC][NUMERIC]	[Table ID][Field ID] (Not required when type is Portal)

For example

1. Get information for a DDW

```
type        DDW
objectid    100001
```

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=DDW&objectid=
100001
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkDynamic"
title="Company Report" method="mGenReport" associatedto="Other" />
</GetDDWInfo>
```

2. Get all DDW that are set to Portal

```
type        Portal
```

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Portal
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100004" name="MyLocations1" type="LinkDynamic"
title="California" method="mShowLocationCA" associatedto="Portal" />
   <DDW ddwid="100007" name="MyLocations2" type="LinkDynamic" title="New
York" method="mShowLocationNY" associatedto="Portal" />
</GetDDWInfo>
```

3. Get all DDW that are set to Other (can be assigned to other objects)

```
type        Other
```

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Other
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkDynamic"
title="Company Report" method="mGenReport" associatedto="Other" />
   <DDW ddwid="100002" name="Drive" type="LinkDynamic" title="Driving
Direction" method="mGetMap" associatedto="Other" />
</GetDDWInfo>
```

4. Get information for a Selection

```
type        Selection
objectid    1 (Physical ID = 1) or  -1 (View ID = -1) or 32001
            (DCS ID = 32000)
```

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=1

http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=-1

http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=32000
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkStatic"
title="Company Report" method="mGenReport" associatedto="Other" />
</GetDDWInfo>
```

5. Get information for a Field

Part of Physical Table Portlet:

```
    =
    =
    =
```

type	**Selection**
objectid	1 (Physical ID = 1)
subobjectid	[1][3]

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=1&subobjectid=[1][3]
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkDynamic"
title="Company Report" method="mGenReport" associatedto="Other" />
</GetDDWInfo>
```

Part of View Portlet:

type	**Selection**
objectid	-1 (View ID = -1)
subobjectid	[5][3]

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=1&subobjectid=[5][3]
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkDynamic"
title="Company Report" method="mGenReport" associatedto="Other" />
</GetDDWInfo>
```

Part of DCS Portlet:

type	**Selection**
objectid	32001 (DCS ID = 32001)
subobjectid	[32001][3]

```
http://localhost/DAX/GetDDWInfo?sessionId=S9222006150444CCWB&type=Selection&obj
ectid=1&subobjectid=[32001][3]
```

Reply

```
<GetDDWInfo>
   <DDW ddwid="100001" name="Special Report" type="LinkDynamic"
title="Company Report" method="mGenReport" associatedto="Other" />
</GetDDWInfo>
```

Command: GetDDWOption

/DAX/GetDDWOption

This command retrieve a DDW option for the Selection level. It should be called when the frontend wants to display a window (Grid/D-Tree/...) or the Editor. The return information will allow the frontend to determine whether it has a DDW option set to the current Selection or Record view.

Parameter

Name	Type	Example Value
objectid	NUMERIC	Physical/View/DCS ID

For example

```
http://localhost/DAX/GetDDWOption?sessionId=S1020200695349OOLN&objectid=4
```

Reply

```
<GetDDWOption>
  <DDW title="Click Me" type="LinkStatic"
url="http://localhost/DAX/GetDDW?sessionid=S1020200695349OOLN&ddwid=10000
3&type=Selection"/>
</GetDDWOption>
```

Command: GetDDW

/DAX/GetDDW

This command executes the DDW method from the backend and returns the HTML result back to the frontend.

Parameter

Name	Type	Example Value
ddwid	NUMERIC	DDW ID
tableid	NUMERIC	Physical, View, DCS ID (+/-)
queryid	TEXT	Query ID
type	TEXT	"Selection" or "Field" (Optional)
data	TEXT	data in the field (Optional)
recordid	[NUMERIC][NUMERIC] or [NUMERIC][NUMERIC][NUMERIC]	TABLE ID][RECORD ID] or [TABLE ID][FIELD ID][RECORD ID] (Optional)

For example

DDW 1 is set to execute a method name mHelloWorld. This method returns the following HTML result

```
<html>
  <body><i>Hello World</i></body>
</html>

   ddwid  =  1

http://localhost/DAX/GetDDW?sessionId=S9222006150444CCWB&ddwid=1

Reply
<html>
  <body><i>Hello World</i></body>
```

```
</html>
```

Frontend behavior when calling a DDW

Portlet Type: Physical Table/View/DCS
DAX Form: Grid/List Form
DDW Location: Toolbar
DDW Used By: Selection
DDW Appears As: Button/Link

Parameters that need to be passed in the URL:

ddwid	**1, 2, 3,..., N**
type	"Selection"
queryid	Query id of the current selection
tableid	1

For example:

```
  /DAX/GetDDW?ddwid=1&type=Selection&queryid=SCU234559900000000&tableid=1
```

Portlet Type: Physical Table/View/DCS
DAX Form: Editor
DDW Location: Toolbar
DDW Used By: Selection
DDW Appears As: Button/Link

Parameters that needs to be passed in the URL:

ddwid	**1, 2, 3,..., N**
type	"Selection"
tableid	1
recordid	[Table ID][Record ID]Record ID (0 or higher for existing record, -3 for new record)

For example:

```
/DAX/GetDDW?ddwid=1&type=Selection&recordid=[1][1]&tableid=1
```

Portlet Type: Physical Table/View/DCS
DAX Form: Editor/Detail Form
DDW Location: Next to a field
DDW Used By: Field
DDW Appears As: Button/Link

Parameters that needs to be passed in the URL:

ddwid	**1, 2, 3,..., N**
type	"Field"
data	Current value in the field

For example:

```
/DAX/GetDDW?ddwid=1&type=Field&data=95118
```

Portlet Type: Physical Table/View/DCS

DAX Form: Grid/List Form
DDW Location: Field Column
DDW Used By: Field
DDW Appears As: Link (in the field column)

If the DDW is set with a method name, the front end will automatically get the link for each record. This link can then be used to compose a <a href...> for each row of the field column.

```
http://www.domain.com

http://www.domain.com/DAX/DAX_GetDDW?ddwid=1

http://www.domain.com/4DCGI/....

/4DCGI/MyCommand...
```

Creating Callbacks

Callbacks are a way for 4D to respond to certain events that occur in a field on the front-end. When an event is triggered, the front-end will post the event information and field value to the 4D. 4D will then execute the method that is set to handle the event and return a response back to the front-end. Currently 4D Ajax Framework supports two events from the front-end: On Load (event id=1) and On Data Change (event id=20).

1. On Load - The field object is about to be drawn in the Detail View.
2. On Data Change - The user tabbed out of the field.

DAX_DevHook_InstallCallBack

Adding a callback is done by calling DAX_Dev_SetCallBack and passing five parameters.

1. Event ID as Longint. This is the event that the callback will handle. Currently On Load and On Data Change are supported.
2. Table ID as Longint. This is the table number for the table that the callback will be installed
3. Field ID as Longint. This is the field number for the field for which the callback will be installed.
4. Method Name as Text. This is the name of the method that will be executed when the event triggers.
5. Modifiable as Boolean. This determines whether or not the callback can be modified via the Admin interface on the front-end.

Method to be executed on a Callback

The method that is set to respond to a callback must return a Text value in $0 to be used as the new value for the field that triggered the event. 4D Ajax Framework callback methods have access to eight attributes when they run:

1. Event ID as Longint. This is the event that triggered the callback.
2. Table ID as Longint. This is the table number for the edited field.
3. Table Name as Text. This is the table name for the edited field.
4. Field ID as Longint. This is the field number for the edited field.
5. Field Name as Text. This is the field name for the edited field.
6. Record ID as Longint. This is the record number (-3 for new records).
7. Value as Text. This is the value the user entered for the edited field.
8. Message as Text. Text message that will be returned to the front-end.

To get the value of these attributes the developer calls DAX_Dev_GetCallBackVar passing the following parameters:

1. Name of Attribute as Text. This is the name of the attribute the developer wants to retrieve. Valid attributes are “Event ID”, “Table ID”, “Field ID”, “Record ID”, “Value, “Message”.
2. Pointer to a Variable. This is a pointer to the variable that will receive the attribute value.

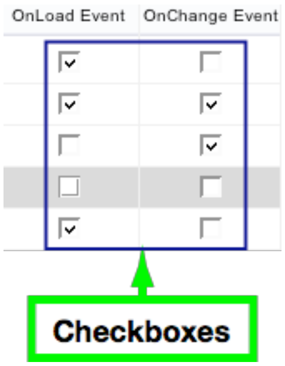
To set the “Message” attribute value the developer calls DAX_Dev_SetCallBackVar passing the following parameters:

2. Name of Attribute as Text. This should be “Message”.
3. Pointer to a Variable. This is a pointer to the variable that contains the message value.

The message attribute is displayed at the bottom of the detail window. For example, if you are doing email validation the message might be set to "Invalid Email Address."

New Callback Implementation

4D Ajax Framework version 1.2 introduced a new Callback implementation. In the Control Panel, there are now checkboxes for fields regarding the On Load and On Data Change events (as opposed to the enterable fields where methods could be specified for these events.) This means that all Callback events are handled in Dax_DevHook_CB_EventFired (details below).

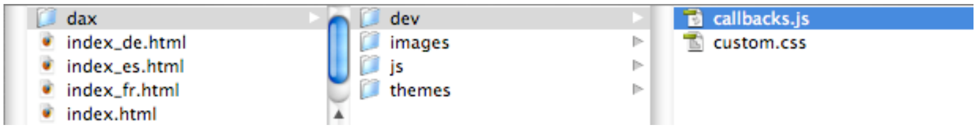


This new implementation now gives the developer control of all the displayed fields in the input form. For instance, as users enter information in one field other fields can be populated with information based on the input.

The previous of implementation of the Callbacks is deprecated but still supported, as the 4D Ajax Framework is backwards compatible. However, developers are encouraged to transition over to the new implementation.

Javascript Callbacks

Callbacks.js (in /dax/dev/) allows developers to append their own Javascript code during certain events in the client via Javascript Callbacks.



Javascript Callbacks are already an existing feature in the framework, but they are now made more easily available in the /dax/dev/ folder to encourage more use of it.

The following events are currently supported in Javascript Callbacks:

- On Editor Pull
- On Query
- After Successful Record Save
- After Successful Login
- Before Record Save

Back End Methods

DAX_DevHook_CB_Install

DAX_DevHook_CB_Install is used by the developer to assign a callback to a field. A callback can be set for the events On Load or On Data Change. The BE is called every time one of these events occur on a field with a callback assigned.

To assign a callback to a field you place a call to DAX_Dev_CB_Install in DAX_DevHook_CB_Install and pass three (3) parameters.:

1. Event ID - The event that should trigger a callback. You can pass the 4D constants On Load or On Data Change.

2. The name of the selection (Table, View, DCS) that the event will be assigned to.

3. The name of the field that the event will be assigned to.

Example

```
DAX_Dev_CB_Install (On Data Change ;Table name(4);Field name(4;3))
```

DAX_DevHook_CB_EventFired

DAX_DevHook_CB_EventFired is used by the developer to modify data or take some other action when a callback event assigned to a field fires. A callback can be set for the events On Load or On Data Change. The BE is called every time one of these events occur on a field with a callback assigned.

You call DAX_Dev_CB_GetInfo to get various information about the record that is being edited and the event that has fired. Available values are:

- "selection name" - Name of the selection that is being edited
- "field name" - Name of the field that is being edited
- "event id" - Event that has fired
- "selection id" - ID of the selection that is being edited
- "field id" - ID of the field that is being edited
- "record id" - Record number that is being edited
- "field value" - Current value for the field that is being edited

Example

```
$eventID_l:=Num(DAX_Dev_CB_GetInfo ("event id"))
$selectionName_t:=DAX_Dev_CB_GetInfo ("selection name")
$fieldName_t:=DAX_Dev_CB_GetInfo ("field name")
$fieldValue_t:=DAX_Dev_CB_GetInfo ("field value")

  ` determine which event is firing
Case of
  : ($eventID_l=On Data Change )
    ARRAY TEXT($fieldNames_at;1)
    ARRAY TEXT($fieldValues_at;1)

    $fieldNames_at{1}:=$fieldName_t
    $fieldValues_at{1}:=Uppercase($fieldValue_t)
    ` tell the Front-end to update the value
    DAX_Dev_CB_SetFieldValues (->$fieldNames_at;->$fieldValues_at)
    DAX_Dev_CB_SetStatus (1)  ` Set Callback OK status
    ` in this example we act only if they are editing the Contact table
    If ($selectionName_t="Contact")
      ` determine which field they are editing
      Case of

        : ($fieldName_t="First_Name") | ($fieldName_t="Last_Name")
          ` Here the user has entered a name on the front-end
          ` we are now going to normalize the case for the name

          ` A method that normalizes case
          $fieldValue_t:=My_Normalize($fieldValue_t)

          ARRAY TEXT($fieldNames_at;1)
          ARRAY TEXT($fieldValues_at;1)

          $fieldNames_at{1}:=$fieldName_t
          $fieldValues_at{1}:=$fieldValue_t
          ` tell the Front-end to update the value
          DAX_Dev_CB_SetFieldValues (->$fieldNames_at;->$fieldValues_at)
          DAX_Dev_CB_SetStatus (1)  ` Set Callback OK status

        : ($fieldName_t="Country")
          ` Here the user has selected a Country from a choice list on
the front-end
          ` Now populate the states (provinces, etc.) choice list based
on their selection
          ARRAY TEXT($listItems_at;0)

          ` A method that populates a text array with state names
          My_GetStatesFromCountry($fieldValue_t;->$listItems_at)

          ` tell the Front-end to update the 'States List'
          DAX_Dev_CB_SetChoiceList ("States List";->$listItems_at)
          DAX_Dev_CB_SetStatus (1)  ` Set Callback OK status

        : ($fieldName_t="Zip Code")
          ` Here the user has entered a zip code on the front-end
          ` we are now going to populate the city and state based on zip
code

          QUERY([Zips];[Zips]Zip_Code=$fieldValue_t)
          C_TEXT($city_t;$state_t)
          If (Records in selection([Zips])=1)
            $city_t:=[Zips]City
            $state_t:=[Zips]State

            ARRAY TEXT($fieldNames_at;0)
            ARRAY TEXT($fieldValues_at;0)

            $fieldNames_at{1}:="City"
            $fieldValues_at{1}:=$city_t
            $fieldNames_at{2}:="State"
            $fieldValues_at{2}:=$state_t

            ` tell the Front-end to update multiple field values
            DAX_Dev_CB_SetFieldValues (->$fieldNames_at;-
>$fieldValues_at)
            DAX_Dev_CB_SetStatus (1)  ` Set Callback OK status

          Else
```

```
            ` we didn't find that zip so send an error
            DAX_Dev_CB_SetMessage ("Invalid Zip Code")
            DAX_Dev_CB_SetStatus (0)  ` Set Callback OK status to 0

          End if

      End case

    End if

  : ($eventID_l=On Load )

End case
```

DAX_Dev_SetCallBackExeStatus

A method named DAX_Dev_SetCallBackExeStatus was added to version 1.1 to further control callbacks. This method sets a flag to let the browser know if the Call Back method's execution is successful. It must be executed for every callback execution.

In each of your Call Back methods, add a call to `DAX_Dev_SetCallBackExeStatus` with a `LONGINT` parameter of either 1 for success or 0 for failure.

Example

```
DAX_Dev_SetCallBackExeStatus(1)
```

Configuring Callbacks

Programmatically

Callbacks are installed in the method DAX_DevHook_InstallCallBack. To install a callback you call the method DAX_Dev_SetCallBack and pass the event id, table name, field name, and the name of the callback method as input parameters.

Parameters

- $1 - Longint - Event ID
- $2 - Text - Table Name
- $3 - Text - Field Name
- $4 - Text - Method Name
- $5 - Boolean - (Optional) True if non-modifiable, otherwise False

For example:

```
` *** Method: DAX_DevHook_InstallCallBack
DAX_Dev_SetCallBack (On Load;Table name(7);Field
name(7;1);"mAssignSequenceID")
DAX_Dev_SetCallBack (On Data Change;Table name(7);Field
name(7;2);"mConvertToUpperCase")
```

or

```
` *** Method: DAX_InstallCallBack
DAX_Dev_SetCallBack (On Load;Table name(->[Customers]);Field name(-
>[Customers]ID);"mAssignDefaultValue")
```

```
DAX_Dev_SetCallBack (On Data Change;Table name(->[Customers]);Field
name(->[Customers]Name);"mConvertToUpperCase")
```

The table and field names can also be from a DCS or custom view. In the case of a View (which is only created from the frontend) you will have to get the name of the View from the Admin area on the frontend.

HTTP Request

Setting a callback can also be done through HTTP request by calling the command AdminSetCallBack. Please refer to the Commands section of the Callback.

Method that determines execution status

A new method named DAX_Dev_SetCallBackExeStatus has been added to version 1.1. This method set a flag to the http response, letting the frontend know whether the execution is successful. It must be executed for every callback execution.

- $1 - LONGINT - (Input) 1 for successful and 0 for failed

Method to be executed on a Callback

The method that is executed on a callback is the method that the developer specified through the Web-Admin or in the DAX_Dev_SetCallBack method. From the example above, we have 2 methods that are installed for the On Load and On Data Change event. These methods must be created by the developer and they must return a text parameter as shown below:

- $0 - TEXT - (Output) Processed value to be sent back to the frontend

Within your callback method you can retrieve 8 attributes from 4D Ajax Framework for each execution:

- Event ID - Longint - Event ID
- Table Name - Text - Table Name
- Table ID - Longint - Table ID
- Field Name - Text - Field Name
- Field ID - Longint - Field ID
- Record ID - Longint - Record ID (-3 for new record)
- Field Value - Text - The current value of the field (in text)
- Message - Text - Text message to you want to be returned to the frontend

To get the value of these attributes, you need to execute the command DAX_Dev_GetCallBackVar. The following are the parameters that you need to pass into DAX_Dev_GetCallBackVar:

- $1 – TEXT
    - (Input) Name of the attribute
    - (Event ID, Table Name, Table ID, Field Name, Field ID, Record ID, Value, Message)
- $2 – POINTER
    - (Output) Pointer to a variable to receive the attribute value

At the moment, you can set a value only to the attribute named "Message". To set the value to the Message attribute, you need to execute the command DAX_Dev_SetCallBackVar. The following are the parameters that you need to pass into DAX_Dev_SetCallBackVar:

- $1 – TEXT
  - (Input) Name of the attribute
  - (Message)
- $2 – POINTER
  - (Input) Pointer to a variable containing a value you want to set to the attribute

Here are some of the examples:

```
       ` *** Method: mAssignDefaultValue
       ` *** Assign a default value if the field is empty of the record
is a new record
       C_TEXT($0)
       C_TEXT($Value_t;$Message_t)
       C_LONGINT($RecordID_l;$TableID_l;$FieldID_l)
       DAX_Dev_GetCallBackVar("Table ID";->$TableID_l)
       DAX_Dev_GetCallBackVar("Field ID";->$FieldID_l)
       DAX_Dev_GetCallBackVar("Record ID";->$RecordID_l)
       DAX_Dev_GetCallBackVar("Field Value";->$Value_t)
       If($RecordID_l=New record) | ($Value_t="")
          If(Type(Field($TableID_l;$FieldID_l)->)=Is Text ) |
(Type(Field($TableID_l;$FieldID_l)->)=Is Alpha Field ) |
(Type(Field($TableID_l;$FieldID_l)->)=Is String Var )
             $Message_t:="A default value has been assigned to the
field."
             DAX_Dev_SetCallBackVar("Message";->$Message_t)
             $0:="My Default Value"
          End if
       Else
          $0:=$Value_t
       End if
       DAX_Dev_SetCallBackExeStatus(1)

       ` *** Method: mConvertToUpperCase
       C_TEXT($0)
       C_TEXT($Message_t;$Value_t)
       $Message_t:="The text value has been converted to uppercase."
       DAX_Dev_SetCallBackVar("Message";->$Message_t)
       DAX_Dev_GetCallBackVar("Field Value";->$Value_t)
       DAX_Dev_SetCallBackExeStatus(1)
       $0:=Uppercase($Value_t)
```

Front End Commands

AdminSetCallBack

/DAX/AdminSetCallBack

This command allows a callback to be set from the Web-frontend. This call can be used to create a new callback or modify an existing callback. However, if the callback event is set programmatically through the method DAX_Dev_SetCallBack (with non-modifiable set), it cannot be modified by AdminSetCallBack command.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
eventide	NUMERIC	0 or 20
tableid	NUMERIC	1, 2, 3,..., N
fieldid	NUMERIC or [NUMERIC][NUMERIC]	1, 2, 3,..., N or [PhysicalTableID][PhysicalFieldID]
methodname	TEXT	mAssignDefaultValue

For example:

Setting a callback for a field that is belongs to a physical table

```
   eventid     =   1  (On Load)
   tableid     =   7  (Physical Table ID)
   fieldid     =   1  (Physical Field ID)
   method name =   mAssignDefaultValue

http://localhost:8000/DAX/AdminSetCallBack?sessionId=S9222006150444CCWB&eventid
=1&tableid=7&fieldid=1&methodname=mAssignDefaultValue
```

OR

```
   eventid     =   1  (On Load)
   tableid     =   7  (Physical Table ID)
   fieldid     =   [7][1]  (Physical Table and Field ID)
   method name =   mAssignDefaultValue

http://localhost:8000/DAX/AdminSetCallBack?sessionId=S9222006150444CCWB&eventid
=1&tableid=7&fieldid=[7][1]&methodname=mAssignDefaultValue
```

Setting a callback for a field that is belongs to a view (or virtual table)

```
   eventid     =   1  (On Load)
   tableid     =   -3  (View's Table ID)
   fieldid     =   [5][2]  (Physical Table and Field ID)
   method name =   mAssignDefaultValue

http://localhost:8000/DAX/AdminSetCallBack?sessionId=S9222006150444CCWB&eventid
=1&tableid=-3&fieldid=[5][2]&methodname=mAssignDefaultValue
```

Setting a callback for a field that belongs to a DCS view

```
   eventid     =   1  (On Load)
   tableid     =   32001  (DCS's Table ID)
   fieldid     =   [32001][2]  (DCS's Table ID and Array Position in the DCS)
   method name =   mAssignDefaultValue

http://localhost:8000/DAX/AdminSetCallBack?sessionId=S9222006150444CCWB&eventid
=1&tableid=32001&fieldid=[32001][2]&methodname=mAssignDefaultValue
```

Reply

```
<AdminSetCallBack>
   <Status Success="True"/>
</AdminSetCallBack>
```

GetCallBack

/DAX/GetCallBack

This command allows the Web-frontend to get Callback information for the one or all fields from a table. If the field id is specified, the backend will find on the callback for the given field. If at least one callback event is found, the return XML will look like the example reply (shown below). If the field id is 0, all fields in the table (with at least one callback event) are returned in the XML response.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	NUMERIC	1, 2, 3,..., N
fieldid	[NUMERIC][NUMERIC]	1, 2, 3,..., N (0 for all fields)

```
tableid     =   7  (Physical, View or DCS Table ID)
fieldid     =   [7][1]  (Physical Table and Field ID)

http://localhost:8000/DAX/GetCallBack?sessionId=S9222006150444CCWB&tableid=7&fi
eldid=[7][1]
```

Reply

```
<GetCallBack>
   <CallBack eventid="1" fieldid="1" method="mConvertCase" tableid="7"
notmodifiable="False"/>
   <CallBack eventid="20" fieldid="1" method="mAssignDefaultValue"
tableid="7" notmodifiable="True"/>
</GetCallBack>
```

Please note that the "notmodifiable" attribute is set to True only if the Developer installed the callback programmatically inside the method DAX_DevHook_InstallCallback.

ExecuteCallBack

/DAX/ExecuteCallBack

This command executes the method that is installed for a specific event.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
eventid	NUMERIC	1, 2, 3,..., N
tableid	NUMERIC	1, 2, 3,..., N (Physical, View or DCS Table ID)
fieldid	[NUMERIC][NUMERIC]	[T][F][R] ([T][F][-3] for new record)

	[NUMERIC]	
value	TEXT	MyValue

For example:

1. Get a default value

```
   eventid = 1 (On Load)
   tableid = 7 (Physical Table ID)
   fieldid = [7][1][-3] (Physical Table , Field and New Record ID)

http://localhost:8000/DAX/ExecuteCallBack?sessionId=S9212006224024CCWB&eventid=
1&tableid=7&fieldid=[7][1][-3]
```

```
Reply
<Callback eventid="1">
   <Result fieldid="[7][1][-3]" value="Default Value" message="A default
value has been assigned to the field." />
</Callback>
```

2. Convert text to uppercase

```
   eventid = 20 (On Data Change)
   tableid = 7 (Physical Table ID)
   fieldid = [7][1][2] (Physical Table , Field and Record ID)

http://localhost:8000/DAX/ExecuteCallBack?sessionId=S9212006224024CCWB&eventid=
20&tableid=7&fieldid=[7][1][2]&value=usa
```

Reply

```
<Callback eventid="20">
   <Result fieldid="[7][1][2]" value="USA" message="The text value has
been converted to uppercase." />
</Callback>
```

Javascript Callbacks

Javascript callbacks allow developers to append their own javascript code to the DAX framework.

Blank callback functions are located in dev/callbacks.js file.

On Editor Pull

Executed when editor get the record from the 4D.

Parameters passed:

- selectionName - name of the current selection. Use this to filter which selections are processed.
- xmlRecord - record in raw XML format. XML structure is located at Query_Calls under Get Record command.

Following sample code will take raw XML record sent from 4D and change every social security number into 'XXX-XX-XXXX' string.

```
function callback_onEditorPull (selectionName, xmlRecord)
```

```
{

        // custom code starts here

        // get reference to field elements
        var result = 
xmlRecord.responseXML.getElementsByTagName("queryResult").item(0);
        var row = result.getElementsByTagName('row').item(0);
        var fields = row.getElementsByTagName('field');

        // loop through fields
        for (l = 0; l < fields.length; l++) {
                // check if field contains social security number, and if 
it does, mask it
                if (getField(selectionName, 
fields[l].getAttribute('id')).fieldname == 'Person_ID')
                        fields[l].textContent = 'XXX-XX-XXXX';
        }

        // custom code ends here

        // return the xml reply back to the DAX
        return xmlRecord;

}
```

On Query

Executed when view request data from the 4D.

Parameters passed:

- selectionName - name of the current selection. Use this to filter which selections are processed.
- xmlRecord - record in raw XML format. XML structure is located at Query_Calls under Get Record command.

Following sample code will convert all names from the 'First_Name' field and replace them with an initial.

```
function callback_onViewQuery (selectionName, xmlRecord, viewType)
{

        // custom code starts here

        // get reference to field elements
        var result = 
xmlRecord.responseXML.getElementsByTagName("queryResult").item(0);
        var rows = result.getElementsByTagName('row');

        // loop through rows
        for (k = 0; k < rows.length; k++) {
                // loop through fields
                var fields = rows[k].getElementsByTagName('field');
                for (l = 0; l < fields.length; l++) {
                        // check if this is 'First_Name' field and if it is 
replace name with initial
                        if (getField(selectionName, 
fields[l].getAttribute('id')).fieldname == 'First_Name')
                                fields[l].textContent = fields[l].substr(0,1);
```

```
              }
       }

       // custom code ends here

       // return the xml reply back to the DAX
       return xmlRecord;

}
```

After Successful Record Save

Executed when record is successfully saved in editor.

Parameters passed:

- selectionName - name of the current selection. Use this to filter which selections are processed.
- recordId - record id returned from 4D in [x][y] format.

Following sample code will display an alert with selection name and record id.

```
function callback_afterRecordSaveSuccess (selectionName, recordId)
{

       // custom code starts here

       alert ('Record ' + recordId + ' from ' + selectionName + ' saved
successfully.');

       // custom code ends here

}
```

After Successful Login

Executed when user successfully logs in.

Parameters passed:

- username - name of the logged in user.

Following sample code will display an welcome message.

```
function callback_onLoginSuccess (username)
{
       // custom code starts here

       alert ('Welcome, ' + username + '.');

       // custom code ends here
}
```

Before Record Save

Executed before saved record is sent to 4D, allowing developer to modify the data.

Parameters passed:

- selectionName - name of the current selection. Use this to filter which selections are processed.

- recordId - saved record id.
- recordData - data object with following arrays attached:
  - .fieldId - array that holds id of the field. Used internally by DAX.
  - .fieldName - name of the corresponding field.
  - .value - value of the corresponding field.

Sample code:

```
function callback_beforeRecordSave(selectionName, recordId, recordData)
{

       // custom code starts here


       // custom code ends here

       //return data array back to the DAX
       return recordData;
}
```

Error Trap

Traps the error returned from the 4D, and returns error hint and full error message.

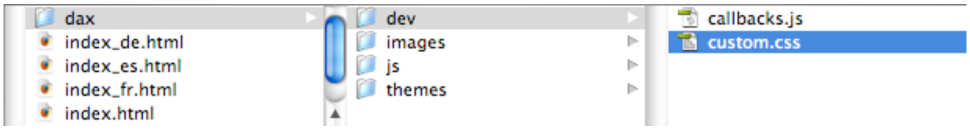
```
function fourdaf_dev_errorTrap (hint, message){

}
```

CSS Override

4D developers are now more easily able to modify the existing CSS of the framework. The file index.html now has a call to a new CSS stylesheet called *custom.css*. Developers can make any CSS modifications to this file and those changes will override any existing CSS styling for the framework. This is a convenient and worry-free way to change CSS styling without having to deal with the existing CSS templates. Also, feel free to include and refer to *custom.css* in your pre-existing files from a previous version of the framework.

