Forms Revisited

By Luis Pineiros, Technical Services Team Member, 4D Inc.

Technical Note 09-20

Table of Contents

Abstract

Forms have been around since the introduction of 4D. Every application has at least two kinds of forms, one to list records and one to add and edit records. The complexity of the forms depends on the designer and the type of application. This Technical Note provides you with some ideas and techniques to plan your next application or apply to your current one.

Introduction

To discuss forms, we need to approach them from the viewpoint of the application that we intend to deliver to our users. Therefore, some advanced analysis and planning should provide us with the necessary information to decide the type of forms we need to create. In addition, once we decide the type of forms, we should also strive to keep a level of consistency. The elements we use in a form, the size of the forms, the elements shared by different forms, the form properties, etc., are all important aspects to keep in mind.

This Technical Note covers the following topics:

- Analyze the different types of forms
  - Input Output forms
  - Project Forms
  - Form Wizard
  - User Forms
- Elements of a form
- Form properties
- Establish the ideal size of forms
- Creating inherited forms to maintain consistency
- Multi-page forms
- Styles Sheets
- Resizing forms
- Hiding and showing elements to maximize the user's experience

What is a form?

You are probably thinking: Why ask that question? We all know what a form is. Well, according to 4D's documentation:

"The form is the interface object that you use for data entry, for listing records, for printing reports and mailing labels, and (in custom applications) for custom dialog boxes and palettes."

This definition just about covers what a form is. Perhaps we need to add the fact that a form does not necessarily have to cover only one of these functions. It can have multiple functions at the same time.

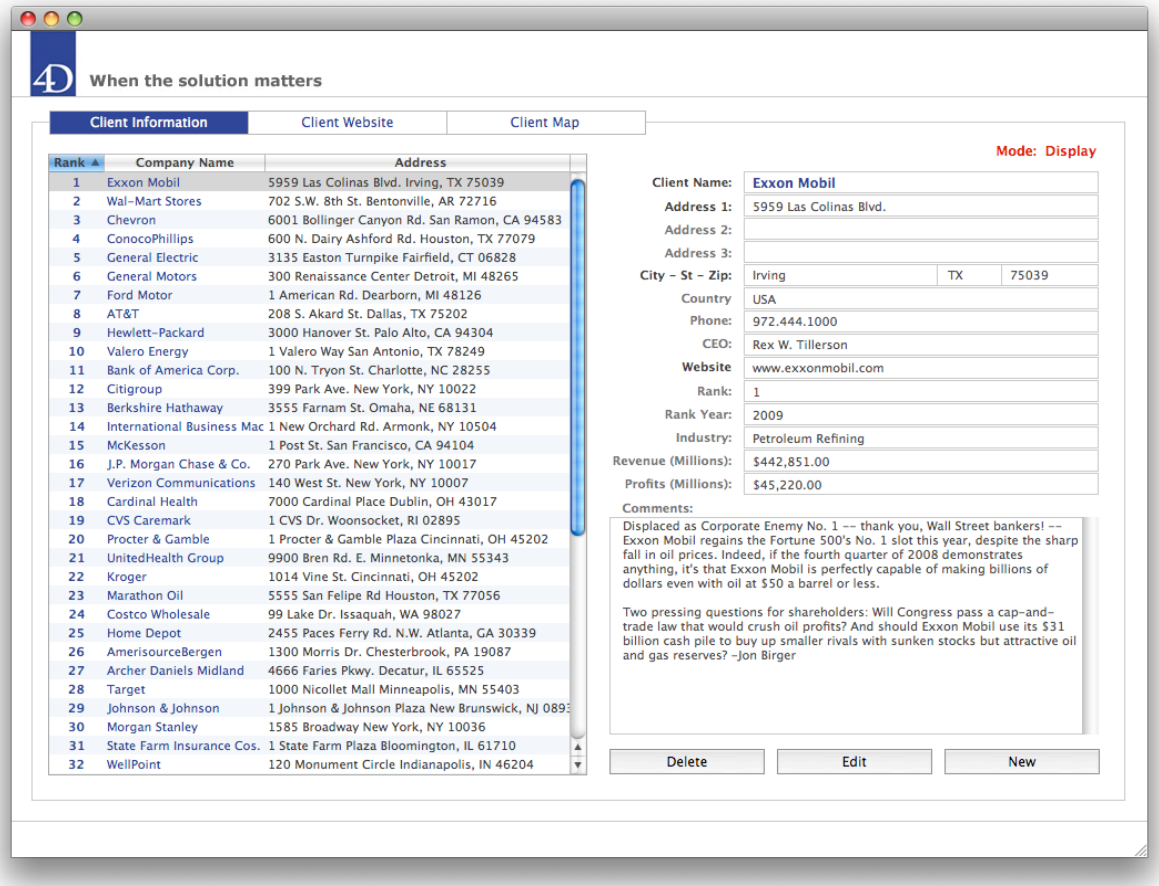
Types of Forms

Input and Output Forms

Traditionally 4D offers a form to list records, several at a time, and a form to add or edit records one record at a time. To this concept we can also add a combination of the two, where you can have a list of records as well as an area where you can display each record individually and add or edit information if necessary.