Thanks to the natural way that CSS operates, since *custom.css* is the last stylesheet to the Javascript code in index.html it naturally overrides pre-existing CSS.

Custom.css can be found in /dax/dev/.



DAX Error Stack

The 4D Ajax Framework v11 introduced an error stack which allows developers to determine if an error has occurred when they called one of the Dax_Dev_@ methods. If an error occurs it is added to the end of the stack. The error stack is at the process level.

With the addition of the error stack new Developer methods have been introduced:

```
DAX_Dev_ERR_ClearErrorStack
DAX_Dev_ERR_ErrorsExist
DAX_Dev_ERR_GetErrorCount
DAX_Dev_ERR_GetErrorCode
DAX_Dev_ERR_GetErrorLocation
DAX_Dev_ERR_GetErrorText
DAX_Dev_ERR_GetErrorTextByCode
```

DAX_Dev_ERR_ClearErrorStack

This method allows Developers to reset or define the error arrays. It is called at the beginning of every DAX web process, as well as in the initialize, shutdown, and Daemon processes. You can call this method at any time to clear all the errors currently in the stack. If you were to call one of the DAX_Dev_@ methods within a new process (not recommended) you must first call this method.

DAX_Dev_ERR_ErrorsExist

Returns True if errors have occurred in this process. You can call this method after any call to a DAX_Dev_@ method to check if an error has occurred.

```
Passed:
None

Returned:
$0         BOOLEAN  True if any errors have occurred
```

DAX_Dev_ERR_GetErrorCount

Returns the total number of errors in the stack.

```
Passed:
None

Returned:
$0         LONGINT  Total number of errors in the stack
```

DAX_Dev_ERR_GetErrorCode

Call this method to retrieve the DAX error code for a given error in the stack.

```
Passed:
$1       LONGINT  Index in the error stack

Returned:
$0         LONGINT  Error code for given index
```

DAX_Dev_ERR_GetErrorLocation

Call this method to retrieve the name of the method where a given error occurred.

```
Passed:
$1       LONGINT  Index in the error stack

Returned:
$0        TEXT     Name of the method that reported an error for
                   the given index
```

DAX_Dev_ERR_GetErrorText

Call this method to get the actual error text for a given error in the stack.

```
Parameters
  Passed:
      $1 LONGINT - Index in the error stack
  Returned:
      $0 TEXT -
   Passed:
   $1       LONGINT  Index in the error stack

   Returned:
   $0        TEXT     Error text for the given index
```

DAX_Dev_ERR_GetErrorTextByCode

Call this method to get the actual error text for any given DAX error number. Note that the difference between *DAX_Dev_ERR_GetErrorText* and this method is that this method is independent from the error stack.

```
Passed:
$1       LONGINT  Valid DAX error code

Returned:
$0        TEXT     Error text for the given error code
```

DAX_Dev_ERR_OnSavingRecord

This method is added to v11.2 to give the developer an ability to return a specific error message for the record that is not being saved. This method must be executed only within the method ***DAX_DevHook_SaveRecord***.

```
Syntax:   DAX_Dev_ERR_OnSavingRecord (ErrorMessage)
```

```
Passed:
$1       TEXT  Error Message you want to report to the frontend
```

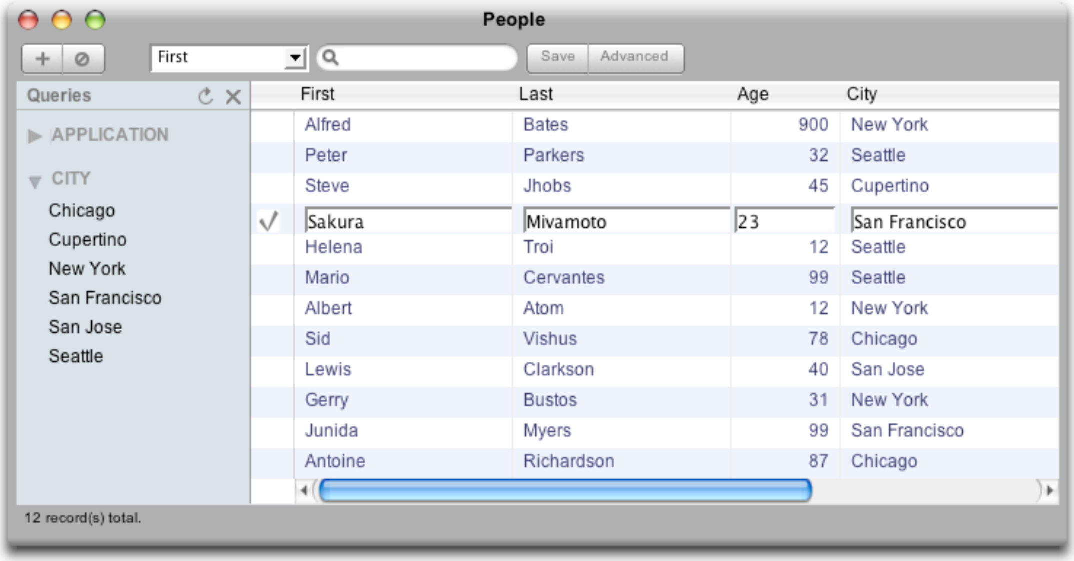
Front End Objects

The Data Grid

Introduction

The new Data Grid introduced in 4D Ajax Framework v11 Release 1 (11.1) was a redesigned version of the existing Grid object built from the ground up. With a re-worked core the Data Grid is now a more universal object with a more granular API for many of the 4D Ajax Framework objects. It has a wealth of features such as the ability to:

- Show and hide columns.
- Select single records, multiple records at the same time, or none at all.
- Lock columns, headers, and footers.
- Fire events when rows or cells are selected, and when data arrives from the back end.
- Apply CSS styling.
- And more…



	First	Last	Age	City
	Alfred	Bates	900	New York
	Peter	Parkers	32	Seattle
	Steve	Jhobs	45	Cupertino
✓	Sakura	Mivamoto	23	San Francisco
	Helena	Troi	12	Seattle
	Mario	Cervantes	99	Seattle
	Albert	Atom	12	New York
	Sid	Vishus	78	Chicago
	Lewis	Clarkson	40	San Jose
	Gerry	Bustos	31	New York
	Junida	Myers	99	San Francisco
	Antoine	Richardson	87	Chicago

12 record(s) total.

The Data Grid's Application Programming Interface (API) commands are outlined in this document for when developers want to build the Data Grid onto their Custom HTML pages.

Create & Display Data Grid

Here are the commands associated with creating a Data Grid on your custom page.

Create a new Data Grid

First, create the Data Grid using the following:

```
var myGrid = new dax_dataGrid(selection, location, headerRows,
lockedLeftColumns, useControlColumn);
```

- **selection** – Name of Table, View, or DCS the Data Grid will display.
- **location** – Name of the <div> on the HTML page where the Data Grid will be embedded. Pass *null* or no value to float the Data Grid as a window.
- **headerRows – (optional)** Number of rows the header is by height. Default size is 1 row. The first header row automatically has the capability of being populated by titles. Additional header rows are blank and are up to the Developer to populate.
- **lockedLeftColumns – (optional)** Number of columns to lock on the left side.
- **useControlColumn – (optional)** Show control column on the left. Default value is ‘True’. Used for inline editing.

Initialize the Data Grid

Then after creating the grid, use the *.go()* call to run and display it.

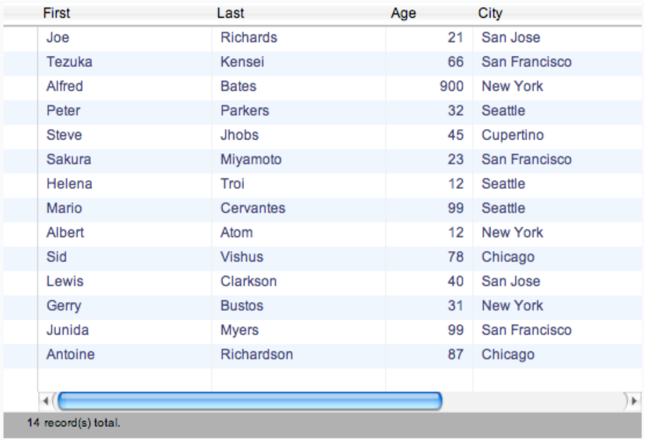
```
myGrid.go();
```

At this point, a Data Grid will appear on your HTML page. The *new dax_dataGrid* and the *.go()* commands are the minimum commands necessary for displaying a Data Grid.

Examples:

3) A Data Grid embedded to the page:

```
var myGrid = new dax_dataGrid('People', $('DataGridDiv'));
myGrid.go();
```



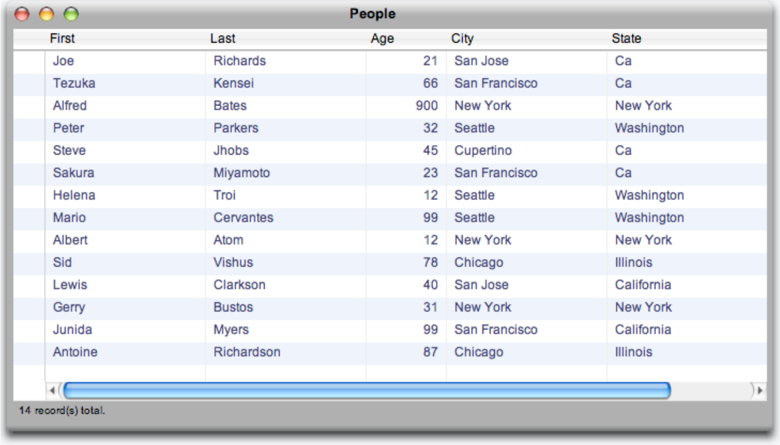
First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

14 record(s) total.

In this example we have a <div> with an ID of "DataGridDiv". Notice the notation used in this example when embedding the Data Grid into a <div>.

4) The same Data Grid as a floating window:

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();
```



People

First	Last	Age	City	State
Joe	Richards	21	San Jose	Ca
Tezuka	Kensei	66	San Francisco	Ca
Alfred	Bates	900	New York	New York
Peter	Parkers	32	Seattle	Washington
Steve	Jhobs	45	Cupertino	Ca
Sakura	Miyamoto	23	San Francisco	Ca
Helena	Troi	12	Seattle	Washington
Mario	Cervantes	99	Seattle	Washington
Albert	Atom	12	New York	New York
Sid	Vishus	78	Chicago	Illinois
Lewis	Clarkson	40	San Jose	California
Gerry	Bustos	31	New York	New York
Junida	Myers	99	San Francisco	California
Antoine	Richardson	87	Chicago	Illinois

14 record(s) total.

Note: var myGrid = new dax_dataGrid('People'); also displays the Data Grid as a floating window.

Add footers

The following command can be used to set the number of footer rows.

```
myGrid.setFooterRows(number of footer rows);
```

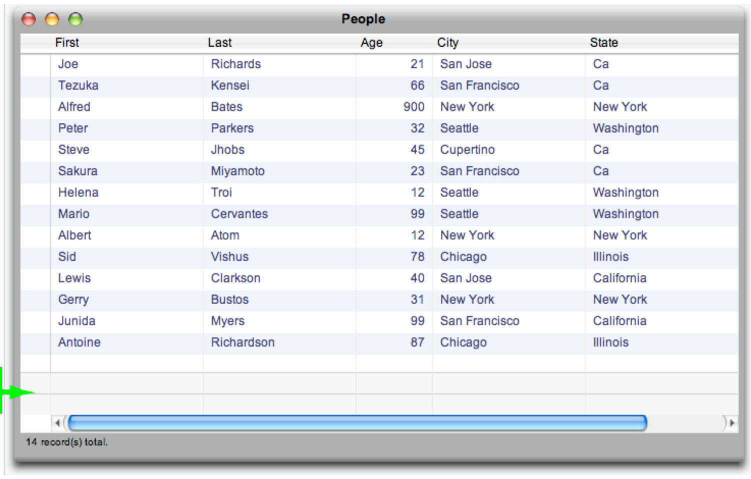
It is recommended to call this command before the *.go()* command.

This command is not to be confused with the 3rd parameter in *new dax_dataGrid* command, which defines the number of header rows.

Example:

1) This example will build a Data Grid as a floating window with 2 footer rows.

```
var myGrid = new dax_dataGrid('People', null);
// Add two footer rows
myGrid.setFooterRows(2);
myGrid.go();
```



People

First	Last	Age	City	State
Joe	Richards	21	San Jose	Ca
Tezuka	Kensei	66	San Francisco	Ca
Alfred	Bates	900	New York	New York
Peter	Parkers	32	Seattle	Washington
Steve	Jhobs	45	Cupertino	Ca
Sakura	Miyamoto	23	San Francisco	Ca
Helena	Troi	12	Seattle	Washington
Mario	Cervantes	99	Seattle	Washington
Albert	Atom	12	New York	New York
Sid	Vishus	78	Chicago	Illinois
Lewis	Clarkson	40	San Jose	California
Gerry	Bustos	31	New York	New York
Junida	Myers	99	San Francisco	California
Antoine	Richardson	87	Chicago	Illinois

14 record(s) total.

2 Footer rows

Lock right columns

The following command can be used to lock the number of right columns.

```
myGrid.setRightLockedColumns(number of locked right columns);
```

It is recommended to call this command before the *.go()* command.

This command is not to be confused with the 4th parameter in *new dax_dataGrid* command, which defines the number of locked left columns.

The number of right-most columns you specify here will always display and remain locked in the Data Grid.

Example:

1) This example will lock 1 column on the far right.

```
var myGrid = new dataGrid ('People', null);
// Lock 1 right column
myGrid.setLockedRightColumn(1);
myGrid.go();
```

Resizing

<div> areas on an HTML page may resize. If that resized <div> contains your Data Grid, you will need to resize the Data Grid so that it fits appropriately. Call the following command so that the Data Grid is correctly resized.

```
myGrid.activate();
```

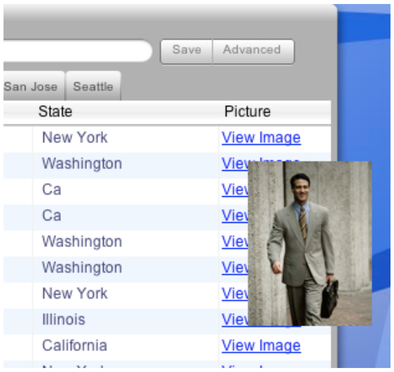
The Data Grid will resize to fit in the <div> that contains it.

Example:

1) In this example, first the <div> that holds the grid is resized to 600 pixels wide. Then, the *.activate()* command is called to resize the Data Grid into the <div>.

```
// Change the size of the div that holds the Data Grid
$('myGridDiv').style.width = '600px';
// Update Data Grid to fit
myGrid.activate();
```

Picture Preview



State	Picture
New York	View Image
Washington	View Image
Ca	Viev
Ca	Viev
Washington	Viev
Washington	Viev
New York	Viev
Illinois	Viev
California	View Image

Every picture field in the Data Grid displays a 'View Image' link to load it. When users mouse over this link a preview of the image appears. Use the following command to turn Picture Preview on or off.

```
myGrid.enablePicturePreview(previewEnabled)
```

previewEnabled – Boolean. Set to *true* to enable Picture Preview. Set to *false* otherwise. Picture Preview is set to *true* by default.

This command must be called before the *.go()* call.

To take this command to another level, think about using it conjunction with some events such as *.onBeforeHover* and *.onAfterHover*.

Example:

1) This example disables Picture Preview.

```
myGrid = new dax_dataGrid('People');
// Disable Picture Preview
myGrid.enablePicturePreview(false)
myGrid.go();
```

Data

Auto refresh

Enable or disable the Auto Refresh feature using the commands below.

```
myGrid.enableAutoRefresh();
myGrid.disableAutoRefresh();
```

Auto Refresh is on by default. With Auto Refresh on, its default interval is 3 minutes (180 seconds). Modify the refresh rate using the command below.

```
myGrid.setRefreshInterval(seconds);
```

Example:

1) In this example the Data Grid is set to refresh every 10 minutes (600 seconds).

```
// refresh grid every 10 minutes
myGrid.setRefreshInterval(600);
```

Set and get cell contents

To get the contents of a cell use this command:

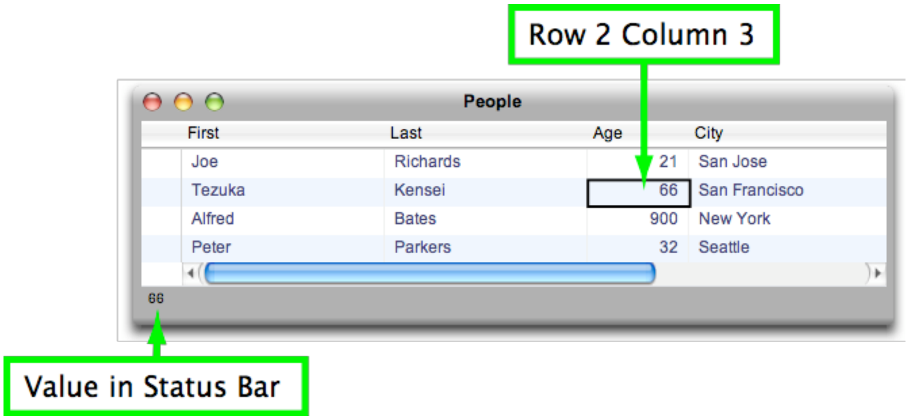
```
var cellValue = myGrid.getCellValue (row, col);
```

- **row –** Specify the row of the cell in which you intend to get the value from.
- **col –** Specify the column of the cell in which you intend to get the value from.

Example:

1) In this example we will get the value of a cell and display it in the Status Bar. We get the cell value during the *onDataLoad* event because the Data Grid cells do not have values until this event occurs.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.showStatusBar(true);
myGrid.go();

myGrid.onDataLoad = function() {
       var cellValue = myGrid.getCellValue (2, 3);
       myGrid.showStatusMessage(cellValue);
}
```



Row 2 Column 3

People

First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle

66

Value in Status Bar

To overwrite the contents of a cell use this command:

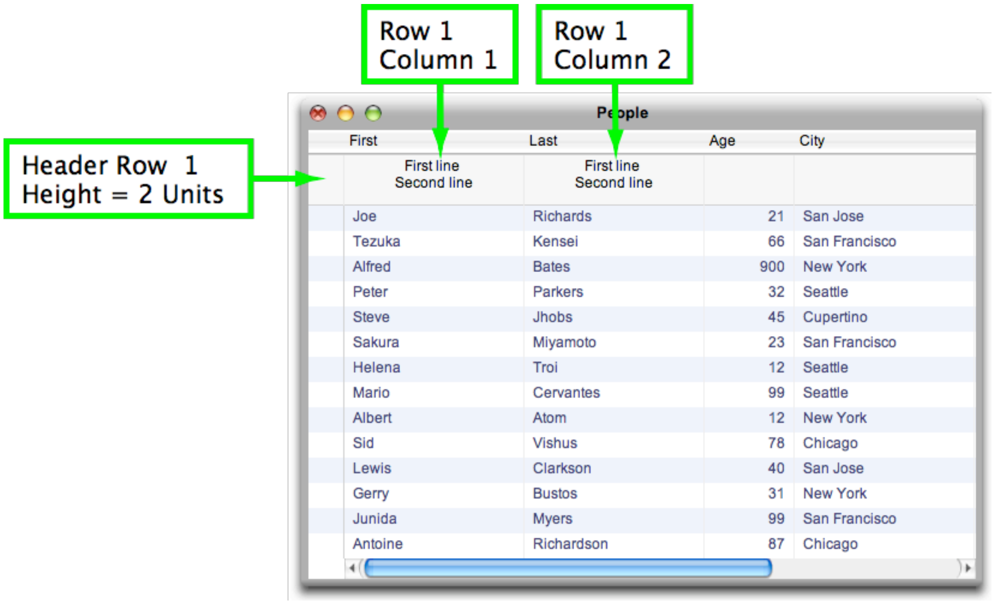
```
myGrid.setCellValue (row, col, content);
```

- **row –** Specify the row of the cell in which you intend to set the *content* into.
- **col –** Specify the column of the cell in which you intend to set the *content* into.
- **content** – the content defined here will overwrite the cell.

Example:

1) Here we will define a variable (*myContents*) as two lines of data. We will change the value of two header cells with the value of *myContents.*

```
// Create Data Grid with 2 Header Rows
var myGrid = new dax_dataGrid('People', null,2);
myGrid.go();

// Set Header Row 1 to height of 2 units
myGrid.setHeaderHeight (1, 2)

// Define myContents which will overwrite specific cells. myContents is 2
// units high.
var myContents = "First line <br/> Second line"

// Set content of Header Row 1 cells
myGrid.setCellValue(1, 1, myContents);
myGrid.setCellValue(1, 2, myContents);
```



First	Last	Age	City
First line Second line	First line Second line		
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

Sleep

This command will cause the Data Grid to no longer automatically communicate with 4D. Thus, periodic updates such as automatic data refreshes and the refreshing of Data Driven Preset Queries will no longer occur.

The user, however, still can manually interact with the Data Grid with full functionality (except for automatic communication).

```
myGrid.sleep();
```

This command is useful when the Data Grid is hidden, and when resources are not intended to be used to update it. Use *.wake()* to resume automatic updates.

Example:

1) This example puts the Data Grid to sleep and then hides it.

```
myGrid.sleep();

// hide grid
$('myGridDiv').style.visibility = 'hidden';
```

Wake

This command will wake a Data Grid that was previously put to sleep. It will resume all periodic updates (data refresh, unique query refresh) that were paused with the *.sleep()* command.

```
myGrid.wake();
```

Example:

1) This example sets a Data Grid that previously set to 'hidden' to be 'visible.' Then all periodic communication with 4D is resumed using the *.wake()* command.

```
// show grid
$('myGridDiv').style.visibility = 'visible';

myGrid.wake();
```

Destroy

This command removes the Data Grid object completely.

```
myGrid.destroy();
```

Example:

```
myGrid.destroy();

// After the Grid is destroyed clear the variable
myGrid = null;
```

Search and advanced search toolbar

The Search Toolbar displays only the fields you specify as searchable in the 4D Ajax Framework Client environment.

> To make a field searchable, in the Client go to Control Panel -> Access Control tab. Select the table the field belongs to. For each field you want searchable, check the 'S' checkbox.

To override these searchable fields with a new list, use the following command.

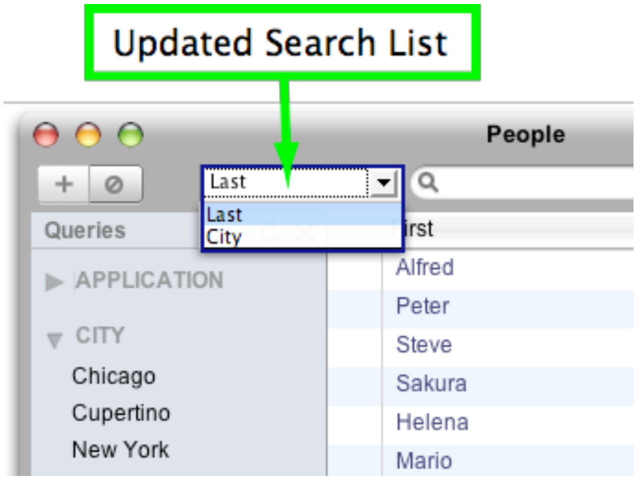
```
myGrid.updateSearchFieldList(recordID);
```

recordId – String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

Example:

1) This example enables the fields *Last* and *City* as searchable based on their Record IDs:

```
myGrid.updateSearchFieldList([ '[1][2]', '[1][4]']);
```



Example:

2) Remember, you can also use *dax_getField* and the attribute fieldid to retrieve field IDs, as seen in this example:

```
myGrid.updateSearchFieldList([
myGrid.dax_getField("Employee","FirstName").fieldid,
myGrid.dax_getField("Employee","LastName").fieldid]);
```

Reference

Use the commands in this section to reference specific areas on the Data Grid such as rows, columns, or cells.

Get Column number

Use this command to get a column number using a Field ID. It is best to use 4D Ajax Framework Bridge command *dax_getField* beforehand to get the Field ID in the first place.

```
var columnNumber = myGrid.getColumnByFieldId(fieldId);
```

Example:

1) Description

```
// find out which column belongs to field [Employee]FirstName
var fieldId = dax_getField('Employee', 'FirstName').fieldid;
var employeeFirstNameColumn = myGrid.getColumnByFieldId(fieldId);
```

Get Row number

Use this command to get a row number:

```
var rowNumber = myGrid.getRowByRecordId(recordId);
```

recordId – String, unique value defined for each record denoted by two values *x* and *y* in the format ‘[x][y]’.

Example:

1) This example will get a row number based on record ID.

```
// check which row, if any, has the record with id '[3][4]'
var recordRow = myGrid.getRowByRecordId('[3][4]');
```

Get footer row number

To get footer row number:

```
var realRowNumber = myGrid.getFooterRowNumber(footerRowNumber);
```

footerRowNumber – Integer, representing the number of the footer row. Numbering begins at 0 (zero) so the first footer row is 0 (zero).

Example:

1) To get the first footer row:

```
var myRow = myGrid.getRow(myGrid.getFooterRowNumber(0));
```

Also see the example in section *Change Header and Footer Row Height.*

Get cell pointer

Use this command to get a cell reference based on its row and column number.

```
var myCell = myGrid.getCell(row, column)
```

Cell Properties

Here are the properties of a given cell. Use the *getCell* command to get a reference to a cell and then use the properties listed below to get more information on it.

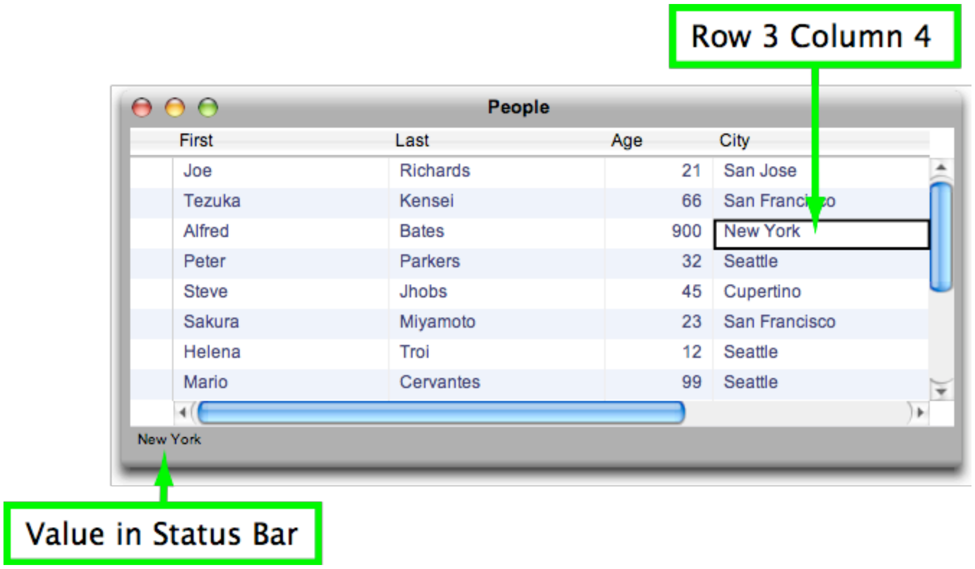
Accessible via myCell.property:

- **allowDragDrop –** Boolean, which returns *true* or *false* answering whether or not this cell supports the Drag and Drop events.
- **allowDragOut –** Boolean, which eturns *true* or *false* answering whether or not this cell is draggable.
- **column –** Integer, which represents the column the cell resides in. Note: Column numbering starts from 0 (zero).
- **row –** Integer, which represents the row the cell resides in. Note: Row numbering starts from 0 (zero).
- **value –** String, which represents the value in the cell.
- **grid -** Pointer to the Data Grid containing the cell.

Example:

1) This example will get a reference to a cell. From there it will display the cell’s value in the Status Bar. We get the cell value during the *onDataLoad* event because the Data Grid cells do not have values until this event occurs.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.showStatusBar(true);
myGrid.go();

myGrid.onDataLoad = function() {
       // Get reference to cell in Row 3 and Column 4
       var myCell = myGrid.getCell(3,4);
       // Display cell’s value in Status Bar
       myGrid.showStatusMessage(myCell.value);
}
```



Row 3 Column 4

First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle

New York

Value in Status Bar

Get row pointer

Use this command to get a reference to a row based on its row number. Row numbering starts at 0 (zero).

```
var myRow = myGrid.getRow(row)
```

Row Properties

Here are the properties of a given row. Use the *.getRow* command to get a reference to a row and then use the properties listed below to get more information on it.

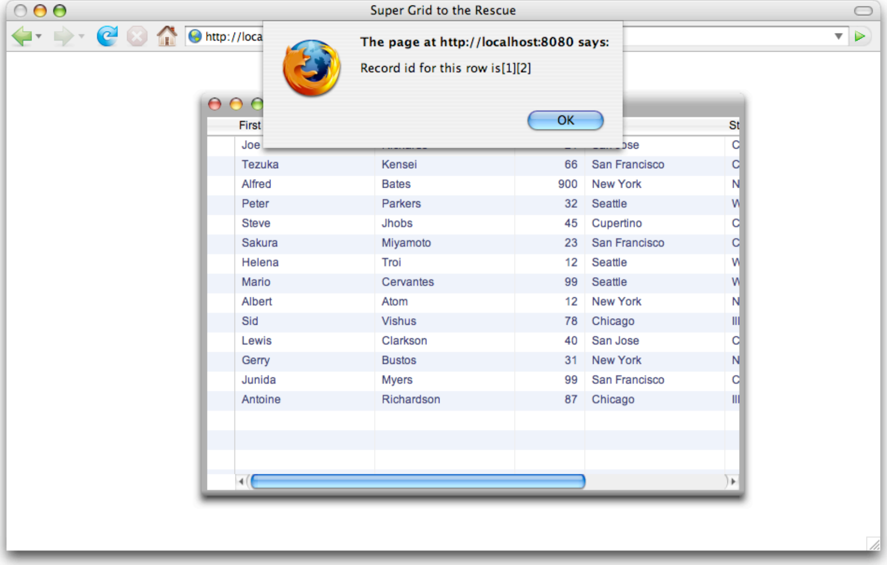
Accessible via myRow.property:

- **cells -** Array of cells. To access cell properties see the *Cell Properties* section.
- **recordid –** String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.
- **rowHeight -** integer, representing row height in units.
- **type** – Possible values are: 'header', 'footer', or 'default', which indicate row type.

Example:

1) This example will get the reference for the 4th row and then display its Record ID in the browser's Alert dialog. We get the row's Record ID during the *onDataLoad* event because Record ID's are undefined before this event occurs.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();

myGrid.onDataLoad = function() {
// get reference for 4th row
var myRow = myGrid.getRow(3);
// show record id via .recordId property
alert ('Record id for this row is' + myRow.recordId);
}
```



The page at http://localhost:8080 says:

Record id for this row is[1][2]

First				St
Joe	[illegible]	[illegible]	[illegible]ose	C
Tezuka	Kensei	66	San Francisco	C
Alfred	Bates	900	New York	N
Peter	Parkers	32	Seattle	W
Steve	Jhobs	45	Cupertino	C
Sakura	Miyamoto	23	San Francisco	C
Helena	Troi	12	Seattle	W
Mario	Cervantes	99	Seattle	W
Albert	Atom	12	New York	N
Sid	Vishus	78	Chicago	Il
Lewis	Clarkson	40	San Jose	C
Gerry	Bustos	31	New York	N
Junida	Myers	99	San Francisco	C
Antoine	Richardson	87	Chicago	Il

Get column pointer

Use this command to get a reference to a column based on its column number. Column numbering starts at 0 (zero).

```
var myColumn = myGrid.getColumn(column)
```

Column properties

Here are the properties of a given column. Use the *.getColumn* command to get a reference to a column and then use the properties listed below to get more information on it.

Accessible via myColumn.property:

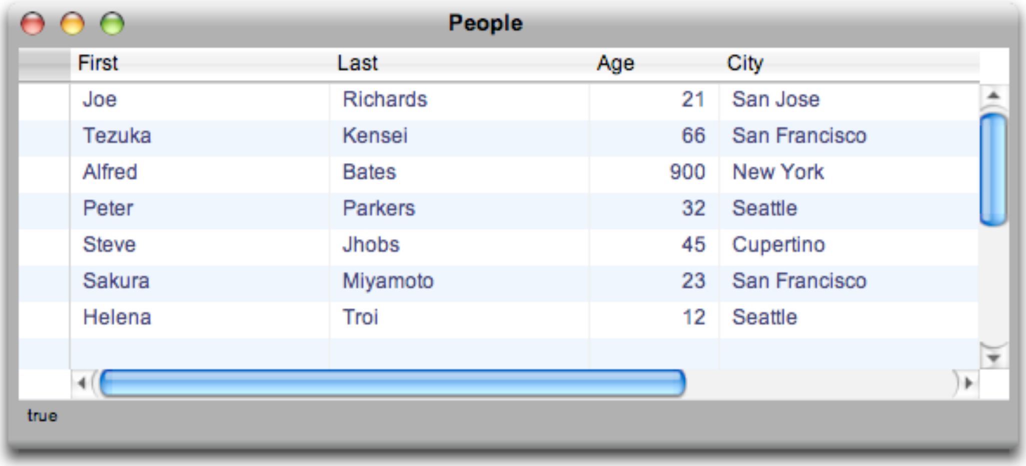
- **cells** – Array that has pointers to cells
- **colWidth –** Integer, representing the width of the column in pixels.
- **field** - Field object, with subproperties .fieldname, .fieldtype, .fieldalias, .fieldid, and so on.
- **isResizable –** Boolean**,** representing if column is resizable.
- **visible –** Boolean, representing if column is visible.

Example:

1) This example will see if the 3rd column is visible and display its status in the Status Bar.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.showStatusBar(true);
myGrid.go();

// check if 3rd column is visible
var myColumn = myGrid.getColumn(3);
var myColumnIsVisible = myColumn.visible;
```

```
// Display column's visibility value in Status Bar
myGrid.showStatusMessage(myColumnIsVisible);
```



People

First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle

true

Presentation

Change the complexion of Data Grid with the following commands.

Set row height in pixels

This will change the base row height in pixels for all rows – this includes all rows as well as all headers and footers. Row height in units is preserved, so row with height of 2 units will have final height of (2 * new size in pixels).

```
myGrid.setRowHeightInPx(rowHeight);
```

- **rowHeight** - new row height in pixels

Example:

1) This example will make the default row height to be 25 pixels. The default row height will change for all rows, headers, and footers.

```
myGrid.setRowHeightInPx(25);
```

Change header and footer row height

Use these commands to change the row height of the header or footer.

These calls must be made immediately after *.go()*:

To set the Header height:

```
myGrid.setHeaderHeight(row number, height in units);
```

- **row number** – A Data Grid defaults to have 1 header row to contain the field title names. This first header row that contains field titles is denoted as header 0. More headers can be added when the Data Grid is created.
- **height in units** – By default headers 1 unit high.

To set the Footer height:

```
daDataGrid.setFooterHeight(row number, height in units);
```

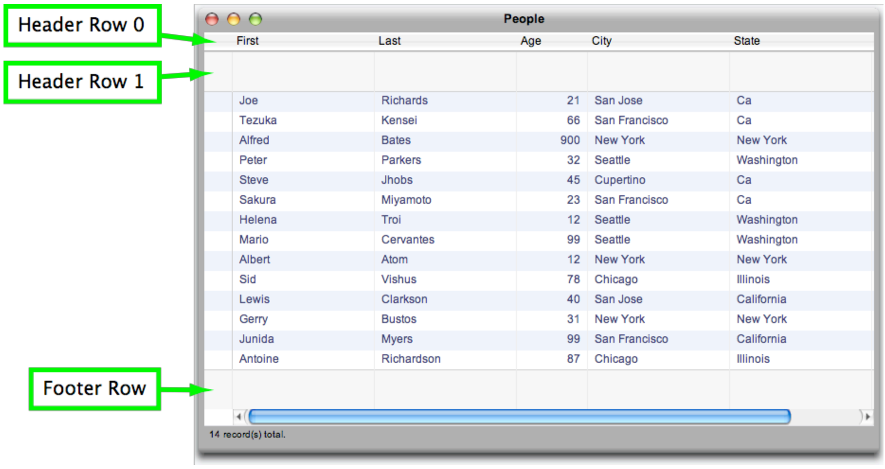
- **row number** – You specify the amount of footer rows in *.setFooterRows().* Use the *.getFooterRowNumber()* command in *.setRowHeight()* to reference the footer row you intend to resize. See example below.
- **height in units** – By default footers are 1 unit high.

1 unit = 21 pixels by default. The amount of pixels in 1 unit can be changed by calling *.setRowHeightInPx().*

Example:

1) Here we will create a grid with 2 header rows and 1 footer row. We will set some of these rows to the height of 2 units:

```
// Create Data Grid with 2 header rows
var myGrid = new dax_dataGrid('People', null, 2);

// Add one footer row
myGrid.setFooterRows(1);

// Initialize the Data Grid
myGrid.go();

// Set Header Row 1 to a height of 2 Units
myGrid.setHeaderHeight (1,2);

// Set Footer Row to a height of 2 Units
myGrid.setFooterHeight (myGrid.getFooterRowNumber(0),2);
```



First	Last	Age	City	State
Joe	Richards	21	San Jose	Ca
Tezuka	Kensei	66	San Francisco	Ca
Alfred	Bates	900	New York	New York
Peter	Parkers	32	Seattle	Washington
Steve	Jhobs	45	Cupertino	Ca
Sakura	Miyamoto	23	San Francisco	Ca
Helena	Troi	12	Seattle	Washington
Mario	Cervantes	99	Seattle	Washington
Albert	Atom	12	New York	New York
Sid	Vishus	78	Chicago	Illinois
Lewis	Clarkson	40	San Jose	California
Gerry	Bustos	31	New York	New York
Junida	Myers	99	San Francisco	California
Antoine	Richardson	87	Chicago	Illinois

14 record(s) total.

Show & hide columns

These commands should be made after the *.go();* call, since columns are not initialized before then.

Hide column based on column number:

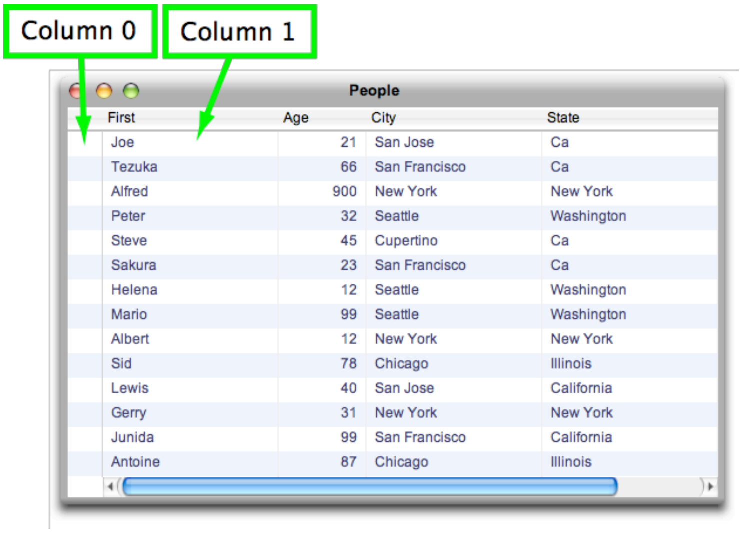
```
myGrid.hideColumn(colNum);
```

- **colNum** – The number of the column starting from the left side of the Data Grid. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).

Example:

1) In this example we will hide column 2. Column 2 is the *Last* name field and it usually appears to the right of the *First* name field.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();
myGrid.hideColumn(2);
```



	First	Age	City	State
	Joe	21	San Jose	Ca
	Tezuka	66	San Francisco	Ca
	Alfred	900	New York	New York
	Peter	32	Seattle	Washington
	Steve	45	Cupertino	Ca
	Sakura	23	San Francisco	Ca
	Helena	12	Seattle	Washington
	Mario	99	Seattle	Washington
	Albert	12	New York	New York
	Sid	78	Chicago	Illinois
	Lewis	40	San Jose	California
	Gerry	31	New York	New York
	Junida	99	San Francisco	California
	Antoine	87	Chicago	Illinois

Note that the Control Column is column 0 because it was set to *true* by default when the Data Grid was created.

Show column based on column number:

```
myGrid.showColumn(colNum);
```

- **colNum** – The number of the column starting from the left side of the Data Grid. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).

Example:

1) After executing the code in the example shown above for *.hideColumn(),* call this code at a later time to make the same column reappear.

```
myGrid.showColumn(2);
```

Get and Set Column width

These commands should be made after the *.go();* call, since columns are not initialized before then.

To set column width in pixels:

```
myGrid.setColumnWidth(colNum, width);
```

- **colNum** – The number of the column starting from the left side of the Data Grid. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).
- **width** – Width to set the column in pixels.

Example:

1) This example will set the 3rd column from the left to a width of 150 pixels:

```
myGrid = new dax_dataGrid('People',null);
myGrid.go();
myGrid.setColumnWidth(2, 150);
```

To get column width in pixels:

```
var myColumnWidth = myGrid.getColumnWidth(colNum);
```

- **colNum** – The number of the column starting from the left side of the Data Grid. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).

Example:

1) In this example will return the width of the 3rd column from the left:

```
var myColumnWidth = myGrid.getColumnWidth(2);
```

Allow user resizing

This command allows or prohibits user resizing of columns with click & drag interface:

```
myGrid.allowColumnResize(allowColumnResize);
```

- **allowColumnResize** – Boolean. Set to *True* to allow column resizing. Set to *False* otherwise.

User resizing is allowed by default.

Example:

1) This example will allow user resizing of the columns:

```
myGrid.allowColumnResize(true);
```

Query

Query-related commands reside here.

New Query, Add Query, and Run Query

Create a new query using the following command:

```
.newQuery();
```

Then add and define the query conditions using the following command:

```
.addQuery(field name, operator, value, and/or flag);
```

- **field name –** String, name of the field to query.
- **operator –** String, representing the operator. Examples are:
    1) ‘=’ (equal)
    2) ‘#’ (not equal)
    3) ‘<’ (less than)
    4) ‘>’ (more than)
    5) ‘<=’ (less than or equal to)
    6) ‘>=’ (more than or equal to)
- **value –** String, representing the value evaluating the query condition.
- **and/or flag –** String, linking queries with *and* or *or.* Values are ‘and’ or ‘or’.

Finally, when the query is ready to be executed, use the following:

```
.runQuery();
```

Example:

1) This example will display records where the *First* name is *Steve* or the *Last* name is *Atom*.

```
myGrid = new dax_dataGrid('People');
myGrid.showStatusBar(true);
myGrid.go();

// Create, build, and run query
myGrid.newQuery();
myGrid.addQuery('First', ‘=’, 'Steve', 'or');
myGrid.addQuery('Last', '=', 'Atom');
myGrid.runQuery();
```

Send custom values to 4D

Send custom value names and pairs to the back end using the following:

```
myGrid.addCustomValue(name, value);
```

- **name –** String, name of the custom value.
- **value –** Value of custom value.

Clear custom values using the following:

```
myGrid.clearCustomValues();
```

These values are retrieved by the GET WEB FORM VALUES command in 4D.

Example:

1) This is a block of code that would perform a query during the user click of an input button.

JavaScript portion:

```
// clear all existing values
myGrid.clearCustomValues();

// get values from input fields
var myCity = $('myCity').value;
var myState = $('myState').value;

// add values to the query
myGrid.addCustomValue('myCity', myCity);
myGrid.addCustomValue('myState', myState);

// run query with stored custom values to 4D
myGrid.runQuery();
```

HTML portion:

```
My City: <input type="text" id="myCity" />
My State: <input type="text" id="myState" />
```

Get custom values from 4D

Use this command to return information from 4D back to the browser:

```
myCustomValues = myGrid.getCustomValuesFrom4D();
```

A set of web variables are passed from 4D to the front end using the 4D method *Dax_Dev_SetCustomVariables.* This method can pass variables to the front end during the following Developer Hooks:

Dax_DevHook_OnQuery (non-DCS support only)
Dax_DevHook_QueryFilter (non-DCS support only)
Dax_DevHook_DCS_SetSelection (DCS support only)

Dax_Dev_SetCustomVariables passes two text arrays to the front end:

$1 – Pointer to a Text Array containing variable **names**
$2 – Pointer to a Text Array containing variable **values**

Example:

```
  ARRAY TEXT($varNames_at;3)
  ARRAY TEXT($varValues_at;3)
```

```
 $varNames_at{1}:="v1"
 $varNames_at{2}:="v2"
 $varNames_at{3}:="v3"
 $varValues_at{1}:="aaa"
 $varValues_at{2}:="bbb"
 $varValues_at{3}:="ccc"
 DAX_Dev_SetCustomVariables (->$varNames_at;->$varValues_at)
```

Important:
1. The size of the 2 arrays must be the same, otherwise, the size of the value array will be adjust to the size of the name array.

2. If the method is called more than once, the one before the last execution will be used.

On the front end side, the object received has the following properties:

* **myCustomValues.length** -> number of returned values

Name/value pair, where valueNumber is an integer starting from 0 and above:

* **myCustomValues[valueNumber].name**
* **myCustomValues[valueNumber].value**

Example:

1) Here is a portion of code where the Data Grid is retrieving custom values from 4D to process column totals in the footer.

```
//Fill in the footer row with the custom value returned from the backend.
myDataGrid.onBeforeDataDisplay = function() {
var custValue = this.getCustomValuesFrom4D();
var len = custValue.length;
for (var i=0; i<len; i++){
       this.setCellValue (realRowNumber, i+1, 'Total of<br>' +
custValue[i].value);
}
};
```

This example can be further inspected in the *Header and Footer* example of the *Data Grid* sample database,

Query all records

Use this command to query all records in the Data Grid.

```
myGrid.queryAllRecords();
```

Example:

1) This call will cause the Data Grid to display all records.

```
myGrid.queryAllRecords();
```

Max number of characters per field

Use the command to set the maximum amount of characters 4D will send per field.

```
myGrid.querySetMaxChar(number of characters);
```

number of characters – Integer, representing the max number of characters 4D will send per field.

Example:

1) In this example the maximum number of characters a field can display is 15.

```
myGrid.querySetMaxChar(15);
```

Sort

Sort records using the following:

```
myGrid.sort(field, order);
```

* **field** - field name or id
* **order** – (optional) 'asc' for ascending order or 'desc' for descending order. Default value is 'asc'.

Example:

1) In this example the *First* name field is sorted in ascending order.

```
myGrid = new dax_dataGrid('People');
myGrid.go();

// Sort
myGrid.sort('First', 'asc');
```

On Before and On After Sort

Here are two events that can be used before or after sorting occurs.

Before sorting:

```
myGrid.onBeforeSort = function() { };
```

After sorting:

```
myGrid.onAfterSort = function() { };
```

Before data display

This event occurs when data arrives from 4D, but it's not displayed yet. This would be an ideal time to perform data manipulation before it gets displayed on the Grid.

```
myGrid.onBeforeDataDisplay = function() { };
```

Here is a recommendation of commands to consider executing during this event:

- getParsedDataValue
- setParsedDataValue
- getCustomValuesFrom4D

Check values before display

Get values before they are displayed:

```
myGrid.getParsedDataValue(record (row), field (column));
```

where record and field are integers that start from 0. Record & field correspond to row & column number.

Modify values before display

Change values before they are displayed:

```
myGrid.setParsedDataValue(record (row), field (column), value);
```

where record and field are integers that start from 0. Record & field correspond to row & column number.

Example:

1) This example will get data back from 4D, and capitalize every true value in the [Employee]IsManager field. This happens before data is displayed inside *onBeforeDataDisplay* handler.

```
// declare Data Grid variable
var myGrid;

// event that fires when user is successfuly logged in
function dax_loginSuccess() {

// create the grid as a floating window
myGrid = new dax_dataGrid('Employee');

// loop through values of [Employee]isManager field and make text
capitalized if employee is a manager
myGrid.onBeforeDataDisplay  = function() {

// find column number for [Employee]isManager field
var fieldId = dax_getField('Employee', 'IsManager').fieldid;
var columnNumber = this.getColumnByFieldId(fieldId);

// since this grid has an extra column at the beginning (control column),
subtract one to get right column number for get/setParsedDataValue
// same should be done if locked columns are present on the left side
columnNumber--;
// loop through all received records
for (var recordCount = 0; recordCount < this.getParsedDataRecordCount();
recordCount++) {

// get value
var value = this.getParsedDataValue(recordCount, columnNumber);

// if value is true, make it uppercase
if (value == 'true') {
      value = value.toUpperCase();
      this.setParsedDataValue(recordCount, columnNumber, value);
      }
}
}

// initialize and show grid
myGrid.go();
}
```

Get number of returned records

Use this command to get the number of records returned from 4D.

```
myGrid.getParsedDataRecordCount();
```

1) This example will get the number of records from 4D and display it in an alert message. This number should match the number of records displayed in the Status Bar.

```
myGrid = new dax_dataGrid('People');
// Show Status Bar to display number records
myGrid.showStatusBar(true);
myGrid.go();

// Get number of returned records during onDataLoad event
myGrid.onDataLoad = function() {
var reCount = myGrid.getParsedDataRecordCount();
alert(reCount);
```

Selection

Select row by record id

Highlight a row based on its Record ID.

```
myGrid.selectRowByRecordId(recordId);
```

recordId – String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

Example:

1) This example will highlight a record with ID [1][2]. The record is highlighted during the *onDataLoad* event since records do not exist until this point in time.

```
myGrid = new dax_dataGrid('People', null);
myGrid.go();

// Select Record [1][2] during the onDataLoad event
myGrid.onDataLoad = function() {
       myGrid.selectRowByRecordId('[1][2]');
}
```

Select row by number

Select a row based on its row number.

```
myGrid.selectRow(rowNumber);
```

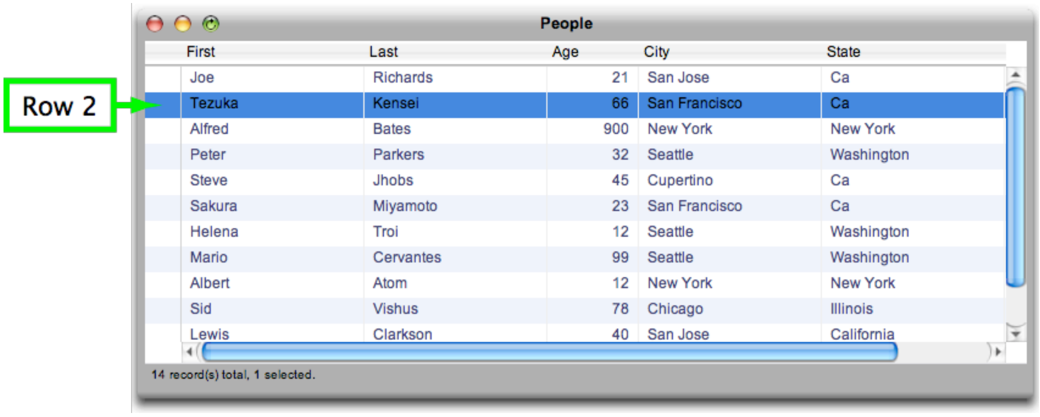
rowNumber – Integer, representing the row number. Numbering begins at zero so the top-most row (the header) is denoted as row 0 (zero).

The header, however, cannot be highlighted by this command.

Example:

1) In this example row 2 is selected.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();
// Select Row 2
myGrid.selectRow(2);
```



People

First	Last	Age	City	State
Joe	Richards	21	San Jose	Ca
Tezuka	Kensei	66	San Francisco	Ca
Alfred	Bates	900	New York	New York
Peter	Parkers	32	Seattle	Washington
Steve	Jhobs	45	Cupertino	Ca
Sakura	Miyamoto	23	San Francisco	Ca
Helena	Troi	12	Seattle	Washington
Mario	Cervantes	99	Seattle	Washington
Albert	Atom	12	New York	New York
Sid	Vishus	78	Chicago	Illinois
Lewis	Clarkson	40	San Jose	California

14 record(s) total, 1 selected.

Add, remove, and clear

Use these commands to add, remove, or clear records to a selection.

To add a record to a selection:

```
myGrid.addRecordToSelection(recordId);
```

recordId - String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

To remove a record from the selection:

```
myGrid.removeRecordFromSelection(recordId);
```

recordId - String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

To clear all records from the selection:

```
myGrid.removeAllRecordsFromSelection();
```

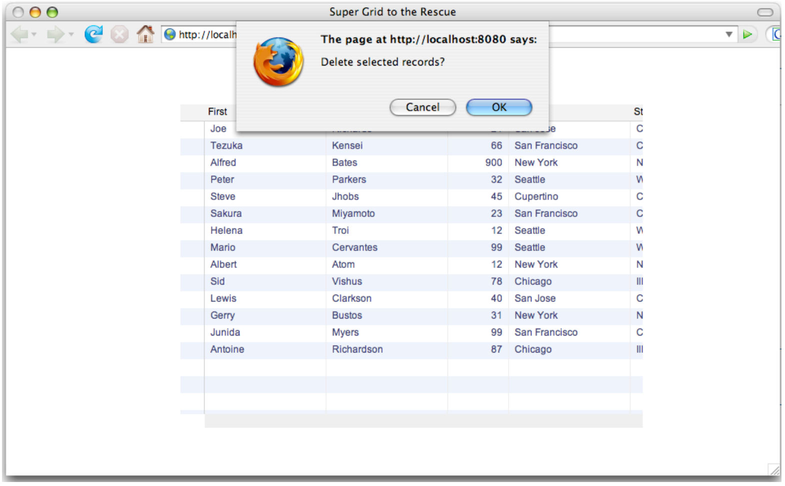
Delete selection

```
myGrid.deleteSelectedRecords(skipConfirmation);
```

Example:

1) In this example we delete a record with the *skipConfirmation* parameter set to *False.*

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();

// Add 1 record to the Selection
myGrid.addRecordToSelection(1,6);

// Delete the Selection. skipConfirmation set to false.
myGrid.deleteSelectedRecords(false);
```



The page at http://localhost:8080 says:

Delete selected records?

First				St
Joe	[illegible]	[illegible]	[illegible]	C
Tezuka	Kensei	66	San Francisco	C
Alfred	Bates	900	New York	N
Peter	Parkers	32	Seattle	W
Steve	Jhobs	45	Cupertino	C
Sakura	Miyamoto	23	San Francisco	C
Helena	Troi	12	Seattle	W
Mario	Cervantes	99	Seattle	W
Albert	Atom	12	New York	N
Sid	Vishus	78	Chicago	Il
Lewis	Clarkson	40	San Jose	C
Gerry	Bustos	31	New York	N
Junida	Myers	99	San Francisco	C
Antoine	Richardson	87	Chicago	Il

Selection Mode

Defines how many rows can be selected at a time with a row click.

```
myGrid.setSelectionMode(selectionMode);
```

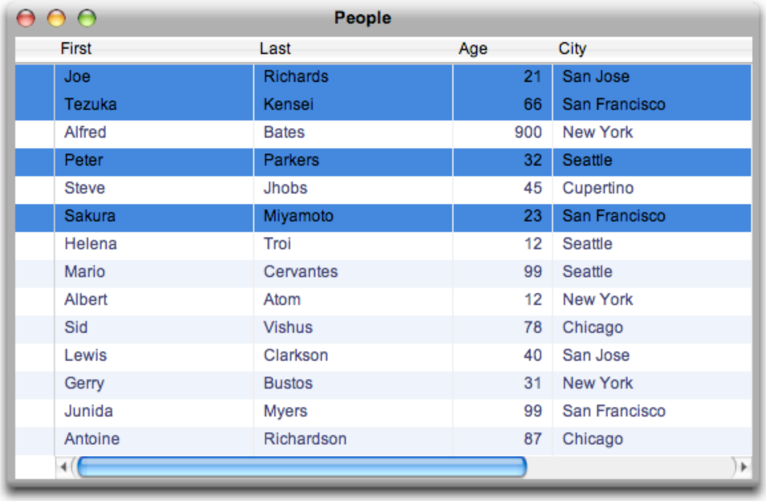
- **selectionMode** – Possible values are 'multi', 'single', or 'none'. 'multi' is chosen by default.
  - **multi –** Multiple records can be selected at the same time. Use CTRL key (PC) or CMD key (Mac) to add or remove records from the Selection.
  - **single –** Only one record can be selected at the time.
  - **none -** No selection is allowed.

These values are case sensitive. Thus, they must be lower-case.

Example:

1) This example sets the Selection Mode to 'multi'.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.go();
// Set to Multi Selection Mode
myGrid.setSelectionMode('multi');
```



People

First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

Drag & drop

Events

3 Drag and Drop events are defined for developers:

The **ondragover** event is when a cell is held and is hovering over an area to be dropped. Use this command to provide a visual indication that the user’s mouse is hovering over a cell. Use CSS style class names to provide the visual queue.

```
myGrid.ondragover(cellRef); → return CSS style class name to override default,
or null to skip visual indication
```

The **ondragout** event is when the original cell is first grabbed.

```
myGrid.ondragout(cellRef);
```

The **ondragrelease** event is when the original cell is dropped into the target area.

```
myGrid.ondragrelease(cellRef);
```

Example:

1) This is an example of Drag and Drop between two Data Grids. Cells from *gridTwo* can be dropped into *gridOne*.

   When a cell is hovering over another, the target cell is highlighted in red. When a cell is dropped, alerts appear announcing the statistics (row number, column number, cell value) of the original and target cells.

   JavaScript portion of code:

```
var gridOne;
var gridTwo;

function daxLoginSuccess() {

// create first grid, this will be the target for drag & drop action
gridOne = new dax_dataGrid ('Employee', $('gridOne'), 1, 0, false);
gridOne.setSelectionMode('none');
// give custom CSS class to visually show if cell is selectable
gridOne.ondragover = onDragOverEvent;

// process drag & drop
gridOne.ondragrelease = onDragReleaseEvent;

gridOne.go();

// create second grid, cells will be dragged from here
gridTwo = new dax_dataGrid ('Branch', $('gridTwo'), 1, 0, false);
// disable selection
gridTwo.setSelectionMode('none');
gridTwo.go();
// make 6th column draggable
gridTwo.setDragCells(null, 5, true);
}

// set css class for cell over which mouse is hovering
// returning null means that there will be no visual indication
function onDragOverEvent(cellRef, event) {
       return 'dragdrop_style';
}

// mouse is let go over the cellRef cell, display cell row, column, and
// value on success
function onDragReleaseEvent(cellRef, event) {
var originCell = dax_bridge.hoverDragging.dragObjectSource;

var targetCellInfo = 'Cell(' + cellRef.row + ',' + cellRef.column + ')
has value ' + cellRef.value;
var originCellInfo = 'Cell(' + originCell.row + ',' + originCell.column +
') has value ' + originCell.value;
}
```

CSS portion:

```
.dax_datagrid .dragdrop_style {
       background-color: red;
}
```

HTML portion:

```
<div id="gridOne" style="width: 500px; height: 300px;"></div>
<div id="gridTwo" style="width: 500px; height: 150px;"></div>
```

Set cells as drop zone

A Drop Zone is an area where an object can be drag and dropped into. Use this command to set specific rows or columns as a drop zone.

```
myGrid.setDropCells(row, column, isDroppable)
```

- **row –** Specify row to be set as a drop zone. Pass *null* for all rows to be set as a drop zone. Numbering begins at 0 (zero) so the top-most row (the header) is row 0 (zero).
- **column –** Specify column to be set as a drop zone. Pass *null* for all columns to be set as a drop zone. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).
- **isDroppable –** Boolean. Enter *True* to set area as droppable. Enter *False* otherwise.

Examples:

1) Row 3 is a drop zone.

```
myGrid.setDropCells(3, null, True);
```

2) Column 4 is a drop zone.

```
myGrid.setDropCells(null, 4, True);
```

3) Cell (3,4) is a drop zone.

```
myGrid.setDropCells(3, 4, True);
```

4) All rows and columns are drop zones.

```
myGrid.setDropCells(null, null, True);
```

Set cell as draggable

Set cells, rows, or columns as draggable using this command.

```
myGrid.setDragCells(row, column, isDraggable);
```

- **row –** Specify row to be set as a draggable. Pass *null* for all rows to be set as a drop zone. Numbering begins at 0 (zero) so the top-most row (the header) is row 0 (zero).
- **column –** Specify column to be set as a draggable. Pass *null* for all columns to be set as a drop zone. Numbering begins at 0 (zero) so the left-most column is column 0 (zero).
- **isDraggable –** Boolean. Enter *True* to set area as draggable. Enter *False* otherwise.

Draggable cells will not support mouse hover and selection mechanics. No cells are set as draggable by default.

During the drag & drop process, source cell is also accessible through:

```
dax_bridge.hoverDragging.dragObjectSource
```

Set draggable object

To make any object on the HTML page draggable, call this command.

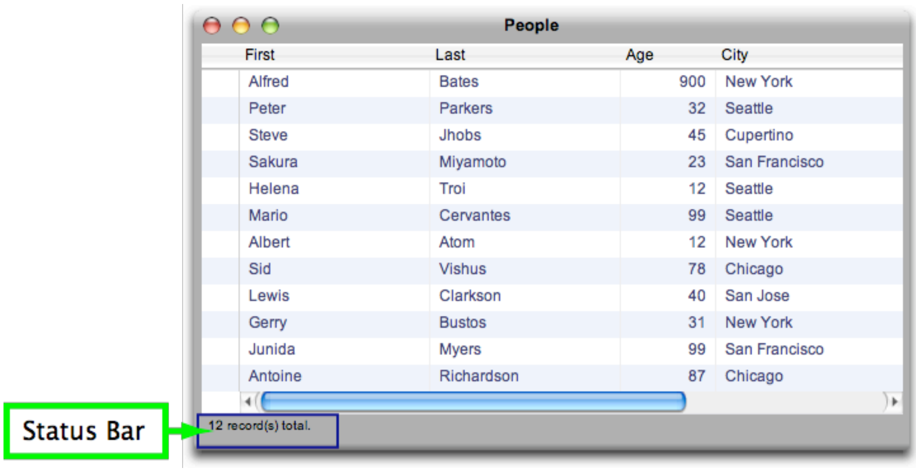
```
dax_setDraggable(draggableObject);
```

This will not move the actual object, but create a floating copy when dragged.

Window

Status bar

The Status Bar is the bar at the bottom of the Data Grid displaying status information.



First	Last	Age	City
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

12 record(s) total.

By default the Status Bar displays generic information such as the total number of records or the amount of records currently selected.

Show or hide status bar using the following command.

```
myGrid.showStatusBar(boolean);
```

Example:

1) This example will hide the Status Bar. The status bar is hideable/showable even after grid is drawn, but hiding it before .go() phase is more efficient since no redraw is required.

```
var myGrid = new dax_dataGrid ('Employee');

// hide the Status Bar
myGrid.showStatusBar(false);

myGrid.go();
```

Show message

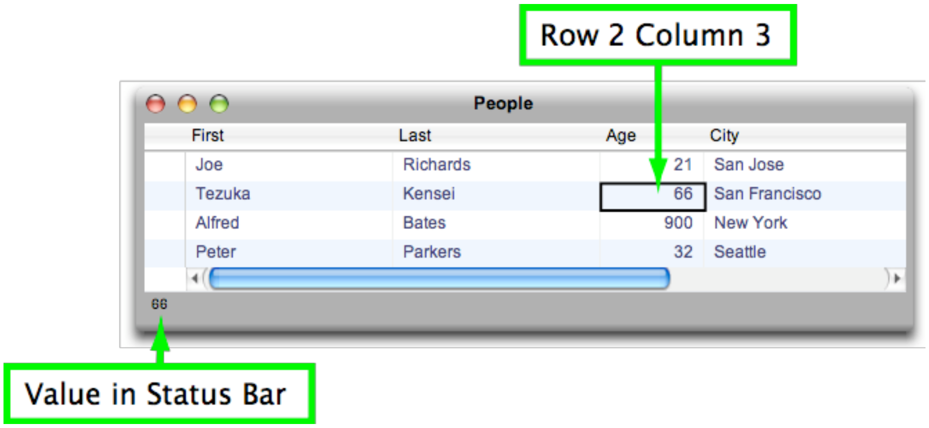
Use this command to display values in the Status Bar.

```
myGrid.showStatusMessage(message);
```

Example:

1) In this example we will get the value of a cell and display it in the Status Bar. We get the cell value during the *onDataLoad* event because the Data Grid cells do not have values until this event occurs.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.showStatusBar(true);
myGrid.go();

myGrid.onDataLoad = function() {
       var cellValue = myGrid.getCellValue (2, 3);
       myGrid.showStatusMessage(cellValue);
}
```



First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle

Toolbar

The Toolbar is a bar above the Data Grid that features the following capabilities:

- Create a new record
- Delete selected record(s)
- Search based on field

To show or hide the Toolbar use the following commands

```
myGrid.showToolbar([Toolbar feature]);
myGrid.hideToolbar([Toolbar feature]);
```

Toolbar feature – Values can be:

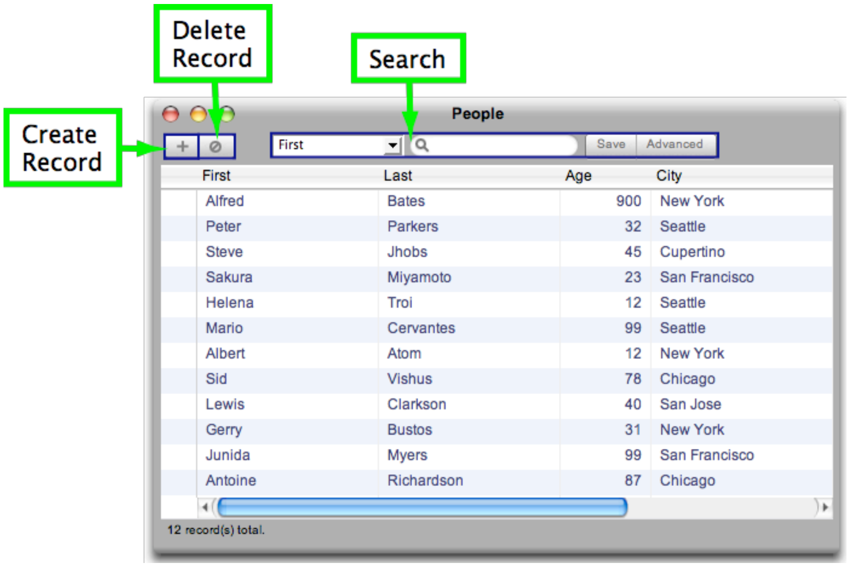
- 'createrecord' – This will display a button that allows users to create new records.
- 'deleterecords' – This is will display a button that allows users to delete the records that are selected.
- 'search' – This will display the

The features are displayed on the Data Grid from left to right in the order they are listed in the *.showToolbar* command.

Example:

1) This example will display all Toolbar features.

```
myGrid.showToolbar(['createrecord','deleterecords','search']);
```



First	Last	Age	City
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

12 record(s) total.

Preset queries

Preset queries are queries created in the control panel via the Query Manager tab. These queries can be represented as **Sidebars** or **Tabs**.

Sidebar

The Sidebar is an area on the Data Grid displaying the Preset Queries a list.

Show and hide Sidebar

To show or hide the Sidebar:

```
myGrid.showQuerySidebar(position);
myGrid.hideSidebar();
```

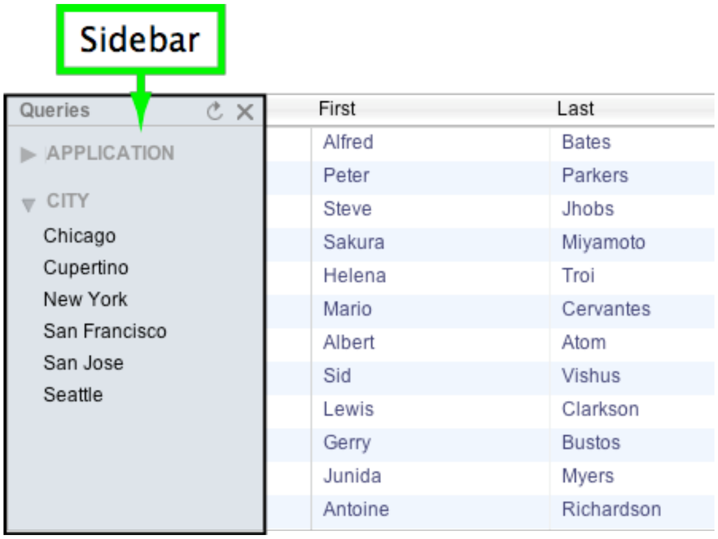
position – (optional) 'left' or 'right'. The Sidebar appears to the left by default.

Field(s) used for Dynamic Queries must have at least one of the visibility checkboxes turned on in the control panel.

Example:

1) This example displays the Sidebar on the left.

```
myGrid = new dax_dataGrid('People');
myGrid.go();
// Show Sidebar
myGrid.showQuerySidebar();
```



First	Last
Alfred	Bates
Peter	Parkers
Steve	Jhobs
Sakura	Miyamoto
Helena	Troi
Mario	Cervantes
Albert	Atom
Sid	Vishus
Lewis	Clarkson
Gerry	Bustos
Junida	Myers
Antoine	Richardson

Tabs

Preset Queries can also appear as Tabs above the row and column area of the Data Grid.

Show and hide Tabs

To show or hide the Tabs:

```
myGrid.showQueryTabs();
myGrid.hideQueryTabs();
```

Example:

1) This example will display Preset Queries as Tabs:

```
myGrid.showQueryTabs();
```

Preset queries

To run individual Preset Queries by query name:

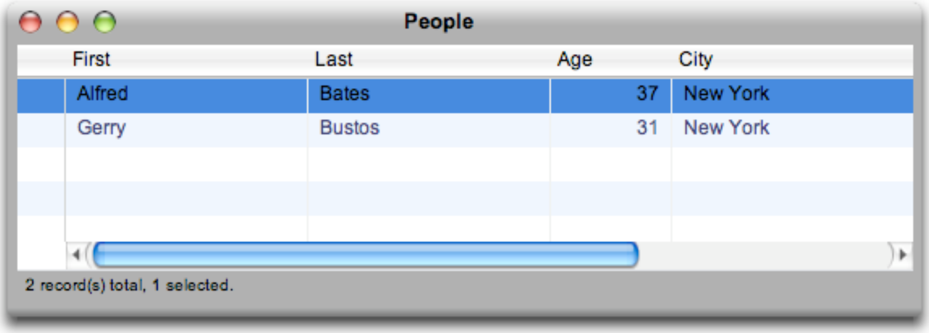
```
myGrid.runPresetQuery (name, field);
```

- **name** – String, name given to the Preset Query as denoted in the Query Manager tab within the Client.
- **field** - Field name, needed only for unique value queries.

Example:

1) This example will run a Preset Query named *B* which will display all records in the *Last* name field which start with the letter B.

```
myGrid = new dax_dataGrid('People');
myGrid.go();

//
myGrid.runPresetQuery ('B');
```

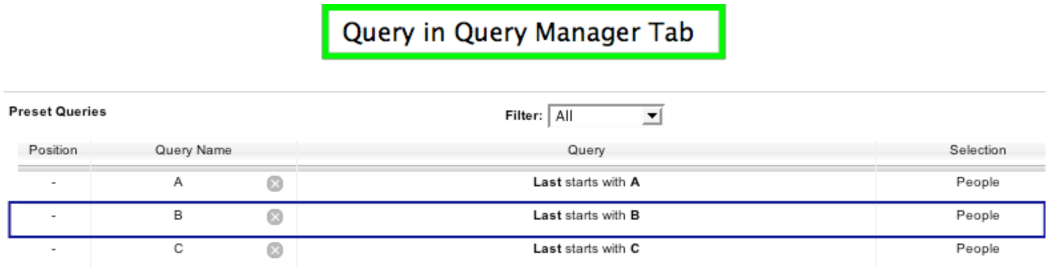


People

First	Last	Age	City
Alfred	Bates	37	New York
Gerry	Bustos	31	New York

2 record(s) total, 1 selected.

Below is a screenshot of the query as seen in the Query Manager Tab in the Client environment.



Query in Query Manager Tab

Preset Queries

Filter: All

Position	Query Name		Query	Selection
-	A	⊗	**Last** starts with **A**	People
-	B	⊗	**Last** starts with **B**	People
-	C	⊗	**Last** starts with **C**	People

Editor

The Editor is a sheet that appears that allows users to modify values or add values for a new record in the Data Grid.

New record

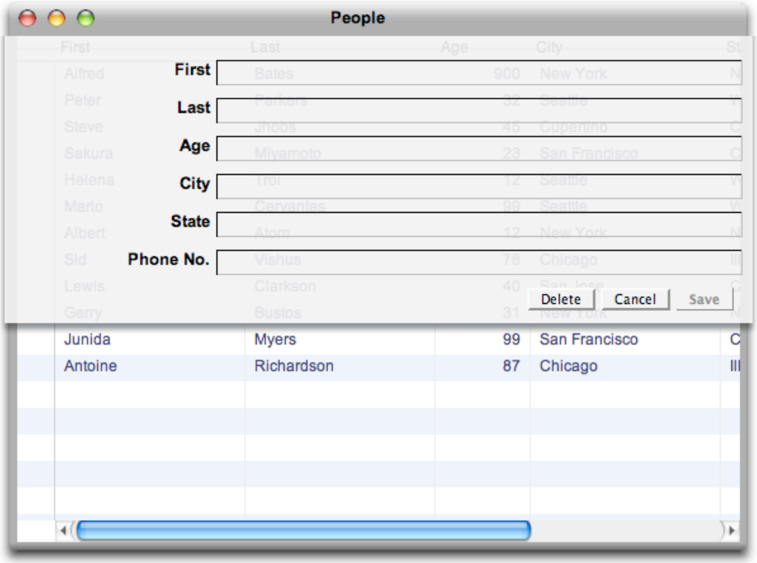
Use the new record command to display the Editor sheet for adding a new record.

```
myGrid.editorNewRecord();
```

Example:

1) This example displays the Editor sheet for a new record:

```
myGrid = new dax_dataGrid('People');
myGrid.go();

// Display Editor sheet for a new record
myGrid.editorNewRecord();
```


People
First
Last
Age
City
State
Phone No.
Delete
Cancel
Save
Junida Myers 99 San Francisco C
Antoine Richardson 87 Chicago Ill

Edit record

Use the following command to modify an existing record using the Editor sheet, based on its record ID.

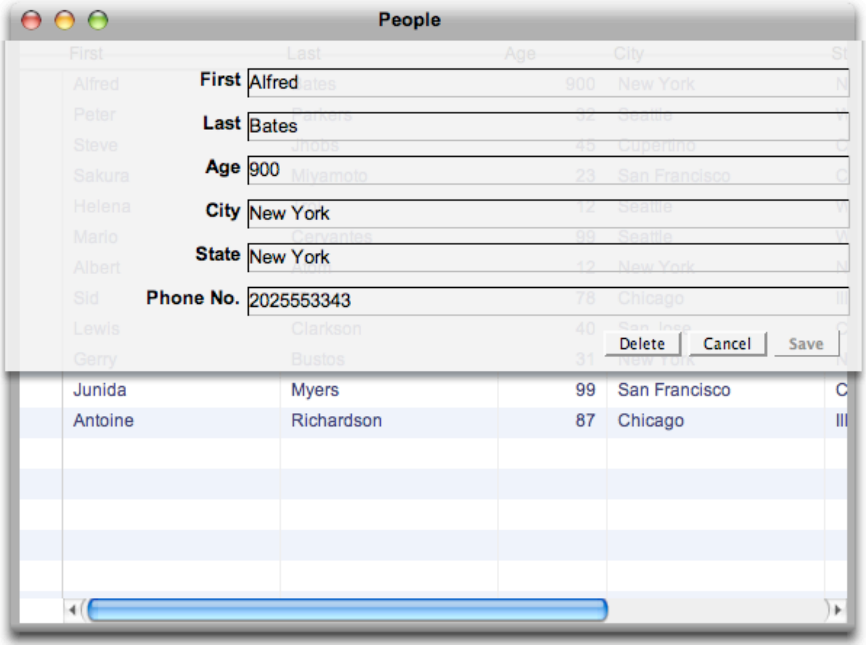
```
myGrid.editorEditRecord(recordId);
```

recordId – String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

Example:

1) This example will display the Editor sheet for the record with ID [1][2]:

```
myGrid.editorEditRecord('[1][2]');
```



First Alfred

Last Bates

Age 900

City New York

State New York

Phone No. 2025553343

First	Last	Age	City	St
Junida	Myers	99	San Francisco	C
Antoine	Richardson	87	Chicago	Il

Allow editor

This command disables or enables the record editor to appear when users double click on a row. The record editor is enabled by default.

```
myDataGrid.allowEditor(boolean);
```

boolean – Set *true* to enable the record editor. Set *false* otherwise.

Inline editing

Typically, to modify records the user would double click on a record and an Editor sheet would appear allowing them to edit field values. Inline Editing, on the other hand, allows the user to modify or add records in the Data Grid without having to do so in another sheet, but in the Data Grid itself.

The Inline Editor supports:

- Choice Lists: Fields that are assigned Choice Lists in 4D automatically behave in the same way in the Data Grid.
- Keyboard Shortcuts:
  - Hit the *Esc* key to Cancel editing a record.
  - Hit the *Tab* key to go to the next field.
  - Hit the *Return* key to Save the record.
- A Calendar Picker appears for Date fields.
- Boolean fields appear as check boxes. The values depend on the formatting rules selected in the Control Panel.
- Callbacks: 4D Callbacks are supported during Inline Editing.

Allow inline editing

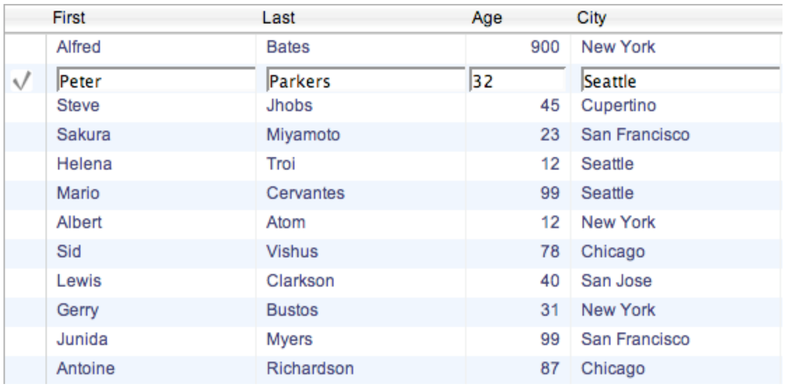
```
myGrid.allowInlineEditing(boolean);
```

boolean – Set to *true* to allow Inline Editing. Set to *False* otherwise.

Example:

1) In this example Inline Editing is set to *true.* Below is a screen shot of a record being edited.

```
myGrid.allowInlineEditing(true);
```



	First	Last	Age	City
	Alfred	Bates	900	New York
✓	Peter	Parkers	32	Seattle
	Steve	Jhobs	45	Cupertino
	Sakura	Miyamoto	23	San Francisco
	Helena	Troi	12	Seattle
	Mario	Cervantes	99	Seattle
	Albert	Atom	12	New York
	Sid	Vishus	78	Chicago
	Lewis	Clarkson	40	San Jose
	Gerry	Bustos	31	New York
	Junida	Myers	99	San Francisco
	Antoine	Richardson	87	Chicago

Initiate inline editing

To initiate Inline Editing programmatically use the following command.

```
myGrid.inlineEdit(editRow, recordId);
```

Pass one or none of the parameters:

* **editRow** – Integer, representing the row number to edit. Number starts from 0 (zero).
* **recordId** – String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

If no parameters are passed, inline edit will be blank and save a new record.

Example:

1) Edit 4th row:

```
// edit 4th row
myDataGrid.inlineEdit(3);
```

2) Edit Record ID [3][4]:

```
// edit record id [3][4]
myDataGrid.inlineEdit(null, '[3][4]');
```

3) Create a new record through Inline Editing:

```
myDataGrid.inlineEdit();
```

Save edited record

Use this command to programmatically save the current record that is being edited through Inline Editing.

```
myGrid.inlineEditSave();
```

Cancel inline editing

Use this command to programmatically clear the current record that is being edited through Inline Editing.

```
myGrid.inlineEditClear();
```

Events

On cell, row, column click

To assign custom functions to the column, row, or cell click events, assign them to the respective grid functions.

Each time a column is clicked use:

```
myGrid.onDataColumnClick = function myFunction(column, fieldReference);
```

Each time a row is clicked use:

```
myGrid.onDataRowClick = function myFunction(row, recordId);
```

Each time a cell is clicked use:

```
myGrid.onDataCellClick = function myFunction(row, column, recordId,
fieldReference);
```

Parameters:

* **column** and **row** are integers, starting from 0.
* **fieldReference** - field object, with fieldid, fieldalias, etc attributes.
* **recordId** – String. Unique value defined for each record denoted by two values *x* and *y* in the format '[x][y]'.

Returning false will prevent default data grid actions (selection, for example).

Example:

1) In this example data in each cell is being converted to UPPERCASE each time it is clicked. The *onDataCellClick* event is used to catch the user clicks.

```
var myGrid = new dax_dataGrid('People');
myGrid.go();

// when cell is clicked, show the value in status bar
myGrid.onDataCellClick = function(row, column) {

// get cell value
var myCellValue = myGrid.getCellValue(row, column);

// convert value to upper case
myCellValue = myCellValue.toUpperCase();

// place the value back into the cell
myGrid.setCellValue(row, column, myCellValue);
}
```

On data load

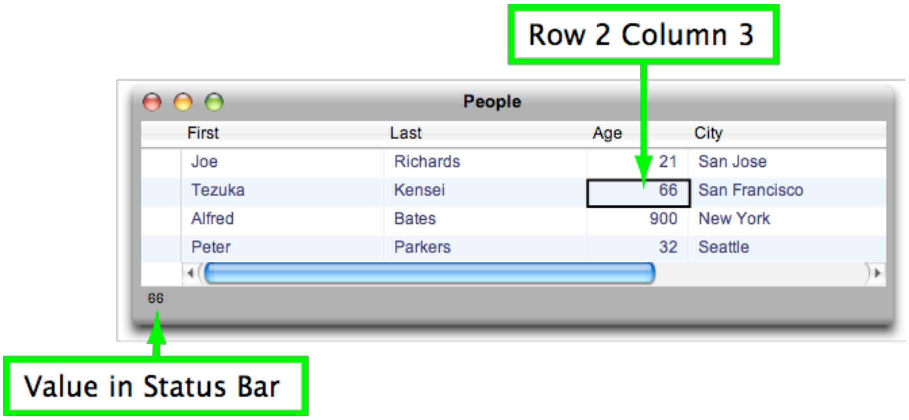
This is the time when data populates the Data Grid. Note that many other Data Grid commands are data dependent, so in many cases *onDataLoad* would be the only appropriate time to use them.

```
myGrid.onDataLoad = function() { };
```

Example:

1) In this example we will get the value of a cell and display it in the Status Bar. We get the cell value during the *onDataLoad* event because the Data Grid cells do not have values until this event occurs.

```
var myGrid = new dax_dataGrid('People', null);
myGrid.showStatusBar(true);
myGrid.go();

myGrid.onDataLoad = function() {
        var cellValue = myGrid.getCellValue (2, 3);
        myGrid.showStatusMessage(cellValue);
}
```



Row 2 Column 3

People

First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle

66

Value in Status Bar

Before and After Hover

When the mouse hovers over a ‘View Image’ link for a picture field, a preview of that image appears. The following two events fire for this action, *onBeforeHover* and *onAfterHover*. They are shown below.

This event fires when the mouse first hovers over the link.

```
myGrid.onBeforeHover = function() { };
```

This event fires when the mouse hovers away from the link.

```
myGrid.onAfterHover = function() { };
```

Example:

1) In this example, a blank image in a separate <div> displays the image that is previewed when it is being hovered over. When the mouse leaves the link, the <div> becomes blank again.

   This is the JavaScript code in *onAfterInit*:

```
myGrid = new dax_dataGrid('Contacts', $('mydiv'), 1, 0, false);
myGrid.enablePicturePreview(true);
myGrid.go();
myGrid.allowEditor(false);

// Display the contents of the image being hovered into the image element
// on the page with ID “mypreview”
myGrid.onBeforeHover = function(athis){
       var imgcontainer = document.getElementById('mypreview');
       imgcontainer.src = athis.aContent;
       imgcontainer.width = athis.parent.picWidth;
       imgcontainer.height = athis.parent.picHeight;
}

// Display nothing in the image element that has the ID “mypreview” when
// the mouse hovers away from the link.
myGrid.onAfterHover = function(athis){
       var imgcontainer = document.getElementById('mypreview');
       imgcontainer.src = 'blank.png';
}
```

   This is the HTML with the <div> containing the image element with ID “mypreview”.

```
<div style="float: right; width: 213px; margin-left: 35px; margin-top:
35px;">
       <img id="mypreview" src="blank.png" width="0" height="0"/>
</div>
```

CSS Styling

Change the look of the Data Grid's rows, columns, and cells using CSS. There are two ways of approaching this:

- Override the framework's existing styling
- Use JavaScript to dynamically assign styling

Override Styling

To override the framework's CSS styling, it helps to know the following CSS classes available based on data type:

- **.datanumeric –** Numeric fields
- **.datatext –** Text fields
- **.datatime –** Time fields
- **.datadate –** Date fields
- **.databoolean –** Boolean fields
- **.dataheader -–** Headers

The point here is that the framework already has CSS styling defined for *.dax_datagrid.* To override the existing styling, define CSS classes prefixed with *.dax_datagrid.*

Example:

1) To override the 'text-align' attributes for the Data Grid add this CSS code:

```
<style type="text/css">

.dax_datagrid .datanumeric, .dax_datagrid .datatext{
text-align:center;
}

</style>
```

Note that *.dax_datagrid* is prefixed before these data types are specified.

This code would override the text-align attributes for all numeric and text fields in the grid.

Set entire grid class

This command appends the class name to the grid. This means that any style that starts with *.myClassName* can be applied for this specific grid.

```
myGrid.setGridClass(className);
```

Example:

1) Following the same example attempted above, *setGridClass* can be used if you prefer to define your own styling instead of overriding the framework's CSS styling for *.dax_datagrid.*

CSS portion:

```
<style type="text/css">
.style1 .datatext, .style1 .datanumeric {font-size: 11pt; text-
align:center; font-weight:bold;}
</style>
```

JavaScript portion:

```
myDataGrid.setGridClass('style1');
```

Like the previous example, this sample will make numeric and text fields center aligned (as well as having bolder text and a modified font size).

Assign classes to Rows, Columns, and Cells

The following commands change CSS styling more dynamically (for example, on a user click).

Change CSS for a row:

```
myGrid.setRowStyle(row, styleName, alternateRowStyleName, useAsDefault);
```

- **row –** Integer, row number.
- **styleName –** String, name of CSS style to be applied.
- **alternateRowStyleName –** (optional) String, name of CSS style to be applied to alternate row.
- **useAsDefault –** (optional)

Change CSS for a column:

```
myGrid.setColumnStyle(column, styleName, alternateRowStyleName, useAsDefault);
```

- **column –** Integer, column number.
- **styleName –** String, name of CSS style to be applied.
- **alternateRowStyleName –** (optional) String, name of CSS style to be applied to alternate row.
- **useAsDefault –** (optional)

Change CSS for a cell:

```
myGrid.setCellStyle(row, column, styleName, alternateRowStyleName,
useAsDefault);
```

- **row –** Integer, row number.
- **column –** Integer, column number.
- **alternateRowStyleName –** String, name of CSS style to be applied to alternate row.
- **useAsDefault –** (optional)

Example:

1) This example will set CSS styling for row, column, and cell based on a user click. Event *onDataCellClick* is used to catch the user clicks. Buttons are also used to change grid class using *.setGridClass()*

   JavaScript code for *dax_loginSuccess():*

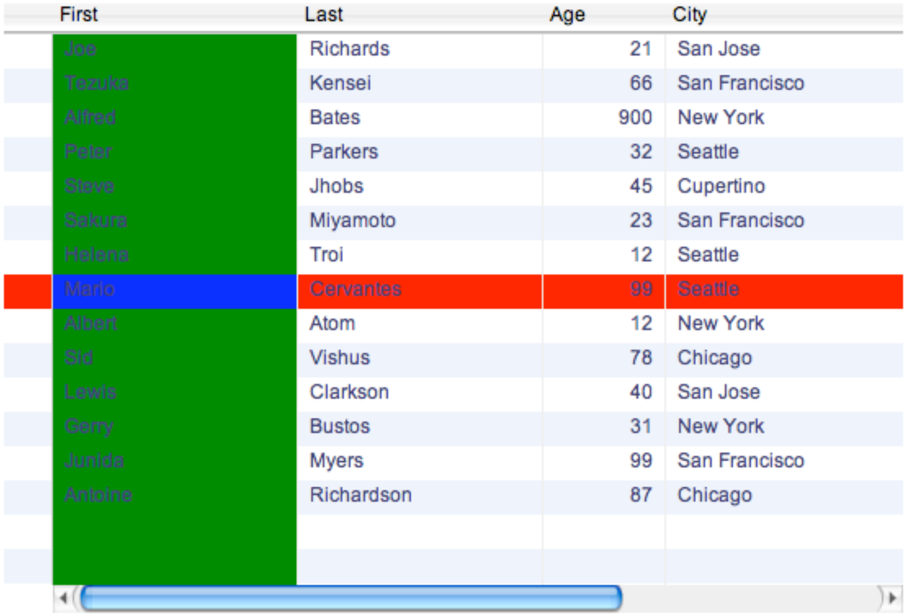
```
function dax_loginSuccess() {
var lastSelectedRow = 0;
var lastSelectedColumn = 0;

var myGrid = new dax_dataGrid('People', $('DataGridDiv'));
myGrid.go();

myGrid.setSelectionMode('none');

myGrid.onDataCellClick = function(row, column) {

// clear previous styles
myGrid.setRowStyle(lastSelectedRow, '');
myGrid.setColumnStyle(lastSelectedColumn, '');
myGrid.setCellStyle(lastSelectedRow, lastSelectedColumn, '');

//update values
lastSelectedRow = row;
lastSelectedColumn = column;

// set values
myGrid.setRowStyle(row, 'redColor');
myGrid.setColumnStyle(column, 'greenColor');
myGrid.setCellStyle(row, column, 'blueColor');
}

function button_changeGridStyle1() {
// check if grid exists. If it doesn't, skip the rest of the method
if (!myGrid)
       return false;

// Set the Data Grid’s CSS class to style inverseStyle.
myGrid.setGridClass('inverseStyle');
}

function button_changeGridStyle2() {
// check if grid exists. If it doesn't, skip the rest of the method
if (!myGrid)
       return false;

// The empty space here will equate to nothing, setting the Data Grid’s
// CSS class to default settings.
myGrid.setGridClass(' ');

}

// Assign events to the buttons

$('changeGridStyle1').onclick = button_changeGridStyle1;
$('changeGridStyle2').onclick = button_changeGridStyle2;

}
```

CSS code. Default Data Grid CSS class *.dax_datagrid* and CSS class *.inverseStyle* are defined here:

```
<style type="text/css">
       .dax_datagrid .redColor {
              background-color: red;
       }
       .dax_datagrid .greenColor {
              background-color: green;
       }
       .dax_datagrid .blueColor {
              background-color: blue;
       }
       .inverseStyle .redColor {
              background-color: purple;
       }
       .inverseStyle .greenColor {
              background-color: yellow;
       }
       .inverseStyle .blueColor {
              background-color: silver;
       }
</style>
```

Code within <Body>. The buttons used to change the Data Grid's CSS class as a whole are defined here:

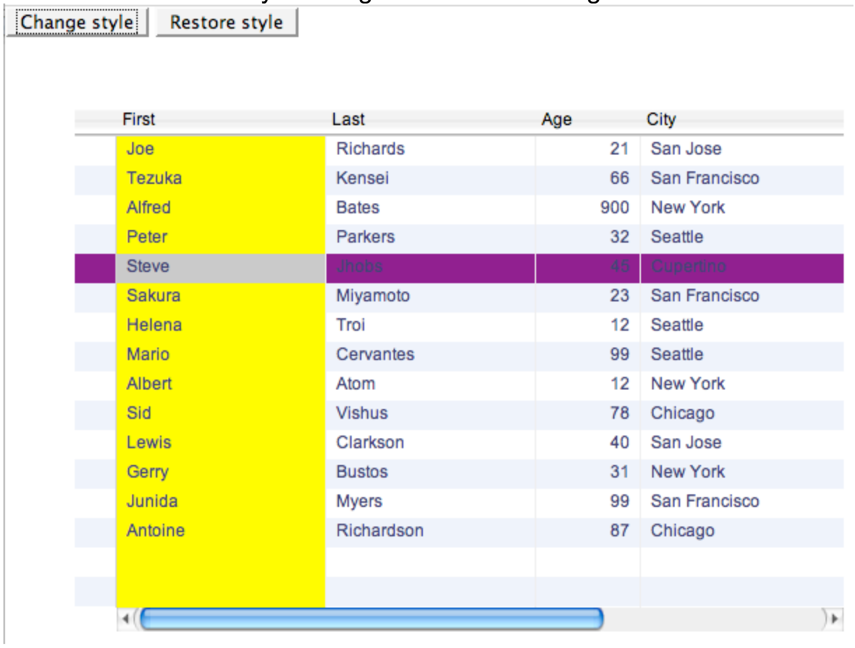
```
<input class="myButton" type="button" id="changeGridStyle1" value="Change
style" />
<input class="myButton" type="button" id="changeGridStyle2"
value="Restore style" />
```



First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

Based on a user click, we see:

- Cell style background color set to blue.
- Row style background color set to red.
- Column style background color set to green



First	Last	Age	City
Joe	Richards	21	San Jose
Tezuka	Kensei	66	San Francisco
Alfred	Bates	900	New York
Peter	Parkers	32	Seattle
Steve	Jhobs	45	Cupertino
Sakura	Miyamoto	23	San Francisco
Helena	Troi	12	Seattle
Mario	Cervantes	99	Seattle
Albert	Atom	12	New York
Sid	Vishus	78	Chicago
Lewis	Clarkson	40	San Jose
Gerry	Bustos	31	New York
Junida	Myers	99	San Francisco
Antoine	Richardson	87	Chicago

Hitting the *Change Style* button uses the *.setGridClass()* command to do the following:

- Cell style background color set to silver
- Row style background color set to purple.
- Column style background color set to yellow.

Hitting the *Restore Style* button will then use *.setGridClass()* to restore the CSS to it’s default settings.

Data Matrix

It's an image gallery! Or a data matrix! Customizable number of rows and columns, as well as live data from 4D! Requires image field for image browser mode and preferably a label field as well.

Create new object

```
myDataMatrix = new dataMatrix(targetNode, selectionName, header, imageField,
contents, zoomContents, layout, margin, zoomLevel, scrollMode, forceRows,
forceColumns, allowCellMouseInteraction, maxCharNumber, skipQueryOnStart);
```

Parameters

Overview of parameters, check below for more details.

- targetNode - in which DOM element would you like insert the image browser? If set to null, browser will pop up in a new window. Div tag with set width and height is highly recommended. Common way to refer to the HTML element is via unique id. For example, use $('myDiv') for DIV element with id "myDiv". Take a look at examples below to see more examples of this.
- selectionName - selection where data resides. Selection id is fine as well.
- header - header string, where [selection]field syntax will be replaced with actual data from corresponding field. Use auto to automatically set first available field as header.
- imageField - name or id of image field. Use auto to automatically find first available image field.
- contents - contents string, where [selection]field syntax will be replaced with actual data from corresponding field.
- zoomContents - same as contents, but displayed when cell is in zoomed-in state.
- layout - location of the image in image mode: top, bottom, left, or right. Top and bottom display header only, left and right display header and contents.
- margin - cell margin size
- zoomLevel - 0 (smallest possible cell size) and above. Adapts cell size to object based on this number.
- scrollMode - vertical or horizontal. Horizontal is recommended for one row object.
- forceRows - forces number of rows instead of using zoom level. Must be accompanied by forceColumns. Using this will disable change zoom level toolbar buttons ('more' and 'less').
- forceColumns - forces number of columns instead of using zoom level. Must be accompanied by forceRows.Using this will disable change zoom level toolbar buttons ('more' and 'less').
- allowCellMouseInteraction - allows for the cell content to be clickable (for example, if links or iframes are used), but also disables some of the animation.
- maxCharNumber - maximum number of characters per field when queried.
- skipQueryOnStart - if true, skip initial query

targetNode

1st parameter.

In order to displaye the dataMatrix object, you must specify the target location where the object will be displayed. This location must be created as a tag. Using CSS style to define tag width and height is highly recommended.

```
myMatrix = new dataMatrix ($('MyDiv'), ... ) // Create a data matrix
object in the div tag with that has id equals to 'MyDiv'
```

```
     <!-- div element where the data matrix will be inserted -->
     <div id="MyDiv" style="width: 650px; height: 500px; background:
#FFFFFF;">
```

selectionName

2nd parameter.

The selectionName refers to the selection that you created or the default selection created by 4D Ajax Framework. You can find out the selectionName simply by logging into the Administration Control Panel. All selection name can be found under Access Control tab (Real Name column). It is important that the selectionName is specified by the real name (not alias).

For example:

```
myMatrix = new dataMatrix ($('MyDiv'), 'Table1', ... ) // Create a data
matrix for a selection named "Table1"
```

header

3rd parameter.

The header parameter is used for declaring the label-header value of each cell. You can specify the header value as a plain text or html code. You can also take advantage of the automatic data population for the header.

```
     //Set header value

     var MyLabelHeader = 'John Smith';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader, ...
) // Set the header for all cell to John Smith


     //Set header value with html code

     var MyLabelHeader = '<b>John Smith<\/b>';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader, ...
) // Set the header for all cell to John Smith


     //Set header value with dynamic data

     var MyLabelHeader = '[Table1]Fullname';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader, ...
) // Get the actually value of [Table1]Fullname and use

// it as the label header.
```

Please note that the format for referencing the field selection is [selectionName]FieldName. Both selectionName and FieldName must be real names, not aliases.

imageField

4th parameter.

This parameter allows you to include the picture (referenced by a field name) of each record in a cell.

```
     // Include the photo of each record in each cell.

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, ... )
```

contents

5th parameter.

The contents parameter allows you to specify the detail of each cell in the matrix. It works in similar fashion as the header parameter. You can specify the contents as plain text, html code or dynamic data.

```
     // Set content value
     // Set the contents for all cell to "Very Nice Guy"

     var MyContent = 'Very Nice Guy';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, ... )


     // Set content value with html code
     // Set the contents for all cell to "Very Nice Guy"

     var MyContent = '<b>Very Nice Guy<\/b>';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, ... )


    // Set header value with dynamic data
    // Get the actually value of [Table1]Comments and use it as the cell
detail.
     var MyContent = '[Table1]Comments';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, ... )
```

zoomContents

6th parameter.

This parameter is used only when the cell is zoomed in. If you specified this parameter, the value in default contents value will be replaced by the value in zoomContents. You can specify the contents as plain text, html code or dynamic data.

```
     // Set zoom content value
     // Set the zoom contents for all cell to "Very Nice Guy"

     var myZoomedContent = 'Very Nice Guy and Very Very Smart';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, ... )


     // Set content value with html code
     // Set the zoom contents for all cell to "Very Nice Guy"

     var myZoomedContent = '<b>Very Nice and Very Very Smart<\/b>';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, ... )
```

```
     // Set header value with dynamic data
     // Get the actually value of [Table1]Comments and use it as the cell
detail.

     var myZoomedContent = '[Table1]Comments';
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, ... )
```

layout

7th parameter.

This parameter is used only if the imageField parameter is specified. It allows you to set the display location of the image in the cell. Once the value is set, the contents or zoomContents value will appear in the opposite position.

```
     // Display the image on the left side of the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', ... )


     // Display the image on the right side of the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'right', ... )


     // Display the image on the top side of the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'top', ... )


     // Display the image on the bottom side of the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'bottom', ... )
```

margin

8th parameter.

This parameter sets the space between cell in the data matrix.

```
     // Set the space between cell to 2 pixels

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, ... )


     // Set the space between cell to 5 pixels

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 5, ... )
```

zoomLevel

9th parameter.

zoomLevel is a semi-automatic option. The data matrix use this value in conjunction with the size of the window to determine how many cell it can display in the data matrix. The lower the zoomLevel value is, the smaller the cell is going to appear in the data matrix.

Note: If you enter a non-null value, you need to define null as values for forceRows and forceColumns. If you enter null as value, you need to define non-null as values for forceRows and forceColumns.

scrollMode

10th parameter.

This parameter sets the scroll bar orientation for the data matrix object. The scroll mode can be set either as "ver" (vertical) or "hor" (horizontal). Vertical scrollbar is object default.

```
     // Enable vertical scrollbar

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, 3, 'ver', ... )


    // Enable horizontal scrollbar

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, 3, 'hor', ... )
```

forceRows

11th parameter.

This parameter works in conjunction with the parameter 12 (forceColumns). It allows you to force the number of rows to be displayed in the matrix object. If you specify a value in this parameter then, you should parameter 9 (zoomLevel) as null.

```
     // Set the data matrix to display 3 rows

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 3, ...
)


     // Set the data matrix to display 5 rows

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 5, ...
)
```

Note: If you enter a non-null value, you need to define a non-null value for forceColumns and a null value for zoomLevel. If you enter a null value, you need to define a null value for forceColumns and a non-null value for zoomLevel.

forceColumns

12th parameter.

This parameter works in conjunction with the parameter 11 (forceColumns). It allows you to force the number of cell to be displayed in the matrix object. If you specify a value in this parameter then, you should parameter 9 (zoomLevel) as null.

```
     // Set the data matrix to display 3 rows and 4 columns

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 3, 4,
... )
```

For example 2: Set the data matrix to display 5 rows and 2 columns

```
     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 5, 2,
... )
```

allowCellMouseInteraction

13th parameter.

```
This parameter allows you to control the interaction mode accept the true
or false value. true means the cell blocks the mouse from entering and
clicking on any link inside the cell. false means that there will be no
mouse interaction within the cell.
     // Allows mouse interaction inside the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 3, 4,
true)


     // Do not allow mouse interaction inside the cell

     myMatrix = new dataMatrix ($('MyDiv'), 'Table1', MyLabelHeader,
[Table1]Photo, MyContent, myZoomedContent, 'left', 2, null, 'ver', 5, 2,
false)
```

maxCharNumber

14th parameter.

Maximum number of characters per field returned when queried. The absolute maximum is the same as the maximum size of a Text field in 4D.

skipInitialQuery

15th parameter.

Skips initial query when object is created.

Simple call example

Simplest code possible, automatic header and image field selection

```
myDataMatrix = new dataMatrix (null, 'Contacts', 'auto', 'auto');
```

Object actions

Hibernate

Hibernate call will hide the object and pause all data updates with 4D. This is recommended instead of destroying and creating a new object in order to avoid potential web browser memory leak.

```
myDataMatrix.hibernate();
```

Use wakeUp call to reveal hibernated object again and resume usual operations.

```
myDataMatrix.wakeUp();
```

Auto Refresh

Data will automatically update after certain interval. Use command below to pause/continue refresh timer. When refresh is enabled, manually query data again after which timer will start ticking again.

```
myDataMatrix.useAutoRefresh(useRefresh);
```

Parameters:

- useRefresh - boolean.

Reinitialize

Changes some or all preferences defined when object was created.

myDataMatrix.reInit(selectionName, header, imageField, contents, zoomContents, layout, margin, zoomLevel, scrollMode, forceRows, forceColumns, allowCellMouseInteraction);

Parameters: Same as in create new object command. Only include the parameters that needs the change and use null for others.

Sort by field

```
myDataMatrix.sort(field, order);
```

Parameters

- field - field name or id.
- order - asc(ending) or desc(ending).

Zooms in or out on the specified cell

```
myDataMatrix.zoomCell(cellRef);
```

Parameters

- cellRef - target zoom cell. If left blank, selected cell will be used.

Customize object

Maximum number of characters per field

```
myDataMatrix.setMaxCharNumber(integer);
```

Developer defined functions

- .onCellHover (cellReference, mouseEvent) - called when pointer hovers over the cell
- .onCellLeave (cellReference, mouseEvent) - called when pointer leaves the cell
- .onCellClick (cellReference, mouseEvent) - called when cell is clicked on
- .onCellDblClick (cellReference, mouseEvent) - called when cell is double clicked
- .onPopulateCustom (queryResult) - called when data is received. queryResult is an array of record objects.

Parameters:

- cellReference - pointer to the cell, here are useful cell properties:
  - .recordId - record id of current cell, null if empty
  - .label - label of the current cell if labels are used
  - .imageURL - link to the displayed image

```
myDataMatrix.onCellHover = function (cellRef) {

   // show cell record number
   alert ('Record number is ' + cellRef.recordId);

   // show cell label
   alert ('and label is ' + cellRef.label);

}
```

Custom query

```
myDataMatrix.fetchData(field name array, field value array, custom option name
array, custom option value array);
```

Examples:

Query field 'Song_Year' for '2006'

```
myDataMatrix.fetchData(['Song_Year'], ['2006']);
```

Query field 'Song_Year' for '2006' and 'Song_Genre' for 'Rock'

```
myDataMatrix.fetchData(['Song_Year', 'Song_Genre'], ['2006', 'Rock']);
```

Query field 'Song_Year' for '2006' and send 'myCustomPar' with value 'myCustomValue' to 4D

```
myDataMatrix.fetchData(['Song_Year'], ['2006'], ['myCustomPar'],
['myCustomValue']);
```

Toolbar preferences

Customizes image browser toolbar.

myDataMatrix.customize(showToolbar, showAuxToolbar, useEditor, createBtn, deleteBtn, zoomBtn, setupBtn, displayBtn, layoutBtn, searchBtn, headerDropdown, ImageDropdown, sortDropdown);

Parameters (true for visible, false for invisible) :

- Basic
    - showToolbar - display or hide toolbar
    - showAuxToolbar - display or hide setup toolbar
    - useEditor - enable or disable editor
- Toolbar buttons
    - createBtn - create new record
    - deleteBtn - delete record
    - zoomBtn - magnify image/record
    - setupBtn - show/hide setup toolbar
    - displayBtn - display more or less records
    - searchBtn - search box
- Setup toolbar buttons
    - layoutBtn - select image position
    - headerDropdown - choose header field
    - imageDropdown - choose image field
    - sortDropdown - choose sorting field

Styles

Following CSS classes are added for user customization. Use these with caution: properties such as background-color or background-image are harmless and will spice up your application, but margin and position are almost guaranteed to break the object. Please be nice to the object.

Primary cell classes

CSS classes for image position within cell and selected state:

- Data Matrix layout (image is on left or right)
    - Not selected
        - .fourdaf_matrix_cell_left
        - .fourdaf_matrix_cell_right
    - Selected
        - .fourdaf_matrix_cell_left_selected
        - .fourdaf_matrix_cell_right_selected
- Image Matrix layout (image is on top or bottom)
    - Not selected

        - .fourdaf_matrix_cell_top
        - .fourdaf_matrix_cell_bottom
    - Selected
        - .fourdaf_matrix_cell_top_selected
        - .fourdaf_matrix_cell_bottom_selected

Cell element classes

These styles affect specific cell element. Primary cell class must be added as a prefix for these styles. Check below for examples.

- Both layouts (any image position)
    - .fourdaf_matrix_header
- Data matrix layout
    - fourdaf_matrix_fieldcolumn
    - fourdaf_matrix_valuecolumn
- Image matrix layout
    - .fourdaf_matrix_caption

Examples

```
<style type="text/css">

/*
// make image caption label bold
// only for image matrix mode (image is placed on top or bottom)
*/

.fourdaf_matrix_cell_top .fourdaf_matrix_caption,
.fourdaf_matrix_cell_bottom .fourdaf_matrix_caption

        {
                font-weight: bold;
        }

/*
// when cell is selected, keep bold text and change background color
// only for image matrix mode (image is placed on top or bottom)
*/

.fourdaf_matrix_cell_top_selected .fourdaf_matrix_caption,
.fourdaf_matrix_cell_bottom_selected .fourdaf_matrix_caption

        {
                font-weight: bold;
                background-color: #FFCC00;
        }

</style>
```

placeRecordData

The command populates a designated area on the webpage with the contents from the record. Integrated with 4DAF objects.

Syntax

```
placeRecordData (targetNode, cellReference, table, record id,
onPopulateFunction);
```

Parameters:

- Common
    - targetNode - reference to the element on the web page that holds other elements where record data is placed.
    - onPopulateFunction - function called when data arrives. Passed parameters are table and record id.
- Object integration
    - cellReference - reference to the specific element from the 4DAF view. Data Filler will extract all other relevant info from that object. (see examples on this page)
- Manual settings (use only if object integration isn't used)
    - Table - table name or id
    - Record id

Supported HTML elements

- Image data type
    - <IMG>
- Number and text data types
    - <INPUT>
    - <DIV>
    - <SPAN>
    - <TEXTAREA>
    - <TD>

Notes

If field name includes spaces (ex: [myTable]Field with spaces), field must be embedded with "." instead of " " (ex: class="4daf_[myTable]Field.with.spaces").

Examples

Examples consist of javascript portion that is slightly different for each object, and html code that is same for all objects.

This example assumes that data is requested from the table named Table with at least following fields: [Table]Field1 and [Table]ImageField.

Javascript

For calendar view

```
myCalendarView.onRecordClick = function (cellRef) {

       placeRecordData($('recordDetail'), cellRef);

}
```

For data tree

```
// on data tree row click, populate 'recordDetail' area with clicked
record

myDataTree.grid.onrowselect = function (rowRef) {

       placeRecordData($('recordDetail'), rowRef);

}
```

For data window

```
// on data window row click, populate 'recordDetail' area with clicked
record

myDataWindow.grid.grid.onrowselect = function (rowRef) {

       placeRecordData($('recordDetail'), rowRef);

}
```

For data matrix

```
// on data matrix cell click, populate 'recordDetail' area with clicked
record

myDataMatrix.onCellClick = function (cellRef) {

       placeRecordData($('recordDetail'), cellRef);

}
```

HTML

HTML code

```
<div id="recordDetail">
   Field1: <input class="4daf_[Table]Field1" />
   Image Field: <img class="4daf_[TableName]ImageField" />
</div>
```

Or

```
<div id="recordDetail">
   Field1: <span class="4daf_[Table]Field1"></span>
   Image Field: <img class="4daf_[TableName]ImageField" />
```

```
</div>
```

Save record

```
saveRecordData(targetNode, selection name, onSaveFunction)
```

Parameters:

- targetNode - reference to container HTML area that contains all fields.
- selection name - name of the selection.
- onSaveFunction - triggered when record is saved.

Example:

```
<div id="myArea">
  FirstName: <input type="text" class="4daf_[myTable]FirstName" />
  <input type="button" value="Save Record"
onclick="saveRecordData($('myArea'), 'myTable');" />
</div>
```

Supported elements:

- <INPUT type="text" />
- <INPUT type="checkbox" />
- <INPUT type="hidden" />

<TEXTAREA>

Data Tree

Data tree consolidates the data on preset break fields and presents in a hierarchical list. Data tree doesn't support predefined or named queries.

Calls and Properties

Create new data tree object.

Syntax:

```
myDataTree = new HGrid (parentObject, tableId, selectSingleRow);
```

Parameters:

- parent - html element in which data tree will be drawn, or null to display data tree in a window. Common way to refer to the HTML element is via unique id. For example, use $('myDiv') for DIV element with id "myDiv". Take a look at examples below to see more examples of this.
- table - table name or id that will be displayed as calendar view
- selectSingleRow - optional. If set to true, clicking on row will deselect all other rows. Default behaviour is to allow multiple row selection.

Returns:

- reference to the data tree object

Example:

```
// create a data tree in a new window for table with id 3
myDataTree = new HGrid (null, '3');
// create a data tree in embedded section called mySection
myDataTree = new HGrid ($('mySection'), '3');
```

.showBreakCount - record count in break label

Show or hide record count in the break level label.

Syntax:

```
myDataTree.showBreakCount (boolean);
```

.customize

Customizes data tree.

Syntax:

```
myDataTree.customize(onrowselect, onrowdblclick, onbreaklvlselect, show
toolbar);
```

Parameters (boolean form):

- onrowselect - set to true to enable custom action on grid row click and overwrites internal selection code. This is set with myDataTree.grid.onrowselect() function.
- onrowdblclick - set to true to enable custom action on grid row double click and overwrites editor popup. This is set with myDataTree.grid.onrowdblclick() function.

- onbreaklvlselect - set to true to enable custom action on data tree header click. This is set with myDataTree.onBreakLevelHeaderClick() function.
- show toolbar - show or hide toolbar buttons.

Example:

```
// enable custom actions for grid, but not for break level click
// also hide toolbar buttons
myDataTree.customize (true, true, false, false);
myDataTree.grid.onrowselect = function () { //user function };
myDataTree.grid.onrowdblclick = function () { //user function };
```

Advanced

.setQuickScroll - fast scrolling for large data

By default, data tree will shift data up or down when user is scrolling. However, when there is a large number of records present data shift slows down the data tree tremendously. Use this function to enable fast scrolling and prevent data shift slowdown. When quick scrolling is enabled, grid will clear itself when scrolling starts and repopulate when scrolling ends.

Quick scroll is automatically activated when large number of records is present.

Use only if there is an issue with data shifting rows when accessing small amount of data.

Syntax:

```
myDataTree.setQuickScroll (boolean);
```

Events

Data Tree supports 2 types of events: onrowselect and onBreakLevelHeaderClick. These events can be captured in onAfterInit function:

```
onAfterInit = function(){
// Actions and events that occurs after the connection is established
};
```

To capture the onrowselect:

```
onAfterInit = function(){
       myDataTree.grid.onrowselect = function(rowreference){
              // Do something
       };
};
```

To capture the onBreakLevelHeaderClick:

```
onAfterInit = function(){
       myDataTree.onBreakLevelHeaderClick = function(header reference,
value, break level){
              // Do something
       };
};
```

Getting the selected row reference

The data tree consists of the Heirarchical list and data grid object. The data in the row are part of the Grid object. Getting the value of the selected row is a bit different than the technique used in the Data Window. When a row is clicked, the selected row object reference can be obtain by calling rowreference.getElements(). For example:

```
onAfterInit = function(){
        myDataTree.grid.onrowselect = function(rowreference){
                var selectedrowref = rowreference.getElements();
        };
};
```

To get a more precise cell reference, the column index must be specified at the element level. For example:

```
onAfterInit = function(){
        myDataTree.grid.onrowselect = function(rowreference){
                var selectedrowref = rowreference.getElements()[1];
        };
};
```

Getting the current value in the selected row

Similar to the Data Window, the row and column id in data window starts from row #0 and column #0 to row #n-1 and column #n-1. However, in the data tree, the first column is reserved for all parent list. Therefore the start column for all rows starts at #1. Here is a simple data tree example:

	0		1
0	Parent		
1		Child	
2			GrandChild1
3			GrandChild2
4			GrandChild3

Suppose the user click on the row that contains the value of "GrandChid2". The value can be obtain by

1. Establish the reference to the selected row

```
        rowreference.getElements();
```

2. Establish the selected column in the row

```
        rowreference.getElements()[1];
```

3. Obtain the content from the target cell

```
        rowreference.getElements()[1].getContent();
```

In the case where you want to object the value when the user clicked on a row, you can try the following approach.

```
onAfterInit = function(){
        myDataTree.grid.onrowselect = function(rowreference){
                var myValue = rowreference.getElements()[1].getContent();
        };
};
```

Set the highlight on the selected row

By default, the row gets highlighted each time the user clicked on it and it will get unhighlighted when the user clicked on the same row for the 2nd time. However, this behavior can be overwritten with the method setFocusAll and setForcus. For example:

```
onAfterInit = function(){
       myDataTree.grid.grid.onrowselect = function(rowreference){
              myDataTree.grid.setFocusAll(false)   // Deselect all rows
in the data window
              rowreference.setFocus(true);               // Select the
row that the user clicked
       };
};
```

Known issues

Predefined query support

Datatree will automatically load first predefined query if all records display is disabled in the Control Panel. Data tree currently doesn't support multiple predefined queries nor predefined queries dynamically created based on data.

Calendar Picker

Calendar picker is an extension to the input field that will automatically pop up a calendar picker when input field is focused.

Calendar picker is written primarily to expand the inline editing for the data grid, but is also usable as a standalone object.

API

Basic

Attach calendar picker

```
dax_attachCalendarPicker(inputNode);
```

Note: as of now, .dateFormat must be set before picker is opened for the first time.

Set formatting

```
myInputObject.dateFormat = dateFormat;
```

* dateFormat - "dMM-DD-YYYY" , "dDD-MM-YYYY", "dDD-MMM-YYYY"

Events

On focus

Calendar picker opens.

```
myInputObject.onCalendarFocus = function() { };
```

On click

Calendar cell is clicked.

```
myInputObject.onCalendarClick = function() { };
```

On blur

Calendar picker closes, either by clicking on cell or by losing focus.

```
myInputObject.onCalendarBlur = function() { };
```

Example

```
$('myInputObject').dateFormat = 'dMM-DD-YYYY';
dax_attachCalendarPicker($('myInputObject'));
```

Calendar View

Calendar view displays records from a single table in a calendar grid. Calendar will use one field from a record to display a value, and another to determine in which date to place the value. Additional end date field could be used to display date range. Calendar view support record editing via editor object, and can be displayed in either html element or window object.

Each cell will show up to 10 records.

Calls and Properties

Basic

```
new CalendarView - Create object
```

Creates new calendar view object.

Syntax:

```
myCalView = new calendarView (parent, selection, dateField, dateEndField,
displayField, auxDisplayField, queryField, queryValue);
```

Parameters:

- parent - DOM element in which calendar view will be drawn, or null to display calendar view in window. If DOM element is used, div with fixed width and height is highly recommended. To eliminate white gap on the right side of the calendar, DOM element width should be divisible by 7 (for example: 602px, or 700px). Common way to refer to the HTML element is via unique id. For example, use $('myDiv') for DIV element with id "myDiv". Take a look at examples below to see more examples of this.
- selection - selection name or id that will be displayed as calendar view
- dateField - date field that will determine in which date will the record be set, or null to pick date automatically
- dateEndField - date field as an end field if date range is used (for multiple day events), null to avoid date range
- displayField - field that will be displayed in calendar date cell
- auxDisplayField - optional second field that will be displayed in calendar date cell

Query Parameters (optional):

- queryField - field name or id to query on
- queryValue - value to query on

Returns:

- myCalView - reference to calendar view object

Examples:

```
// create a calendar view in a div with id 'myCal' using selection named
'Tasks'
// with automatic date and display fields
myCalView = new calendarView ($('myCal'), 'Tasks', null, null, null,
null);

// create a calendar view in a window with a selection named 'Meetings'
and date range
// from field '[Meetings]Start Date' to field '[Meetings]End Date' with
automatic display field
myCalView = new calendarView (null, 'Meetings', '[Meetings]Start Date',
'[Meetings]End Date', null, null);
```

.refresh - Refresh calendar view

Refresh function will refresh calendar view display and reload the data from the backend.

Syntax:

```
calendarView.refresh();
```

.customize - Hide or show calendar components

Hides or display calendar view components: display bar and search bar.

Syntax:

```
myCalendarView.customize(create and delete buttons, display bar, search bar);
```

Parameters (boolean: true for visible, false for hidden):

- Toolbar buttons - create and delete records.
- Display bar - choose date, display, and range fields.
- Search bar - live search bar.

.setTitle - Window title

If calendar is displayed inside a window, use this function to update default window title.

Syntax:

```
calendarView.setTitle('Window title');
```

.setWeekStart - set week start day

This function sets the start day of the week to Monday ('Mon') or Sunday ('Sun'). Default value is 'Sun'.

Syntax and example:

```
// set week start day to Monday
calendarView.setWeekStart('Mon');
```

.fetchData

Starts new calendar data query.

Syntax:

```
calendarView.fetchData(query field name, query value, erase query when done,
update search box);
```

For new query:

- Query field name - field name or id to query.
- Query value - value to query. 4D wildcard '@' is supported.
- Erase query when done - query parameters will be erased after querying if this is set to 'true'.
- Update search box - update search field and value when fetch is called.

Customize object

Developer defined functions

- .onRecordClick (recordReference) - called when cell is clicked on
- .onRecordDblClick (recordReference) - called when cell is double clicked

Parameters:

- cellReference - pointer to the record object, here are useful cell properties:
  - .recordId - record id of current cell, null if empty
  - .value - value of the cell

Advanced

.setBlankValue - text to display if display field is blank

This function sets the value that is displayed if display field has no data. Default value is '-'.

Syntax:

```
calendarView.setBlankValue(text);
```

Example:

```
calendarView.setBlankValue('Blank field');
calendarView.refresh();
```

.selectMultiple - allow multiple record selection

This function allows selection of multiple records in a calendar view. Default value is false.

Syntax and example:

```
// enable multiple record selection
calendarView.selectMultiple(true);
```

Examples

European style calendar in window

This example function will create a calendar inside a window. Window title is set to "Daily Planner" and week will start on Monday.

```
function calInWindow() {

   // create new calendar from selection named 'Events' in a window
   calWin= new calendarView(null, 'Events', null, null,null);

   // change window title and week start date
   calWin.setTitle('Daily Planner');
   calWin.setWeekStart('Mon');
}
```

How does it work?

Calendar view will get all records that belong to the current month and display a selected field in the calendar. Date range is supported as well.

Dashboard

The Dashboard provides a quick snapshot of data. It is constructed using the following objects, making the API of those objects available to the developer:

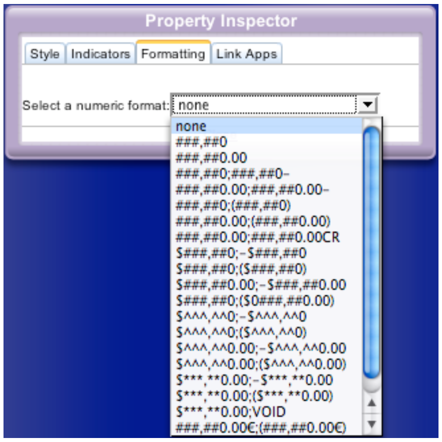
Object Name	API Access
Grid	myDashboard.commandName
Report Viewer	myDashboard.commandName
Toolbar	myDashboard.toolbar.commandName
Viewport	myDashboard.viewport.commandName
Window	myDashboard.window.commandName

DAX_Dev_GetRequestedReportName

Some Developer Hooks currently do not have parameters for retrieving the Dashboard name. (Dax_DevHook_OnQuery is an example, and it can be a very powerful Developer Hook for Dashboards). Thus, *Dax_Dev_GetRequestedReportName* can be inserted for such a situation.

```
Returned:
$0          TEXT    Name of the requested report.  If "" string, the Web
                    request is not to run a report.
```

Dashboard Formats



Select a numeric format: none

Here is a screenshot of existing formats for Dashboard fields. Use Developer Hook *Dax_DevHook_RPT_Formats,* and methods *DAX_Dev_RPT_FormatsGet* and *DAX_Dev_RPT_FormatsSet* to modify it.

New Field Formats In Version 11.2

Field formats for Dashboard have been extended to 10 more formats. These formats are added to support EURO currency symbol.

```
"###,##0€;-###,##0€"
"###,##0€;(###,##0€)"
"###,##0.00€;-###,##0.00€"
"###,##0€;($0###,##0.00€)"
"^^^,^^0€;-^^^,^^0€"
"^^^,^^0€;(^^^,^^0€)"
"^^^,^^0.00€;-^^^,^^0.00€"
"^^^,^^0.00€;(^^^,^^0.00€)"
"***,**0.00€;-***,**0.00€"
"***,**0.00€;(***,**0.00€)"
"***,**0.00€;VOID"
```

DAX_DevHook_RPT_Formats

This method is executed once at the startup of the database (during 4D Ajax Framework initialization). Thus be sure the restart the database to see any changes.

1. Retrieve the list of existing formats by executing the method *DAX_Dev_RPT_FormatsGet*.

2. Based on the returned formats (returned in the form of TEXT Array), you can now append a new format to the array, update an existing format in the array or delete the array element.

3. Save the current (modified) formats by executing the method *DAX_Dev_RPT_FormatsSet.*

DAX_Dev_RPT_FormatsGet

This method retrieves the list of current formats used in report into a Text array. It should only be used only in Developer Hook *DAX_DevHook_RPT_Formats*.

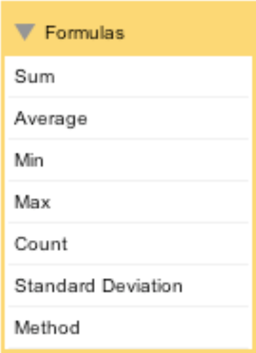
```
Returned:
  $0   -      -
   Passed:
   $1        TEXT      Name of field type
   $2        POINTER   Pointer to a TEXT array

   Returned:
   $0          BOOLEAN   True if the requested formats for a given field type
                         exists
```

DAX_Dev_RPT_FormatsSet

This method update the current format list (available for Dashboard or iPhone report) for a specific field type. It should only be used in *DAX_DevHook_RPT_Formats*.

```
   Passed:
   $1        TEXT      Name of field type
   $2        POINTER   Pointer to a TEXT array

   Returned:
   $0          BOOLEAN   True if the updated formats are set properly
```

DAX_DevHook_RPT_Formula



The Dashboard Editor allows you to add formulas to your reports providing calculations on many levels. All of the formulas are predefined and cannot be edited.

However, a custom formula on a specific group level can be used through the method *DAX_DevHook_RPT_Formula*. This method gets executed when the "Method" formula is set to a detail footer of a report. This method allows the developers to calculate and define the footer value based on the current selection, table, report and field.

```
Passed:
$1        TEXT     Report name
$2        TEXT     Current formula assigned to the column footer
$3        POINTER  Pointer to the current table (Null if DCS)
$4        POINTER  Pointer to the current field (column)

Returned:
$0          TEXT      Result or value that will be inserted to
                      the column footer
```

DAX_Dev_RPT_SetCSSClass

There is another method that should be used in Developer Hook *DAX_DevHook_RPT_Formula.*

It allows the developer to set the name of the CCS class that will be used in each cell value.

```
Passed:
$1        TEXT   Name of the CSS class
```

Front End Commands

Create new Dashboard

```
var myDashboard = new dax_dashboardViewer(reportName, location);
```

- location - null to create Dashboard as a window, or node reference to embed the Dashboard.
- reportName - name of the report.

Toolbar

Show or hide

```
myDashboard.showToolbar(showToolbar);
```

- showToolbar - boolean

Viewport

Show or hide scrollbars

```
myDashboard.viewport.showVerScrollbar(boolean);

myDashboard.viewport.showHorScrollbar(boolean);
```

Refresh

Set refresh interval

```
myDashboard.setRefreshInterval(refreshInterval);
```

- refreshInterval- refresh interval in seconds, must be more than 10 seconds

Update dashboard

```
myDashboard.refresh();
```

Send custom values to 4D

To add a custom value:

```
myDashboard.addCustomValue(name, value);
```

To clear custom values:

```
myDashboard.clearCustomValues();
```

Use GET WEB FORM VARIABLES 4D command to retrieve custom commands.

Since initial dashboard call will not include custom values, make sure that .refresh is called to update dashboard with custom values sent. Example:

```
var myDashboard = new dashboardViewer('myDashboard',
$('myDashboardDiv'));
myDashboard.addCustomValue('valueName', 'myCustomValue');
myDashboard.refresh();
```

Sleep

Pause all periodic updates (data refresh, unique query refresh).

```
myDashboard.sleep();
```

This is useful when object is in background or hidden, and resources shouldn't be used to update the object. Use .wake() to resume updates.

Wake

Resume all periodic updates (data refresh, unique query refresh) paused with .sleep().

```
myDashboard.wake();
```

Destroy

Removes the object completely.

```
myDashboard.destroy();
```

Example

Create a Dashboard and then use Grid command to increase row size.

```
// create a Dashboard
var myDashboard1 = new dashboardViewer($('reportGoesHere'), 'Songs1');

// increase row height to 30 pixels
myDashboard1.setRowHeightInPx(30);
```

Charts

Previously, charts for Dashboards were rendered using Apple’s popular Canvas element. Now PNG, SVG, and Image URL support joins the fray for 4D Dashboards. As an added bonus, iPhone users can switch between multiple charts by tapping on the screen.

Charts for Dashboards now support several formats:

	Works with following browsers				Support multiple charts on iPhone
	Firefox 3	**Safari 4**	**IE 7 IE 8**	**iPhone**	
Canvas	Yes	Yes	Yes	Yes	No
PNG	Yes	Yes	Yes	Yes	Yes
Image URL	Yes	Yes	Yes	Yes	Yes
SVG	Yes	Yes	No	No	N/A
Interactive Charts	Yes	Yes	Yes	No	No

Conveniently, there is no update to the JavaScript Chart API. The new formats are automatically supported.

Only new Developer methods need to be called in *Dax_DevHook_DefineChart* to pass these newly supported charts to the front end. These new Developer Hooks are:

- DAX_Dev_SetDashboardSVG
- DAX_Dev_SetDashboardPNG
- DAX_Dev_SetDashboardImageURL

To avoid cut-off or distorted charts, make sure that area where chart is embedded is similar to the actual chart width and height.

iPhone chart area is restricted to 400x200px in landscape orientation or 240x200px in portrait orientation. iPhone will automatically change orientation when rotated.

Refresh chart / dashboard

Use this command to programmatically refresh Charts and Dashboards:

```
myChart.refresh();
```

Send/Receive custom values From 4D

Developers can send and receive custom variables from 4D in a name/value pair.

To add a custom value:

```
myChart.addCustomValue(name, value);
```

To clear custom values:

```
myChart.clearCustomValues();
```

Variable sent to 4D are retrieved using the GET WEB FORM VARIABLES 4D command.

Since initial chart/dashboard call will not include custom values, make sure that *.refresh* is called to refresh the chart/dashboard with custom values sent.

Example:

```
var myChart = new chartViewer('myChart', $('myChartDiv'));
myChart.addCustomValue('valueName', 'myCustomValue');
myChart.refresh();
```

CUSTOM VARIABLES

In the case that Developers want to get custom values from 4D, use ***DAX_Dev_SetCustomVariables*** to send the information. This method can be called in one of the following Developer Hook methods:

1. ***DAX_DevHook_OnQuery*** (Non-DCS only)
2. ***DAX_DevHook_QueryFilter*** (Non-DCS only)
3. ***DAX_DevHook_DCS_SetSelection*** (DCS only)

The method ***DAX_Dev_SetCustomVariables*** takes 2 parameters:

```
Passed:
$1        Pointer to a Text Array containing variable names
$2        Pointer to a Text Array containing variable values
```

Example:

```
  ARRAY TEXT($varNames_at;3)
  ARRAY TEXT($varValues_at;3)
  $varNames_at{1}:="v1"
  $varNames_at{2}:="v2"
  $varNames_at{3}:="v3"
  $varValues_at{1}:="aaa"
  $varValues_at{2}:="bbb"
  $varValues_at{3}:="ccc"
  DAX_Dev_SetCustomVariables (->$varNames_at;->$varValues_at)
```

Important:

The size of the 2 arrays must be the same. Otherwise, the size of the value array will be adjusted to the size of the name array.

If the method is called more than once, the one the last execution will be used.

Custom variables support is to be used in conjunction with the new front-end API command *getCustomValuesFrom4D*(). For more information about this Javascript API command, please visit:

DEVELOPER METHODS

DAX_Dev_Query_GetSelectionName

This method is to be executed in *DAX_DevHook_OnQuery*, *DAX_DevHook_QueryFilter* or *DAX_DevHook_DCS_SetSelection*. It returns the name of the selection that is being queried.

```
Passed:
None

Return:
$0        TEXT   Selection Name
```

DAX_Dev_SetDashboardSVG

This method inserts one or more SVG file for a specific dashboard report. All SVG chart will be append to the end of the dashboard. This method is to be executed in *DAX_DevHook_DefineChart* only.

```
Passed:
$1        Pointer to a Text, String or Blob variable

Return:
None
```

Example:

This example will add 2 SVG charts to a dashboard named "Spending" based off the "Dept Spending" selection:

```
C_BLOB($svgblob1;$svgblob2)
 $ReportName_t:=$1 ` *** Name of the report
 $SelectionName_t:=$2 ` *** Name of the Selection (Table/View/DCS)
 $ReportOwner_t:=$3 ` *** Owner of the report ("public" or specific
username)
 Case of
   : ($ReportName_t="Spending") & ($SelectionName_t="Dept Spending") &
($ReportOwner_t="public")
     DOCUMENT TO BLOB ("graph1.svg";$svgblob1)
     DOCUMENT TO BLOB ("graph2.svg";$svgblob2)
     DAX_Dev_SetDashboardSVG (->$svgblob1;->$svgblob2)
 End case
```

DAX_Dev_SetDashboardPNG

This method inserts one or more pictures for a specific dashboard report. All pictures will be converted to a PNG format and appended to the end of the dashboard. This method is to be executed in *DAX_DevHook_DefineChart* only.

```
Passed:
$1        Pointer to a Text, String or Blob variable

Return:
None
```

Example:

This example will add 1 PNG image to the Dashboard named "Spending" based on the "Dept Spending" selection:

```
C_PICTURE($picVar)
 $ReportName_t:=$1 ` *** Name of the report
 $SelectionName_t:=$2 ` *** Name of the Selection (Table/View/DCS)
 $ReportOwner_t:=$3 ` *** Owner of the report ("public" or specific
username)
 Case of
  : ($ReportName_t="Spending") & ($SelectionName_t="Dept Spending") &
($ReportOwner_t="public")
     $chartArea:=CT New offscreen area
```

```
   GRAPH($chartArea;1;aCategory;aSeries;aValues)
   $picVar:=CT Area to picture ($chartArea;-1)
   CT DELETE OFFSCREEN AREA ($chartArea)
   DAX_Dev_SetDashboardPNG (->$picVar)
 End case
```

DAX_Dev_SetDashboardImageURL

This method allows the developer to append one or more external URL to a dashboard report. This method is to be executed in *DAX_DevHook_DefineChart* only.

```
Passed:
$1        Pointer to a Text, String or Blob variable

Return:
None
```

Example:

This example will add 1 Google Chart image URL to the Dashboard.

```
C_TEXT ($url)
ARRAY TEXT ($urlArray;2)
$urlArray
{1}:="http://chart.apis.google.com/chart?cht=p3&chd=t:90,49&chs=350x150&c
hl=Foo|Bar"
$urlArray
{2}:="http://chart.apis.google.com/chart?cht=p3&chd=t:50,49&chs=350x150&c
hl=Foo|Bar"
DAX_Dev_SetDashboardImageURL (->$urlArray)
```

DAX_Dev_SetDashboardChart

This method allows the developer to build arrays of charts values and x and y label values.

Parameter	**Type**	**Description**	**Optional**
$1	Array	Array of x-axis label values	
$2	Array	Array of y-axis label values	
$3	Array	Array of values for the chart	
$4	Numeric	Minimum Value for y-label	x
$5	Numeric	Maximum Value for y-label	x

Example:

This example will build a chart, manually defining array values:

```
` Project method: DAX_DevHook_DefineChart
$ReportName_t:=$1
$SelectionName_t:=$2
$ReportOwner_t:=$3
Case of
: ($ReportName_t="Spending") & ($SelectionName_t="Dept Spending") &
($ReportOwner_t="public")
C_REAL($minYvalue_r;$maxYvalue_r)
```

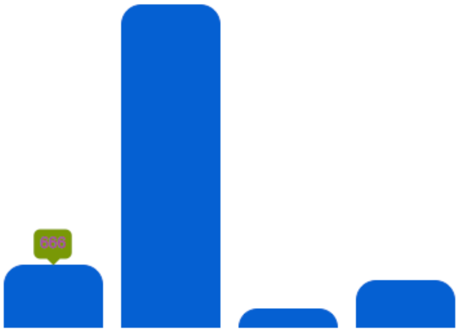
```
ARRAY TEXT(xlabel_at;4)  `*** Label on X-Axis (Must use
process_array_variable - TEXT)
ARRAY TEXT(ylabel_at;5)  `*** Label on Y-Axis (Must use
process_array_variable - TEXT and must have 5 elements)
ARRAY REAL(value_ar;10)  `*** Trend values (Must use
process_array_variable - REAL)

xlabel_at{1}:="1pm"
xlabel_at{2}:="2pm"
xlabel_at{3}:="3pm"
xlabel_at{4}:="4pm"

ylabel_at{1}:="20%"
ylabel_at{2}:="40%"
ylabel_at{3}:="60%"
ylabel_at{4}:="80%"
ylabel_at{5}:="100%"

value_ar{1}:=60
value_ar{2}:=30
value_ar{3}:=25
value_ar{4}:=23
value_ar{5}:=40
value_ar{6}:=57
value_ar{7}:=70
value_ar{8}:=49
value_ar{9}:=45
value_ar{10}:=90

` *** The following variables will be passed into the
DAX_Dev_SetDashboardChart method
` *** as the 4th and 5th parameters (Optional). If omitted,
DAX_Dev_SetDashboardChart
` *** will automatically calculate min and max value (based on the
ylabel_at) for you.
$minYvalue_r:=0  `*** It should correspond to the minimum value in
ylabel_at
$maxYvalue_r:=0  `*** It should correspond to the maximum value in
ylabel_at                    `

DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)

End case
```

DAX_Dev_SetDashboardChart and the Interactive Charts

The Interactive Charts introduced in 4D Ajax Framework v11 Release 5 (11.5) do not necessarily use all of the parameters passed in *Dax_Dev_SetDashboardChart* (shown above). It all depends on the chart type. The required parameters still must exist, but they do not necessarily need to be filled with array values.

Here is a look at each:

Interactive Bar Charts:

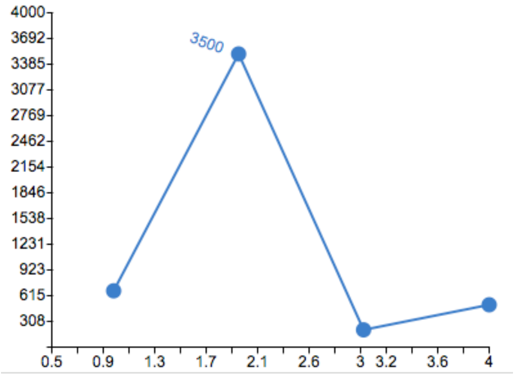


Actual values do not have to be passed into the x and y label arrays, as Interactive Bar Charts do not support labels. For Bar charts, the minimum value displayed is always zero.

Here is an example of creating a Bar Chart:

```
: ($ReportName_t="4DAF_BarChart")
  C_REAL($minYvalue_r;$maxYvalue_r)
  ARRAY TEXT(xlabel_at;8)   `*** Label on X-Axis (Must use
process_array_variable - TEXT)
  ARRAY TEXT(ylabel_at;8)   `*** Label on Y-Axis (Must use
process_array_variable - TEXT)
  ARRAY REAL(value_ar;20)   `*** Trend values (Must use
process_array_variable - REAL)
  ARRAY STRING(40;$label_at;0)
  ARRAY REAL($value_ar;0)

  ALL RECORDS([Department])
  ORDER BY([Department];[Department]Name;>)
  For ($i;1;Records in selection([Department]))
    If ($i<=8)
        RELATE MANY([Department]ID)
        APPEND TO ARRAY($value_ar;Records in selection([Doctor]))
    End if
    NEXT RECORD([Department])
  End for

  COPY ARRAY($value_ar;value_ar)

  DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)
```

Line Charts:

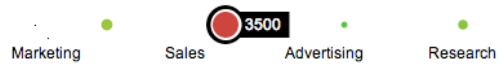


Actual values do not have to be passed into the x and y label arrays, as Interactive Line Charts generate the x and y label values automatically based on the array of values passed in the value array.

Here is an example of creating a Line Chart:

```
: ($ReportName_t="4DAF_LineChart")
  C_REAL($minYvalue_r;$maxYvalue_r)
  ARRAY TEXT(xlabel_at;8)  `*** Label on X-Axis (Must use
process_array_variable - TEXT)
  ARRAY TEXT(ylabel_at;8)  `*** Label on Y-Axis (Must use
process_array_variable - TEXT)
  ARRAY REAL(value_ar;20)  `*** Trend values (Must use
process_array_variable - REAL)
  ARRAY STRING(40;$label_at;0)
  ARRAY REAL($value_ar;0)

  ALL RECORDS([Department])
  ORDER BY([Department];[Department]Name;>)
  For ($i;1;Records in selection([Department]))
    If ($i<=8)
        RELATE MANY([Department]ID)
        APPEND TO ARRAY($value_ar;Records in selection([Doctor]))
    End if
    NEXT RECORD([Department])
  End for

  COPY ARRAY($value_ar;value_ar)

  DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)
```

Dot Charts:

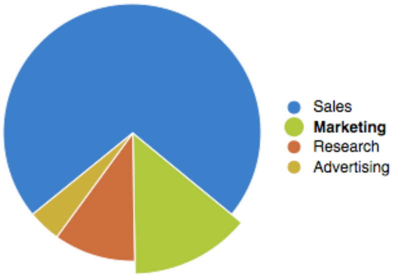


Interactive Dot Charts do support labels, but it is probably best to manually truncate the data. Dot Chart labels are horizontally oriented and they all reside on one line, which is why truncating the data is recommended.

Here is an example of creating a Dot Chart:

```
: ($ReportName_t="4DAF_DotChart")
  C_REAL($minYvalue_r;$maxYvalue_r)
  ARRAY TEXT(xlabel_at;8)  `*** Label on X-Axis (Must use
process_array_variable - TEXT)
  ARRAY TEXT(ylabel_at;8)  `*** Label on Y-Axis (Must use
process_array_variable - TEXT)
  ARRAY REAL(value_ar;20)  `*** Trend values (Must use
process_array_variable - REAL)
  ARRAY STRING(40;$label_at;0)
  ARRAY REAL($value_ar;0)

  xlabel_at{1}:="A & C"
  xlabel_at{2}:="Cardiology"
  xlabel_at{3}:="Pharma"
  xlabel_at{4}:="Internal Med"
  xlabel_at{5}:="Hematology"
  xlabel_at{6}:="Immunology"
  xlabel_at{7}:="Infectious"
  xlabel_at{8}:="Crit Care"

  ALL RECORDS([Department])
  ORDER BY([Department];[Department]Name;>)
  For ($i;1;Records in selection([Department]))
    If ($i<=8)
        RELATE MANY([Department]ID)
        APPEND TO ARRAY($value_ar;Records in selection([Doctor]))
    End if     NEXT RECORD([Department])
  End for

  COPY ARRAY($value_ar;value_ar)

  DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)
```

Pie Charts:



Interactive Pie Charts do support labels.

Here is an example of creating a Pie Chart:

```
: ($ReportName_t="4DAF_PieChart")
  C_REAL($minYvalue_r;$maxYvalue_r)
  ARRAY TEXT(xlabel_at;8)  `*** Label on X-Axis (Must use
process_array_variable - TEXT)
  ARRAY TEXT(ylabel_at;8)  `*** Label on Y-Axis (Must use
process_array_variable - TEXT)
  ARRAY REAL(value_ar;20)  `*** Trend values (Must use
process_array_variable - REAL)
  ARRAY STRING(40;$label_at;0)
  ARRAY REAL($value_ar;0)

  ALL RECORDS([Department])
  ORDER BY([Department];[Department]Name;>)
  For ($i;1;Records in selection([Department]))
    If ($i<=8)
        APPEND TO ARRAY($label_at;[Department]Name)
        RELATE MANY([Department]ID)
        APPEND TO ARRAY($value_ar;Records in selection([Doctor]))
    End if
    NEXT RECORD([Department])
  End for

  COPY ARRAY($label_at;xlabel_at)
  COPY ARRAY($value_ar;value_ar)

  DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)
```

DAX_DevHook_DefineChart

This Developer Hook method allows 4D Developers to define one or more charts that will be included with a dashboard report. The chart type that can be set in this method are Canvas, SVG, PNG (Chart as a picture), Image URL (e.g. Google Chart), and the Interactive Charts.

```
Passed:
$1        TEXT    Report Name
$2        TEXT    Selection name
$3        TEXT    Owner of the report name

Return:
```

```
None
```

With the given parameter, the developers can use them to determine the name of the dashboard (report), the selection name and the ownership of the dashboard. Then a specific developer methods must be called to insert a specific report into the dashboard.

For example:

```
  ` Project method: DAX_DevHook_DefineChart
  $ReportName_t:=$1
  $SelectionName_t:=$2
  $ReportOwner_t:=$3
  Case of
    : ($ReportName_t="Spending") & ($SelectionName_t="Dept Spending") &
($ReportOwner_t="public")
      C_REAL($minYvalue_r;$maxYvalue_r)
      ARRAY TEXT(xlabel_at;4)  `*** Label on X-Axis (Must use
process_array_variable - TEXT)
      ARRAY TEXT(ylabel_at;5)  `*** Label on Y-Axis (Must use
process_array_variable - TEXT and must have 5 elements)
      ARRAY REAL(value_ar;10)  `*** Trend values (Must use
process_array_variable - REAL)

      xlabel_at{1}:="1pm"
      xlabel_at{2}:="2pm"
      xlabel_at{3}:="3pm"
      xlabel_at{4}:="4pm"

      ylabel_at{1}:="20%"
      ylabel_at{2}:="40%"
      ylabel_at{3}:="60%"
      ylabel_at{4}:="80%"
      ylabel_at{5}:="100%"

      value_ar{1}:=60
      value_ar{2}:=30
      value_ar{3}:=25
      value_ar{4}:=23
      value_ar{5}:=40
      value_ar{6}:=57
      value_ar{7}:=70
      value_ar{8}:=49
      value_ar{9}:=45
      value_ar{10}:=90

      ` *** The following variables will be passed into the
DAX_Dev_SetDashboardChart method
      ` *** as the 4th and 5th parameters (Optional). If omitted,
DAX_Dev_SetDashboardChart
      ` *** will automatically calculate min and max value (based on the
ylabel_at) for you.
      $minYvalue_r:=0  `*** It should correspond to the minimum value in
ylabel_at
      $maxYvalue_r:=0  `*** It should correspond to the maximum value in
ylabel_at
                      `
      DAX_Dev_SetDashboardChart (->xlabel_at;->ylabel_at;-
>value_ar;$minYvalue_r;$maxYvalue_r)
    End case
```

JavaScript API for Interactive Charts

What follows is the JavaScript API for embedding an Interactive Chart into your own custom page. Only these two lines of code are required for successfully embedding a Chart.

```
var myChart = new dax_chartViewer('chartName', $('chartDiv'), {
type   : dax_chartViewer.TYPE_BAR,
});
```

type:

Bar Chart:	`dax_chartViewer.TYPE_BAR`
Line Chart:	`dax_chartViewer.TYPE_LINE`
Dot Chart:	`dax_chartViewer.TYPE_DOT`
Pie Chart:	`dax_chartViewer.TYPE_PIE`

Customization

There are many optional commands for further customization. Some commands are unique to one particular Chart type while others are universal for all types.

Bar Charts:

Bar Charts offer the most options for customization allowing changing *edges, orientation, color, hoverBackgroundColor,* and *hoverTextColor.*

edges: For hard or soft edges.
orientation: For horizontal or vertical Bar Charts.
color: Specify the chart color.
hoverBackgroundColor: When the mouse hovers over a Bar chart, specify the background color for the value.
hoverTextColor: When the mouse hovers over a Bar chart, specify the color for the value.

Possible values:

```
edges                          :     BAR_EDGE_HARD, BAR_EDGE_SOFT
orientation                    :     BAR_ORIENTATION_VERTICAL,
                                     BAR_ORIENTATION_HORIZONTAL
color                          : a hexadecimal value as a string
hoverBackgroundColor           : a hexadecimal value as a string
hoverTextColor                 : a hexadecimal value as a string
```

Here is a full example of a horizontal Bar Chart with rounded edges:

```
var myChart = new dax_chartViewer('chartName', $('chartDiv'), {
type   : dax_chartViewer.TYPE_BAR,

/* optional */
edges         : dax_chartViewer.BAR_EDGE_SOFT,
orientation   : dax_chartViewer.BAR_ORIENTATION_HORIZONTAL,
/* optional   */
color                             : '#94302a',
hoverBackgroundColor              : '#502b93',
hoverTextColor                    : '#30a9d2'
});
```

Dot Charts:

Dot charts allow for color customization, such as *color, hoverBackgroundColor,* and *hoverTextColor.*

color: Specify the chart color.
hoverBackgroundColor: When the mouse hovers over a Bar chart, specify the background color for the value.
hoverTextColor: When the mouse hovers over a Bar chart, specify the color for the value.

Possible values:

```
color                         : a hexadecimal value as a string
hoverBackgroundColor          : a hexadecimal value as a string
hoverTextColor                : a hexadecimal value as a string
```

Here is a full example of a Dot Chart:

```
var myChart = new dax_chartViewer('chartName', $('chartDiv'), {
type   : dax_chartViewer.TYPE_DOT,

/* optional         */
hoverBackgroundColor      : '#7890ab',
hoverTextColor            : '#cdef01',
labelColor                : '#abcdef'
});
```

Line Charts:

Line charts allow for color customization, such as *color, hoverBackgroundColor, hoverTextColor, and labelColor.*

color: Specify the chart color.
hoverBackgroundColor: When the mouse hovers over a Bar chart, specify the background color for the value.
hoverTextColor: When the mouse hovers over a Bar chart, specify the color for the value.
labelColor: The color of the labels.

Possible values:

```
color                         : a hexadecimal value as a string
hoverBackgroundColor          : a hexadecimal value as a string
hoverTextColor                : a hexadecimal value as a string
labelColor                    : a hexadecimal value as a string
```

Here is a full example of a Line Chart:

```
var myChart = new dax_chartViewer('chartName', $('chartDiv'), {
type   : dax_chartViewer.TYPE_LINE,

/* optional   */
color                       : '#abcdef',
hoverBackgroundColor        : '#ccffcc',
hoverTextColor              : '#cdef01',
labelColor                  : '#babac0'
});
```

Pie Charts:

Line charts allow for *labelColor* customization.
labelColor: The color of the labels.

Possible values:

```
labelColor               : a hexadecimal value as a string
```

Here is a full example of a Pie Chart:

```
var myChart = new dax_chartViewer('chartName', $('chartDiv'), {
type   : dax_chartViewer.TYPE_PIE,

/* optional   */
labelColor                         : '#67890a'
});
```

Admin Calls

Commands

Get Group List

Command /DAX/AdminGetGroupList

Gets the list of groups and returns it to the Admin front-end

Sample call:

http://localhost.8000/DAX/AdminGetGroupList?sessionId=S9112006173404CCWB

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB

Reply

Gets the list of groups and returns it to the Admin front-end

Sample reply:

```
<GroupList>
  <Group ID="15001" Name="Admins"/>
  <Group ID="15002" Name="Group A"/>
</GroupList>
```

Set Table Preferences

Command /DAX/AdminSetTablePreferences

Save the Table Preferences set by the admin web front-end

Sample call:

```
http://localhost:8000/DAX/AdminSetTablePreferences?sessionId=S9112006173404CCWB&groupid=0&tableid=1&alias=ZIPCodes&visible=False&calendarview=False&datatreeview=False
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
groupid	Numeric	0, 1, 2,..., N
tableid	Numeric	0, 1, 2,..., N
alias	TEXT	MyTable
visible	BOOLEAN	True/False
calendarview	BOOLEAN	True/False
datatreeview	BOOLEAN	True/False

Reply

If the preferences are successfully set, the success status is sent to the frontend.

Sample reply:

```
<AdminSetTablePreferences>
  <Status Success="True"/>
</AdminSetTablePreferences>
```

Get Table Preferences

Command /DAX/AdminGetTablePreferences

Reply will return all group names and ids.

Sample call:

```
http://localhost:8000/DAX/AdminGetTablePreferences?sessionId=S9112006173404CCWB&groupid=15001
```

Parameters

Name	Type	Example Value

sessionId	TEXT	S9112006173404CCWB
groupid	Numeric	0, 1, 2,..., N

Reply

Return the preference of all tables to the frontend.

Sample reply:

```
<tables>
  <table alias="ZIPCodes" calendarfieldid="" calendarview="False"
datatreeview="False" id="1" name="ZIPCodes" visible="True"/>
  <table alias="ConfSpecs" calendarfieldid="" calendarview="False"
datatreeview="False" id="2" name="ConfSpecs" visible="True"/>
  <table alias="TestTable" calendarfieldid="" calendarview="False"
datatreeview="False" id="3" name="TestTable" visible="True"/>
</tables>
```

Set Field Preferences

Command /DAX/AdminSetFieldPreferences

Save the Field Preferences set by the admin web front-end

Sample call:

```
http://localhost:8000/DAX/AdminSetFieldPreferences?sessionId=S9112006173404CCWB
&groupid=0&tableid=1&fieldid=2&alias=ZIPCodes&inlist=true&indetail=true&format=
””
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
groupid	Numeric	0, 1, 2,..., N
tableid	Numeric	0, 1, 2,..., N
fieldid	Numeric	0, 1, 2,..., N
alias	TEXT	My Table
inlist	BOOLEAN	True/False
indetail	BOOLEAN	True/False
format	TEXT	YYYY-MM-DD

Reply

If the preferences is successfully set, the success status is sent to the frontend.

Sample reply:

```
<AdminSetFieldPreferences>
  <Status Success="True"/>
```

```
</AdminSetFieldPreferences>
```

Get Field Preferences

Command /DAX/AdminGetFieldPreferences

Returns the properties of all fields in the target table.

Properties: alias, breaklevel, format, id, indetail, indexed, inlist, invisible, mandatory, name, nonEnterable, nonModifiable, type, unique.

Sample call:

```
http://localhost:8000/DAX/AdminGetFieldPreferences?sessionId=S9112006173404CCWB
&groupid=15001&tableid=1
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
groupid	Numeric	0, 1, 2,..., N
Tableid	Numeric	0, 1, 2,..., N

Reply

Sample reply:

```
<fields tableid="1">
  <field alias="ZIPCode" breaklevel="0" format="" id="[1][1]"
indetail="True" indexed="True" inlist="True" invisible="False"
mandatory="False" name="ZIPCode" nonEnterable="False"
nonModifiable="False" type="alpha" unique="False" choicelist=""/>
  <field alias="State" breaklevel="0" format="" id="[1][2]"
indetail="True" indexed="True" inlist="True" invisible="False"
mandatory="False" name="State" nonEnterable="False" nonModifiable="False"
type="alpha" unique="False" choicelist=""/>
  <field alias="City" breaklevel="0" format="" id="[1][3]"
indetail="True" indexed="False" inlist="True" invisible="False"
mandatory="False" name="City" nonEnterable="False" nonModifiable="False"
type="text" unique="False" choicelist=""/>
</fields>
```

Set General Preferences

Command /DAX/AdminSetPreferences

This command allows the Administrator to set any general preferences.

Sample call:

Suppose you want to set the following preference values:

TimeOut=2 TaskbarLocation=Top Style=xPress

```
http://localhost:8000/DAX/AdminSetPreferences?sessionId=S9112006173404CCWB&Time
Out=2&TaskbarLocation=Top&Style=xPress
```

Parameters

Name	**Type**	**Example Value**
Value1	TEXT	0, 1, 2,..., N / a-z;A-Z
Value2	TEXT	0, 1, 2,..., N / a-z;A-Z
…	TEXT	0, 1, 2,..., N / a-z;A-Z
Value[N]	TEXT	0, 1, 2,..., N / a-z;A-Z

Reply

Sample reply:

```
<AdminSetPreferences>
  <Status Success="True"/>
</AdminSetPreferences>
```

Get General Preferences

Command /DAX/AdminGetPreferences

This command allows the Administrator to retrieve any general preferences.

Sample call:

Suppose your general preferences contain the following values:

TimeOut=2 TaskbarLocation=Top Style=xPress

```
http://localhost:8000/DAX/AdminGetPreferences?sessionId=S9112006173404CCWB
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB

Reply

Sample reply:

```
<Preferences>
  <TimeOut>2</TimeOut>
  <TaskbarLocation>Top</TaskbarLocation>
  <Style>xPress</Style>
</Preferences>
```

Create a View or Virtual Table

Command /DAX/AdminCreateVirtualTable

This command creates a View (or Virtual Table) based on the real relation that is already established in the database structure. The command requires 2 parameters: primarytableid and lastmanytableid. The assumption is that the primary table is the One Table and last-many-table is one of the related many table (at any levels).

For example:

Here we have a relation of [Customers] <- [Invoices] <- [LineItems]. [Customers] is table #1, [Invoices] is table #2 and [LineItems] is table #3. To create a view that includes data from [Customers] and [Invoices], the primarytableid (variable) must be set to 1 and lastmanytableid to 2.

If the command is called with the primarytableid=1 and lastmanytableid=3, the created view will include data from all 3 tables.

Sample call:

```
http://localhost:8000/DAX/AdminCreateVirtualTable?sessionId=S9112006173404CCWB&
primarytableid=1&lastmanytableid=3
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
primarytableid	Numeric	0, 1, 2,..., N
lastmanytableid	Numeric	0, 1, 2,..., N

Reply

If a new view is successfully created, the name of the view and id will be returned to the frontend. Please note that table ID of of views are return as a negative number and the default of the view is "View_"+ABS(ID).

Sample reply:

```
<Tables>
  <Table ID="-1" name="View_1" />
</Tables>
```

Delete a View or Virtual Table

Command /DAX/AdminDeleteVirtualTable

This command deletes a View (or Virtual Table) based on a given View-Table-ID.

Sample call:

```
http://localhost:8000/DAX/AdminDeleteVirtualTable?sessionId=S9112006173404CCWB&
tableid=-1
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	0, 1, 2,..., N

Reply

If the target view is successfully deleted, the success status will be sent to the frontend.

Sample reply:

```
<AdminDeleteVirtualTable>
  <Status Success="True"/>
</AdminDeleteVirtualTable>
```

Set Break Point

Command /DAX/AdminSetBreakPoint

This command allows the administrator to set a break point (or break level) on a selection for a datatree style.

Sample call:

```
http://localhost:8000/DAX/AdminSetBreakPoint?sessionId=S9112006173404CCWB&tableid=2&fieldid=[2][3]&breaklevel=1
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	0, 1, 2,..., N
fieldid	[Numeric][Numeric]	0, 1, 2,..., N
breaklevel	Numeric	0, 1, 2,..., N

Reply

If the break level is successfully set, the success status will be sent to the frontend.

Sample reply:

```
<AdminSetBreakPoint>
  <Status Success="True"/>
</AdminSetBreakPoint>
```

Get Related Table

Command /DAX/AdminGetRelatedTable

This command allows the administrator to retrieve the list of all related-many-table (all levels) from the backend. The related-many-table list depends on the primary table id.

Sample call:

```
http://localhost:8000/DAX/AdminGetRelatedTable?sessionId=S9112006173404CCWB&tableid=1
```

Parameters

Name	**Type**	**Example Value**
sessionId	TEXT	S9112006173404CCWB

tableid	Numeric	0, 1, 2,..., N

Reply

Sample reply:

```
<tables>
  <table alias="Invoices" id="2" name="Invoices"/>
  <table alias="Line Items" id="3" name="LineItems"/>
</tables>
```

Get Table Position

Command /DAX/AdminGetTablePosition

This command allows the administrator to retrieve the current position a table. Generally won't be needed as the table list is returned in position order.

Sample call:

http://localhost:8000/DAX/AdminGetTablePosition?sessionId=S9112006173404CCWB&tableid=1

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	0, 1, 2,..., N

Reply

Sample reply:

```
<AdminGetTablePosition position="1" tableid="1">
  <Status Success="True"/>
</AdminGetTablePosition>
```

Set Table Position

Command /DAX/AdminSetTablePosition

This command allows the administrator to set the position for a table.

Sample call:

http://localhost:8000/DAX/AdminSetTablePosition?sessionId=S9112006173404CCWB&tableid=1&position=8

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB

tableid	Numeric	0, 1, 2,..., N
Position	Numeric	0, 1, 2,..., N

Reply

Sample reply:

```
<AdminSetTablePosition position="8" tableid="1">
  <Status Success="True"/>
</AdminSetTablePosition>
```

Get Field Position

Command /DAX/AdminGetFieldPosition

This command allows the administrator to retrieve the current position of a field. Generally won't be needed as the field list is returned in position order.

Sample call:

```
http://localhost:8000/DAX/AdminGetFieldPosition?sessionId=S9112006173404CCWB&tableid=1&fieldid=[1][1]
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	0, 1, 2,..., N
fieldid	[Numeric][Numeric]	0, 1, 2,..., N

Reply

Sample reply:

```
<AdminGetFieldPosition position="1" tableid="1" fieldid="[1][1]">
  <Status Success="True"/>
</AdminGetFieldPosition>
```

Set Field Position

Command /DAX/AdminSetFieldPosition

This command allows the administrator to set the position for a field.

Sample call:

```
http://localhost:8000/DAX/AdminSetFieldPosition?sessionId=S9112006173404CCWB&tableid=1&fieldid=[1][1]&position=8
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	0, 1, 2,..., N
fieldid	[Numeric][Numeric]	0, 1, 2,..., N
position	Numeric	0, 1, 2,..., N

Reply

Sample reply:

```
<AdminSetFieldPosition position="8" fieldid="[1][1]" tableid="1">
  <Status Success="True"/>
</AdminSetFieldPosition>
```

Views (Virtual Tables)

Overview

A view is a virtual structure that can be made up of one or more tables in one single display. The structure of the view is based on the relations in the database (One-to-Many, no circular). Here is a simple structure that you can create a view from:

[Customers] <- [Invoices] <- [LineItems] -> [Products]

The idea is that the view is based on 2 key information: One-Table (Primary) and Last-Related-Many-Table (Cut-Off-Many-Table).

1. One-Table is the parent table where each record may have multiple related records. In this case, the One-Table can be any table.

2. Last-Related-Many-Table is the sub-level table. It can go as deep as 8 level from the One-Table. If we use the example from the above to create a view, the Last-Related-Many-Table can

be: [Invoices] or [LineItems] if [Customers] is selected as the One-Table [LineItems] if [Products] is selected as the One-Table.

Structure Requirements

In order for 4D Ajax Framework to establish the relation properly, all fields that have one or more relations must have the Index property turned on.

Recommendation

In order to create a view, the first information that you need is whether the One-Table has at least one One-to-Many relation. This information can be retrieved by calling the command GetRelatedTable. Once you have selected the Last-Related-Many-Table, you will then make a call to AdminCreateVirtualTable with the id of the One-Table and the id of the selected Last-Related-Many-Table.

Error Handling

Overview

Most of the error handling happen in the 4D Ajax Framework back-end. When an error occurs during the execution, the error handler will generate a report back to the front-end in the XML format.

The structure of the error-xml is generic to all errors. Therefore there may be some additional attributes (information) that are not required for particular error. Depending on the error, you may check only a certain attributes when the error is returned to the front-end.

Standard Response

```
<error>
  <errorCode fieldid="" hint="" message=""
tableid="">errorcode</errorCode>
</error>
```

Attributes

Name	Type	Example Value
fieldid	Numeric	0, 1, 2,..., N
tableid	Numeric	0, 1, 2,..., N
hint	TEXT	InvalidUsernameOrPassword (Error code in text)
message	TEXT	The user name or password supplied is incorrect... (Error text in detail)

Example

```
<error>
  <errorCode fieldid="0" hint="InvalidUsernameOrPassword" message="The
user name or password supplied is incorrect. Please login again to
continue." tableid="0">InvalidUsernameOrPassword</errorCode>
</error>
```

Special Cases

Duplicate Key

In an event where the Duplicate Key (or -9998) occurs in the backend, the following error-xml will be returned to the frontend. The XML file will include all instance of DuplicateKey error in the XML file.

```
<errors>
  <errordetail code="DuplicatedKey" value="[T1][F1]"/>
  <errordetail code="DuplicatedKey" value="[T1][F1]"/>
  ...
  <errordetail code="DuplicatedKey" value="[Tn][Fn]"/>
</errors>
<pre>

[T] - table id
[F] - field id
```

Themes

4D Ajax Framework themes can be edited and created by those who have access to the server.

Themes are based upon CSS (Cascading Style Sheet) files and numerous PNG (Portable Network Graphics) files that are laid out in HTML by the 4D Ajax Framework.

Modifying Themes

Existing themes can be modified by editing the CSS file (e.g. osx.css) within each theme's folder. These folders are contained inside the "Themes" directory in 4D Ajax Framework.

CSS files are typically edited with a text editor, such as WordPad (Windows) or BBEdit (Mac OS). It is not recommended to use WYSIWYG editors such as Macromedia Dreamweaver or Microsoft FrontPage to edit these files, as they may insert errant code that can break the interface.

Once a CSS file has been opened with a text editor, elements such as fonts, type styles, text sizes, and background colors may be changed. It is not recommended to change dimensions of the elements.

Graphics can be edited with a bitmap graphic editor, such as Adobe Photoshop, that is capable of saving files in PNG format. If you wish to edit graphical elements, open them from their respective folders (i.e. "taskbar" or "window") to make modifications.

If you choose to make modifications to any of the themes, it is recommended you create a backup of the affected theme folders.

Creating New Themes

New themes can be created for 4D Ajax Framework using a combination of CSS and graphics editing. As recommended above, a text editor and bitmap graphic application should be used for theme creation.

For your convenience, there is a theme called "Template" provided in 4D Ajax Framework's "Themes" folder.

To start work on your new theme, duplicate the Template folder and rename it. For example, you can rename the new folder MyTheme and rename the template.css file within to mytheme.css.

Open mytheme.css in a text editor to start making your changes. The most easily customizable parts are denoted with comments in the CSS code, as follows:

```
 /* CUSTOMIZABLE */
```

This comment is followed by a description of what the ensuing CSS code controls.

For example, you will encounter this block of CSS code:

```
 /* CUSTOMIZABLE */
 /* The following style determines the font properties of the Taskbar
button. */

 .button_text {
 font-size: 12px;
 font-family: Arial, Helvetica, sans-serif;
 color: #000000;
 margin-top: 7px;
 margin-left: 8px;
 float: left;
 }
```

If you would like to enlarge the text, make it red, and give it a different font, you may change the CSS code as follows:

```
 .button_text {
 font-size: 14px;
 font-family: Times New Roman, serif;
 color: #FF0000;
 margin-top: 7px;
 margin-left: 8px;
 float: left;
 }
```

Debugger

Debugger is a simple object that appends messages in the debugger window. Debugger requires initialized bridge (which happends automatically on login) and working window object.

Calls and Properties

showDebugger - Show debugger

Opens debugger window.

Syntax and example:

```
showDebugger();
```

debugAlert - Send message

Adds a message to the debug window if open.

Syntax and example:

```
debugAlert('myVariable is' + myVariable);
```

How does it work?

debugAlert function checks if debugger window exists, and if it does message is appended. If debugger is not initialized or window is not accessible, debugAlert silently ignores the passed message.

debugAlert and showDebugger are shortcut functions, as debugger exist as part of the bridge.

Error Handling

4D based

Critical errors

Critical errors will refresh the page.

Following errors are considered critical:

1. Invalid Access Point
2. Invalid Session Id - session id is expired or invalid.

Other errors

Other errors will be silently displayed in the console, which can be opened via button on the sidebar if user is logged in as administrator, or via showDebugger command. Check Console entry for more information on console.

Browser based

Browser based errors are triggered by browser, which are most often Javascript errors.

Trapping errors

Developer can trap for errors in fourdaf_dev_errorTrap function located in callbacks.js function.

Parameters received:

1. type - error type, can be 4D or browser.
2. hint - error hint sent by 4D.
3. message - error message sent by 4D.

```
fourdaf_dev_errorTrap (type, hint, message) {

}
```

Third party integration

Firebug and Firebug Lite

If Firebug plug-in for Firefox is installed, errors will be displayed in the Firebug console as well. This also includes Firebug Lite which is supported in Internet Explorer 7 and Safari 2.

Safari console

If Safari debug menu is enabled, errors will be displayed in Safari Javascript console.

Formatting

The formatting function is used to display the data in a specific way where the information can be displyed in a friendly format.

Calls and Properties

To use the formatting function just 2 parameters needs to be passed.

The formatting will be configured on the resources.js

Eg.

```
format( type, value )
```

type: is the formatt selected on the admin section

value: is the data value that will be formatted

Types of Formatting

There are different types of formatting options as showing below:

[a] Date
[b] Number
[c] Text/Alpha
[d] Boolean
[e] Numeric

Date

The DATE formatting allows the user to display the data in the following formats:

The general formatting for date is yyyd-mm-dd.

- none no formatting
- mm-dd-yyyy: 11-23-2006
- dd-mm-yyyy: 24-12-2006
- dd-mmm-yyyy: 13 Oct, 2006

The date fomatting is defined on the resource.js file. If no formatting is selected "NONE" a default formatting will be selected by getting the first formatting on the right after the NONE formatting on the resources.js file. If there is no ohter formatting option after the NONE on the resources.js file, for TIME and DATE fields, a 4D formatting will be de default one.

Eg1. ["NONE" , "MM-DD-YYYY"] If none is selected on the admin,the formatting MM-DD-YYYY will be the default formatting for this field.

Eg2. ["NONE"] If none is selected the default formatting will be YYYY-MM-DD as 4D formatting.

Number

The NUMBER formatting is used for (integer, long integer, real) fields.

For numbers (integer, real, longinteger) the formatting will be defined by the user. The type of formatting will be declared on the resources.js file, following the parameters showed below.

The variable is composed of 5 parameters:

1] preFix: any value to be placed before the data. Eg: $, R$, #, etc)
2] thousand separator: defines if the number will be separated in thousands. Any value is accept .

3] decimal places: defines if the number will have a decimal place at the end. Any value is accept.
4] number of decimal places. (indicaes how many digits after the decimal separator. 2, 3, 4, etc .
5] postFix: any value to be placed after the data. EUR, C, F, etc.

For example to a number to be displayed as: $999,999.99, the parameter will have the following formatting:

"$,cm,per,2,"

[$]	**prefix**
[cm]	comma separator for thousands
[per]	period for decimal points
[2]	number of decimal points
[blank]	postfix

All the parameters must be passed, otherwise the function won't work.

Text / Alpha

The Text / Alpha formatting is used to display the data in lowercase or in UPPERCASE formatting.

Formatting available:

- lowercase: text in lowercase
- UPPERCASE: TEXT IN UPPERCASE

For these cases the formatting will be displayed just on the GRID, the formatting won't take place on the editor.

Boolean

The boolean formatting is really useful when the information is 0 or 1 and it can be displayed in TRUE or FALSE formatting or any another specified formatting.

Formatting available:

- True/False
- Vrai/Faux
- Yes/No
- Male/Female

For boolean formattings, the new formatting can be chosen by typing the first letter of the options.

Eg: If the user types "T" or "t" on the True/False formatting the system will formatt to True.

Localization

Localization Front End

4D Ajax Framework uses a localization technique that allows developers to deploy their applications in most languages. Localized strings are located in the localization folder within a js folder.

Every language file and STR object inside will have a two letter language code appended to it, like in these examples:

- resources_en.js, with STR_EN for English
- resources_fr.js, with STR_FR for French

Adding a Language

Create a template

Open a template.js file located in js/localization folder and copy it as a resources_LC.js file, where LC will correspond to the language code. Customize the strings inside the file between /* BEGIN LOCALIZATION */ and /* END LOCALIZATION */ and rename STR into STR_LC, where LC is (again) the language code.

For example, finalized Spanish localization would be saved as resources_es.js and STR is renamed into STR_ES.

Add a template to 4D Ajax Framework

To load the new language file with 4D Ajax Framework, append the file name to the js/framework.js file. Near the beginning of the file is language localization:

```
// add or remove languages here
languages[0] = '/resources_en.js'; // English
languages[1] = '/resources_fr.js'; // French
```

In the above example, to add Spanish localization file, append the following line:

```
// add or remove languages here
languages[2] = '/resources_es.js'; // Spanish
```

Localization Back End

4D Ajax Framework uses a localization technique that allows developers to deploy their applications in most languages. All messages that are displayed in the application interface and on the web front end are retrieved from XML files located in the \Extras\Support folder. By default these messages appear in English.

Setting the Default Language

The messages and errors in the backend database is determined by the language set in DAX_DevHook_Preferences. The method that is 4D Ajax Framework use to set the language is called DAX_Dev_SetPreference. To have 4D Ajax Framework use a different language, set the first paramater in DAX_Dev_SetPreference to 1 and the 2nd parameter to either “fr”, “de”, “es” or “ja” in the method named DAX_DevHook_Preferences and restart the application.

For example:

```
 Change   DAX_Dev_SetPreference (1;"en")   to   DAX_Dev_SetPreference
(1;"fr")
```

Managing the Language Files

There are 2 files that are used in the localization technique, Localized_strings.xml and English_error.xml. All of the messages that appear in the 4D interface and on the web site front end are in these files.

Localized_strings.xml

This file contains the progress, alert, and confirm messages that are displayed in the 4D interface. Any time 4D Ajax Framework needs to display a message, 4D Ajax Framework will find an appropriate translation based on the value that was set in DAX_DevHook_Preferences through DAX_Dev_SetPreference.

For example: When the initialization routine runs at application startup, 4D Ajax Framework will locate "DAXInitializing" in the Localized_strings.xml file and load the message from a language node for the progress message. If language is set to "en" the message will appear as "DAX Initializing" and if language is set to "fr" the message will appear as "DAX Initialization."

Here is an example of the content of Localized_strings.xml

```
<Strings>

   <DAXInitializing>
      <en>DAX Initializing</en>
      <fr>DAX Initialisation</fr>
   </DAXInitializing>

   <DAXShuttingdown>
      <en>DAX Shutting down</en>
      <fr>DAX Fermeture en cours</fr>
   </DAXShuttingdown>

   <DAXInitializationError>
      <en>DAX Initialization Error. DAX will not be available.
Error:</en>
      <fr>DAX Erreur d'initialisation. Dax ne peut pas se lancer.
Erreur:</fr>
   </DAXInitializationError>

   ...

</Strings>
```

The developer can add new language elements to the XML file and then specify that language as the language to be used by changing the 2nd parameter for DAX_Dev_SetPreference(1;”en”) in DAX_DevHook_Preferences.

For example: If you want "DAXInitializing" to appear in Spanish simply add a <es> element as a child of the <DAXInitializing> element with the message in Spanish as element value.

```
<Strings>

   <DAXInitializing>
      <en>DAX Initializing</en>
      <fr>DAX Initialisation</fr>
      <es>DAX Inicialización</es>
   </DAXInitializing>

   ...

</Strings>
```

You will then change the 2nd parameter for DAX_Dev_SetPreference(1;”en”) in the method DAX_DevHook_Preferences to "es."

When adding a new language you must add a new translation element for every message in the Localized_strings.xml file and you must add a new error XML file for the language.

en_errors.xml

This file contains all error messages that are sent to the front end. By default, the name of the file matches the language value set in DAX_DevHook_Preferences.

Here is an example of the en_errors.xml file's contents:

```
<errors>

   <CallBackEventCannotBeModified>
      <message>The selected Callback event cannot be modified</message>
   </CallBackEventCannotBeModified>

   <CommandNotRecognized>
      <message>The requested action is not available. Please inform the
system administrator.</message>
   </CommandNotRecognized>

   <DuplicatedKey>
      <message>A unique value is required for this field.</message>
   </DuplicatedKey>

   ...

</errors>
```

You can create an error file for another language by duplicating the en_errors.xml file and changing the contents of the <message> nodes.

To create an error file for a different language:

- Duplicate the en_errors.xml file.
- Rename the duplicate file to begin with the language that you set in DAX_DevHook_Preferences (fr_errors.xml for example).
- Go through the new file and translate the contents of all of the <message> nodes to the language you are adding.

Set a particular language to a group

To set a language to a user group, you must call a method named **DAX_Dev_InstallLanguage** in **DAX_DevHook_Preferences** with 2 input parameters.

- Name of a user group that you want to set the language to

- 2-character-country-code (en, fr, de, ..., etc.)

For example:

DAX_Dev_InstallLanguage ("Group A";"en")
DAX_Dev_InstallLanguage ("Group B";"fr")

Offline Mode

Imagine connecting to your 4D Ajax Framework (4DAF) powered webpage in your office and then deciding to go to the factory to gather more data. There's a good chance you may lose your laptop's online connection between the office and factory. How can you gather and edit new information if communication with your database is lost?

The solution is Offline Mode, a brand new feature implemented in the 4DAF Release 2 v11.2. Set your web application to Offline Mode and you can now travel anywhere to save your records locally and then upload them to 4D at a later time.

Requirements

Web browsers that are HTML 5 compatible or have Google Gears installed support Offline Mode. Safari 3 and Firefox 3 support HTML 5. Users of Internet Explorer 7 and Firefox 2+ must have Google Gears installed to use Offline Mode.

> As of Release 2 v11.2 of the 4DAF, HTML 5 is the latest revision of the core language of the World Wide Web. Offline storage is a new native feature of HTML 5, which is why web browsers that are compatible with HTML 5 automatically support Offline Mode. For more information on HTML 5, see http://en.wikipedia.org/wiki/HTML_5.
>
> At this time Internet Explorer is not compatible with HTML 5. Thus, IE 7 users must install Google Gears, which is an open source implementation of HTML 5. Google Gears can be downloaded at http://gears.google.com/.

Currently, only the 4DAF's Data Grid object supports Offline Mode. Offline records can be added, modified, and erased via both Inline Editing and the sheet window editor.

Currently, picture fields are not supported in Offline Mode.

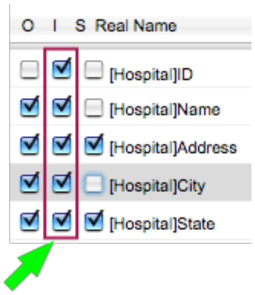
Offline Mode is not a Full Client

A common misconception that arises with the mention of Offline Mode is that it must equate to a full client in the web browser. This is not true. It does not allow for storage and caching of existing data offline. It can only add and modify records entered while offline.

Limitation

A noteworthy limitation of Offline Mode is that offline tables will not work correctly unless all Input fields for that offline table are set to viewable. To set all fields for viewable:

1) Log in to the 4DAF Client as Administrator.
2) Open the Control Panel. Go to the Access Control tab.
3) Select the table you intend to set for Offline Mode.
4) Go to that table's fields. Under the 'I' column, check all boxes (as shown below).



O	I	S	Real Name
□	☑	□	[Hospital]ID
☑	☑	□	[Hospital]Name
☑	☑	☑	[Hospital]Address
☑	☑	□	[Hospital]City
☑	☑	☑	[Hospital]State

API Calls

For developers creating their own custom web pages using the 4D Ajax Framework, here are the API commands for incorporating Offline Mode to their applications.

Go offline

This command places the 4DAF into Offline Mode, where the 4DAF will stop making network calls and the data grid object will automatically show the "Offline Record" tab for tables set for Offline Mode. (To designate a table for Offline Mode see section "*Administrator: Enable Tables for Offline Mode*"). Choice List values are also preserved for tables marked for Offline Mode.

```
dax_goOffline();
```

Once the offline command is called, one of two events are called:

```
dax_onOfflineModeSuccess = function() { };
dax_onOfflineModeFail = function() { };
```

Go online

This command returns the 4DAF into Online Mode. All offline records will be uploaded, and when upload is completed a success or fail event handler will fire. Only records that are successfully uploaded are erased from the offline storage database.

```
dax_goOnline();
```

Once the online command is called, one of two events are called:

```
dax_onOnlineModeSuccess = function() { };
dax_onOnlineModeFail = function() { };
```

Purge offline cache

Purging the offline cache will erase all offline records and structure, allowing for the offline storage to be recreated with new a structure.

```
dax_purgeOfflineCache();
```

Offline Mode is designed to work with a stable table structure. If the table or field structure has changed, purge the offline cache to make sure that offline storage structure matches current structure.

Sample page

Here is an example for a custom page that displays an Offline and Online button. Alerts will display should the transitions to Offline or Online Mode succeed or fail.

```
<?xml version="1.0" encoding="utf-8" ?>
<!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">
```

```
<head>
<title>4DAF Offline Mode sample page</title>

<!-- link to 4DAF libraries with English localization -->
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/dev/callbacks.js"></script>
<script language="javascript" type="text/javascript" charset="ISO-8859-1"
src="dax/js/compile.js"></script>
<script language="javascript" type="text/javascript" charset="utf-8"
src="/dax/js/localization/resources_en.js"></script>

<!-- 4DAF CSS -->
<link charset="ISO-8859-1" href="dax/themes/leopard/leopard.css"
media="all" type="text/css" title="Leopard" />

<!-- initialization code that logs user in when web page is loaded -->
<script language="javascript">
      window.onload = function() {
          // perform login
         dax_login('myUsername', 'myPassword');
   }
</script>
<!-- online and Offline Mode event handlers -->
<script language="javascript">

function dax_onOfflineModeSuccess() {

// hide offline button and show online button
       $('offlineButton').style.display = 'none';
       $('onlineButton').style.display = 'block';
       alert('Offline Mode entered successfully.');
};

function dax_onOfflineModeFail() {
       alert('Offline Mode was not entered successfully, 4DAF is still in
Online Mode');
};

function dax_onOnlineModeSuccess() {

       // hide online button and show offline button
       $('onlineButton').style.display = 'none';
       $('offlineButton').style.display = 'block';

       alert('Online Mode entered successfully.');
};

function dax_onOnlineModeFail() {
       alert('Online Mode was not entered successfully, 4DAF is still in
Offline Mode.');
};

</script>

</head>
<body>

<!-- button that places 4DAF into Offline Mode -->
<input id="offlineButton" type="button" value="Go offline"
style="display: block;" onclick="dax_goOffline();" />
```

```
<!-- button that places 4DAF into Online Mode, hidden by default -->
<input id="onlineButton" type="button" value="Go online" style="display:
none;" onclick="dax_goOnline();" />

</body>
</html>
```

Query Calls

Get All Records

Command /DAX/AllRecords

Gets all records for the given table.

Sample call:

```
http://localhost:8000/DAX/AllRecords?sessionId=S9122006172949ZUDL&tableid=4&start=0&length=14
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N
start	Numeric	1, 2, 3,..., N
length	Numeric	1, 2, 3,..., N

Reply

Sample reply:

```
<queryResult queryid="C9122006175603JEIP" recordsinselection="4" size="4"
tableid="1" tablename="Customers">
  <row recordid="[1][0]" selectionid="1" locked="False"
parenttableid="0">
    <field fieldrecordid="0" id="[1][1]" mandatory="false">7</field>
    <field fieldrecordid="0" id="[1][2]" mandatory="false">105</field>
    <field fieldrecordid="0" id="[1][3]" mandatory="false">1</field>
  </row>
  <row recordid="[1][1]" selectionid="2" locked="False"
parenttableid="0">
    <field fieldrecordid="1" id="[1][1]" mandatory="false">8</field>
    <field fieldrecordid="1" id="[1][2]" mandatory="false">150</field>
    <field fieldrecordid="1" id="[1][3]" mandatory="false">2</field>
  </row>
  <row recordid="[1][2]" selectionid="3" locked="False"
parenttableid="0">
    <field fieldrecordid="2" id="[1][1]" mandatory="false">9</field>
    <field fieldrecordid="2" id="[1][2]" mandatory="false">643</field>
    <field fieldrecordid="2" id="[1][3]" mandatory="false">3</field>
  </row>
  <row recordid="[1][3]" selectionid="4" locked="False"
parenttableid="0">
    <field fieldrecordid="3" id="[1][1]" mandatory="false">10</field>
    <field fieldrecordid="3" id="[1][2]" mandatory="false">123</field>
    <field fieldrecordid="3" id="[1][3]" mandatory="false">1</field>
  </row>
</queryResult>
```

Query

Command /DAX/Query&sessionid=&tableid=&start=&length=

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N
start	Numeric	1, 2, 3,..., N (first record number that will be returned - selection numbers start with 1)
length	Numeric	1, 2, 3,..., N (number of records to return per call)

Sample call:

```
http://localhost:8000/DAX/query?tableid=18&start=1&length=1&sessionid=S90220061
03304YWPU
```

Reply

Reply will return new query id that should be used with subsequent calls.

Sample reply:

```
<queryResult queryid="C9022006114606KYRG" recordsinselection="13"
size="1" tableid="18" tablename="Inventory">
 <row recordid="[18][0]" selectionid="1" locked="False"
parenttableid="0">
  <field fieldrecordid="0" id="[18][1]" mandatory="false">Book</field>
  <field fieldrecordid="2" height="600" id="[18][2]" mandatory="false"
width="800">/DAX/GetImage/18.2.2?sessionid=S9182006120617CCWB&unq=S918200
6132033GVGF</field>
 </row>
</queryResult>
```

Get a Record

Command /DAX/GetRecord

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N
recordid	[Numeric][Numeric]	1, 2, 3,..., N ([Table ID][Record ID])

Sample call:

```
http://localhost:8000/DAX/GetRecord?sessionid=S9182006120617CCWB&tableid=1&reco
rdid=[1][1]
```

Reply

Reply will the requested record.

Sample reply:

```
<queryResult queryid="" recordsinselection="1" size="1" tableid="1"
tablename="ZIPCodes">
  <row recordid="[1][1]" selectionid="1" locked="True" parenttableid="0">
    <field fieldrecordid="1" id="[1][1]" mandatory="false">John</field>
    <field fieldrecordid="1" id="[1][2]" mandatory="false">Smith</field>
    <field fieldrecordid="1" id="[1][3]" mandatory="false">39</field>
    <field fieldrecordid="1" height="600" id="[1][4]" mandatory="false"
width="800">/DAX/GetImage/1.4.1?sessionid=S9182006120617CCWB&unq=S9182006
132033GVGF</field>
    <field fieldrecordid="1" id="[1][5]" mandatory="false"/>
  </row>
</queryResult>
```

Query Engine (BE)

The 4D Ajax Framework Query Engine is a generic way to query the database from the front-end. The Query Engine can be sent multiple sets of query parameters for tables, Developer Created Selections, and Views.

The 4D Ajax Framework Query Engine is designed to be called from the front-end only and would generally be called from the front-end's Query Engine object. The Query Engine returns an XML result that can be parsed or handed off to another object.

Implementation

The Query Engine is provided with the text name of a table and field to be queried as opposed to using table ids. Multiple fields can be queried at one time by passing multiple query lines. A query line consists of the field name, argument operator, value to search for, and an and/or flag.

The argument operator can be one of the following:

- equal (=)
- notequal (#)
- less (<)
- greater (>)
- lesseq (<=)
- greatereq (>=)

The and/or flag can be:

- and
- or

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
dax_qe_table	TEXT	1
dax_qe_field1..n	TEXT	[1][3]
dax_qe_argument1..n	TEXT	equal
dax_qe_searchvalue1..n	TEXT	jones
dax_qe_andor1..n	TEXT	and

For more information on table and field id, as well as how to obtain them, take a look at the Bridge 2.0 page.

When sending multiple query lines each of the line items should be suffixed with a unique number for that line (example: dax_qe_field1, dax_qe_argument1, dax_qe_searchvalue1, dax_qe_andor1 for line 1; dax_qe_field2, dax_qe_argument2, dax_qe_searchvalue2, dax_qe_andor2 for line 2, etc.)

Important

The and/or parameter effects the same line it is sent with. So to query for last name equals 'smith' or 'jones' you would create it with the first line containing an 'and' and the second line containing the 'or'.

Sample Call

```
http://127.0.0.1/DAX/QueryEngine?sessionID=S9122006172949ZUDL&dax_qe_table=1&dax_qe_field=[1][3]&dax_qe_argument=equal&dax_qe_searchvalue=jones&dax_qe_andor=and
```

Sample Reply

```
<queryResult queryid="C9122006175603JEIP" recordsinselection="2" size="2"
tableid="1" tablename="People">
  <row recordid="[1][0]" selectionid="1" locked="False">
    <field fieldrecordid="0" id="[1][1]" mandatory="false">Joan</field>
    <field fieldrecordid="0" id="[1][2]" mandatory="false">Jones</field>
  </row>
  <row recordid="[1][1]" selectionid="2" locked="False">
    <field fieldrecordid="1" id="[1][1]" mandatory="false">John</field>
    <field fieldrecordid="1" id="[1][2]" mandatory="false">Jones</field>
  </row>
</queryResult>
```

Advanced Use

You can also pass as a parameter either a Query ID or a Named Query. In this case first the selection for the ID/Name will be loaded and then after the Query Engine has created a selection the two selections will have an Intersection performed. This is basically a way to do a query in selection on a named query or existing query id.

Query Engine (FE)

The Query Engine object can be placed within any div. The object consists of a simple query editor that allows the user to query multiple fields at within a table at one time. The Query Engine (BE) result is returned as XML which you can then use to display the records to the user or for continued processing.

Call

To add the Query Engine you call QE_PageLoad and pass the element on the page where the Query Engine should be placed.

Syntax

```
QE_PageLoad(document.getElementById('IdOfDivOnPage'));
```

Example

```
<body onload="QE_PageLoad(document.getElementById('QueryEditor'));">
<div id="QueryEditor"></div>
</body>
```

Wildcard Search Support

You may be able to use a wildcard character of “@” as part of the search value. A wild card can be used in all positions of the search string. Here are some of the examples on how to use a wildcard to do a search in Field1.

1. Find the record where Field1 is equal to “Sa@”:

```
Field        -      Field1
Operator     -      Equals
Value        -      “Sa@”        (equivalent to starts-with “Sa”)
```

Example: API Call

```
myGrid.newQuery();
myGrid.addQuery('Field1, '=', 'Sa@');
myGrid.runQuery();
```

2. Find the record where Field1 is equal to “@er”:

```
Field        -      Field1
Operator     -      Equals
Value        -      “@er”        (equivalent to ends-with “er”)
```

Example: API Call

```
myGrid.newQuery();
myGrid.addQuery('Field1, '=', '@er');
myGrid.runQuery();
```

3. Find the record where Field1 is equal to “Sa@er”:

```
Field        -      Field1
Operator     -      Equals
Value        -       “Sa@er”(equivalent to starts-with “Sa” and ends-with
“er”)
```

Example: API Call

```
myGrid.newQuery();
myGrid.addQuery('Field1, '=', ' Sa@er ');
myGrid.runQuery();
```

Enable and Disable Wildcard Search Option

- By default, the wildcard search is on automatically. This means that the @ character will automatically be considered as a wildcard character. In a version 2004 database, you can disable the wildcard feature by enabling the “Consider @ as a character for Query and Order By” option in the Database Preferences. When a search or an order by is performed, the @ character is considered a literal character in the search string.

- In version 11, the developer can disable the wildcard searching for 4D Ajax Framework from the method named DAX_DevHook_Preferences. Here is an example:

```
   ` Project method: DAX_DevHook_Preferences
DAX_Dev_SetPreference (8;"True")   ` Consider @ as a character (not
wildcard) for Query
```

Please note that calling the command **DAX_Dev_SetPreference** will apply the rule to the whole database. The developer can still override the wildcard option to a specific field in the method **DAX_DevHook_OnQuery** or **DAX_DevHook_DCS_SetSelection**. Here is an example:

```
   ` Consider @ as a character (not wildcard) for Query
DAX_Dev_SetWildCardMode (Table name(1);Field name(1;3);True)
```

For a DCS search, wildcard option can be overridden in **DAX_DevHook_DCS_SetSelection**. Here is an example:

```
     ` Consider @ as a character (not wildcard) for Query
  DAX_Dev_SetWildCardMode (“MyDCS”;”MyDCSField”;True)
```

DAX_Dev_SetWildCardMode

This method is added to v11.2 to allow 4D Developer to override the wildcard option to a specific field. This method must be executed only within the method ***DAX_DevHook_SaveRecord*** or **DAX_DevHook_DCS_SetSelection.**

```
Syntax:   DAX_Dev_SetWildCardMode (Table Name;Field Name;Wildcard Off)
```

Passed:

$1	TEXT	Table or DCS Name
$2	TEXT	Field Name
$3	BOOL	True to consider @ as a character (not wildcard) for Query

Please note that feature is available only in the v11 SQL component.

Query Filter

DAX_DevHook_QueryFilter is used by the developer to change the selection when it is about to be sent to the front-end. This method is always called whenever 4D Ajax Framework is about to display a selection. A pointer to a longint array of table numbers and a pointer to a text array of set names are passed in. The developer can then remove records that are not valid. For instance if you have a 'Deleted' flag field you could remove all of the records that are flagged as deleted.

Parameters

Passed:

- $1 POINTER - Pointer to a longint array of Table Numbers
- $2 POINTER - Pointer to a Text array of Set Names

Returned:

- By default $1 is assigned to the local variable $setTableIDs_p and $2 is assigned to the local variable $setNames_p.

Implementation

For every table in the $setTableIDs_p array there is a corresponding Set in the $setNames_p array. If you need to modify the selection you should: 1. Use Set ($setNames_p{$index}->) 2. Modify the selection 3. Update the set by Clearing and recreating (or Union etc.)

Example

```
For ($i;1;Size of array($setTableIDs_p->))

    Case of

        : ($setTableIDs_p->{$i}=Table(->[My_Table]))
            USE SET($setNames_p->{$i})
            CLEAR SET($setNames_p->{$i})
            QUERY SELECTION([My_Table];[My_Table]My_Field="My_Value")
            CREATE SET([My_Table];$setNames_p->{$i})

    End case

End for
```

NOTE: The passed sets must still exist after this call. If you use Clear Set be sure to use Create Set with an identical set name.

Record Calls

Commands

Add a Record

Command /DAX/AddRecord

This command create a record based on the given data of specific fields and tables.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N
recordid	Numeric	-1 (Fixed value)
field	[Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID]
field	[Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID]
...	...	1, 2, 3,..., N [Table ID][Field ID]

Sample call:

```
http://localhost:8000/DAX/AddRecord?sessionid=S9182006120617CCWB&tableid=1&reco
rdid=-1&field[1][1]=John&field[1][2]=Smith&field[1][3]=39
```

Reply

If the record[s] is successfully created, the success status and the record id will be sent to the frontend.

```
<addedRecord success="true">
  <recordid>[1][0]</recordid>
</addedRecord>
```

If there is one or more picture fields in the table that the record is created for, the id of the picture field will be included in response as well.

```
<addedRecord success="true">
  <recordid>[1][0]</recordid>
  <binaryfield fieldid="[1][4]">0</binaryfield>
</addedRecord>
```

Please note that the AddRecord command doesn't save a picture. If you want to save a picture into a record, you need to call the command UploadBinary.

Delete a Record

Command /DAX/DeleteRecord

This command delete one or more records based on the given record id(s).

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N (+/-)

recordid	Numeric][Numeric],[Numeric][Numeric], ... ,[Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID]

For example:

1. Delete one record

```
   tableid  = 1 (Physical, View or DCS Table ID)
   recordid = [1][4]  ([Physical Table ID][Record ID])

http://localhost:8000/DAX/DeleteRecord?sessionid=S9182006135719JGGP&tableid=1&r
ecordid=[1][4]
```

REPLY

If the record[s] is successfully deleted, the success status and the record id will be sent to the frontend.

```
<result tableid="1">
  <recordErased error="none" recordid="[1][4]">true</recordErased>
</result>
```

2. Delete multiple records

```
   tableid  = 1 (Physical, View or DCS Table ID)
   recordid = [1][4]  ([Physical Table ID][Record ID])
   recordid = [1][5]  ([Physical Table ID][Record ID])
   recordid = [1][7]  ([Physical Table ID][Record ID])

http://localhost:8000/DAX/DeleteRecord?sessionid=S9182006135719JGGP&tableid=1&r
ecordid=[1][4],[1][5],[1][7]
```

REPLY

If the record[s] is successfully deleted, the success status and the record id will be sent to the frontend.

```
<result tableid="1">
  <recordErased error="none" recordid="[1][4]">true</recordErased>
  <recordErased error="none" recordid="[1][5]">true</recordErased>
  <recordErased error="none" recordid="[1][7]">true</recordErased>
</result>
```

Modify a Record

Command /DAX/ModifyRecord

This command modify a record based on the given record ids and values.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N (+/-)
recordid	[Numeric][Numeric]	1, 2, 3,..., N [Table ID][Record ID]
field	[Numeric][Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID][Record ID]
field	[Numeric][Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID][Record ID]
...	[Numeric][Numeric][Numeric]	1, 2, 3,..., N [Table ID][Field ID][Record ID]

Sample call:

```
http://localhost:8000/DAX/ModifyRecord?sessionid=S9182006135719JGGP&tableid=1&r
ecordid=[1][2]&field[1][1][2]=John&field[1][2][2]=Doe
```

Reply

If the record[s] is successfully deleted, the success status and the record id will be sent to the frontend.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<addedRecord success="true">
  <recordid>[1][2]</recordid>
</addedRecord>
```

Please note that the reply to the frontend is the same as the reply of the command AddRecord.

Record Lock Status

Command /DAX/RecordLockStatus

This command returns true or false for the locked status of the passed record.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	3
recordid	[Table ID][Record ID]	[3][5]

Sample call:

```
http://localhost:8000/DAX/RecordLockStatus?tableid=3&recordid=[3][5]&sessionid=
S9112006173404CCWB
```

Reply

```
<result tableid="3">
  <recordLocked recordid="[3][5]">False</RecordLocked>
</result>
```

Lock a Record

Command /DAX/LockRecord

This command allows you to explicitly lock a record. Generally you shouldn't need to call LockRecord directly as the back-end will handle that when you request an individual record.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	3
recordid	[Table ID][Record ID]	[3][5]

Sample call:

```
http://localhost:8000/DAX/LockRecord?tableid=3&recordid=[3][5]&sessionid=S91120
06173404CCWB
```

Reply

```
<result tableid="3">
  <recordLocked recordid="[3][5]">True</RecordLocked>
</result>
```

Unlock a Record

Command /DAX/UnlockRecord

This command allows you to explicitly unlock a record. The lock will only be released if it is owned by the requesting user.

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	3
recordid	[Table ID][Record ID]	[3][5]

Sample call:

```
http://localhost:8000/DAX/UnlockRecord?tableid=3&recordid=[3][5]&sessionid=S911
2006173404CCWB
```

Reply

```
<result tableid="3">
  <recordLocked recordid="[3][5]">False</RecordLocked>
</result>
```

Record Locking

To prevent multiple users from trying to modify the same record at the same time, 4D Ajax Framework implements a record locking system. This system keeps track of records and the users editing them to prevent conflicts. It is important to understand that the 4D Ajax Framework record locking system does not actually lock records at the Engine level. The system uses interprocess arrays to handle the states of 4D Ajax Framework locks. This means that it is not compatible with multiple 4D servers (or Client servers) running in a Round Robin setup.

Implementation

Every time a 4D Ajax Framework user attempts to load a record for editing the locked status of that record is checked. If the record is not currently locked by another user, a lock is set for the requesting user. If the record is currently locked by another user the requesting user is shown the record in Read Only mode. If the record is already locked by the requesting user the lock is refreshed.

When a user closes the detail for a record the lock that was set for that user is released. A lock is also automatically released after a set period of inactivity which can be set in the method DAX_DevHook_Preferences using the variable <>DAX_LOCK_TIMEOUT_m (by default 5 minutes). Set <>DAX_LOCK_TIMEOUT_m to ?00:00:00? to never timeout a lock.

Developer Responsibilities

While 4D Ajax Framework handles all necessary record locking tasks within its own processes, it is up to the developer to respect these locks in any code that does not interact with 4D Ajax

Framework. If your database has Client/Server connections or handles web connections outside of the 4D Ajax Framework you will need to add calls to check the locked status of records with 4D Ajax Framework.

When you are loading a record for the user to edit in one of your own processes you should call DAX_Dev_Lock_IsRecordLocked to verify that 4D Ajax Framework has not currently given a web user exclusive access to the record. If the method returns True then you should not allow editing of the record. Note that DAX_Dev_Lock_IsRecordLocked checks both the 4D Engine locked state and the 4D Ajax Framework locked state to determine locked status of a record.

Example:

```
$locked_b:=DAX_Dev_Lock_IsRecordLocked($tableNumber_l;$recordNumber_l)
```

Structure Calls

Commands

Get Table List

Command /DAX/GetTableList

Gets the list of all tables.

Sample call:

```
http://localhost.8000/DAX/GetTableList?sessionId=S9122006172949ZUDL
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB

Reply

Sample reply:

```
<tables>
  <table alias="Customers" calendarview="False" datatreeview="False"
id="1" name="Customers" visible="True"/>
  <table alias="Invoices" calendarview="False" datatreeview="False"
id="2" name="Invoices" visible="True"/>
  <table alias="Line Items" calendarview="False" datatreeview="False"
id="3" name="LineItems" visible="True"/>
</tables>
```

Get Single Table

Command /DAX/GetSingleTable

Gets the properties of a table.

Sample call:

```
http://localhost.8000/DAX/GetSingleTable?sessionId=S9122006172949ZUDL&tableid=1
```

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB

tableid	NUMERIC	1, 2, 3,..., N

Reply

Sample reply:

```
<tables>
  <table alias="Customers" calendarview="False" datatreeview="False"
id="1" name="Customers" visible="True"/>
</tables>
```

Get Field List

Command /DAX/GetFieldList

Gets the list of all fields including the properties for a table.

Sample call:

`http://localhost.8000/DAX/GetFieldList?sessionid=S9122006172949ZUDL&tableid=1`

Parameters

Name	Type	Example Value
sessionId	TEXT	S9112006173404CCWB
tableid	Numeric	1, 2, 3,..., N

Reply

Sample reply:

```
<fields tableid="1">
  <field alias="Customers" breaklevel="0" format="" id="[1][1]"
indetail="True" indexed="True" inlist="True" invisible="False"
mandatory="False" name="InvoiceID" nonEnterable="False"
nonModifiable="False" type="longint" unique="False" choicelist=""/>
  <field alias="Total" breaklevel="0" format="" id="[1][2]"
indetail="True" indexed="False" inlist="True" invisible="False"
mandatory="False" name="Total" nonEnterable="False" nonModifiable="False"
type="longint" unique="False" choicelist=""/>
  <field alias="CustID" breaklevel="0" format="" id="[1][3]"
indetail="True" indexed="True" inlist="True" invisible="False"
mandatory="False" name="CustID" nonEnterable="False"
nonModifiable="False" type="longint" unique="False" choicelist=""/>
</fields>
```

User Access Calls

Login

Command /DAX/Login

User login via User Name and Password.

Sample call:

```
http://localhost:8000/DAX/Login?username=john&password=1234
```

Parameters

Name	Type	Example Value
username	TEXT	john
password	TEXT	1234

Reply

Sample reply:

```
<loginresult success="true">
  <sessionid AdminFlag="no">S9192006121749IUTR</sessionid>
</loginresult>
```

Logout

Command /DAX/Logout

User logout.

Sample call:

```
http://localhost:8000/DAX/Logout?sessionid=S9192006121749IUTR
```

Parameters

Name	**Type**	**Example Value**
sessionid	TEXT	S9192006121749IUTR

Reply

Sample reply:

```
<result>
  <logoutSuccess sessionid="S9192006121749IUTR">true</logoutSuccess>
</result>
```

DAX_Dev_Lock_IsRecordLocked

DAX_Dev_Lock_IsRecordLocked determines if a record is currently locked by another user. It first checks if the record is locked by the 4D database engine and then checks if the record is locked.

`Passed:`

$1	LONGINT	`Real or Virtual table number`
$2	LONGINT	`Record number`

Returned:

$0	BOOLEAN	the Record locked (True=Locked, False=Not locked)

A Virtual table number is specific to calls from other 4D Ajax Framework methods.

Implementation

DAX_Dev_Lock_IsRecordLocked first checks with the database engine to determine if the record is locked through standard 4D actions. The selection and its Read Write state are preserved. If the record has not been locked at the engine level, 4D Ajax Framework checks if the record has been locked within the 4D Ajax Framework system. If the record has been locked by a different user, and neither that user's session has expired nor the lock has expired, then this method returns True. Otherwise this method returns False and the record may be edited.