The demonstration database has a combined form, so you can see the possibility of having output and input forms combined.

When you click on a record on the list, the information gets displayed on the right. No record is locked at this point, so you can view the information and not prevent other users from editing the record. When you click on Edit or double-click on a record, the information can be edited. This approach allows you to provide a quick and efficient interface for navigating records without having two different forms. Other functionalities can also be added to this basic concept. You can have query and custom query functions, display selection functions, etc.



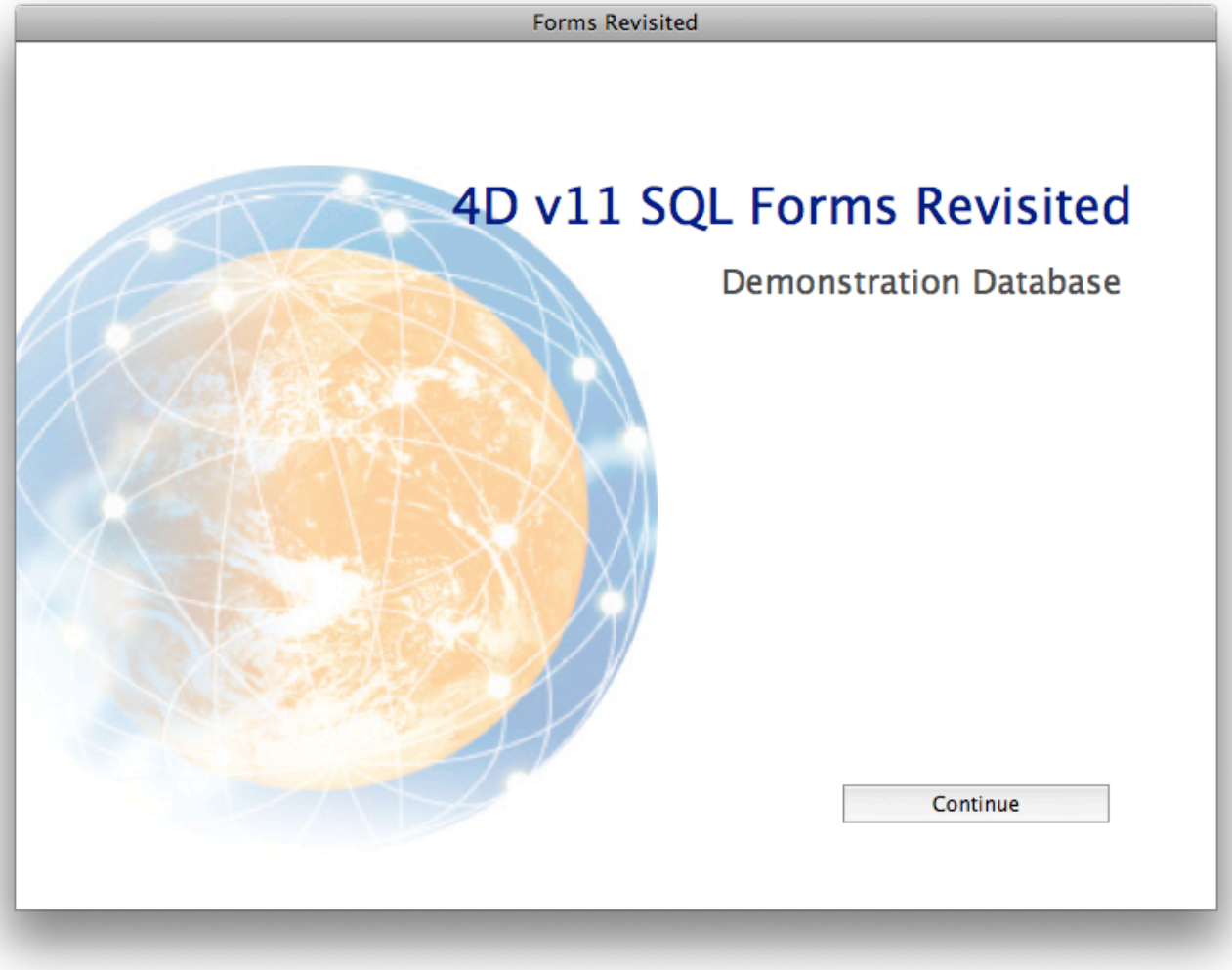
Project Forms

Project Forms are a new type of form introduced in 4D v11 SQL. This type of form is not attached to a table. This concept allows you to create and manage forms that do not require the elements of a table. You can use Project Forms for interface dialogs or Components. In the past, we had to create tables just to have these types of forms available to our users, but that is no longer necessary.

Project Forms are different than Table forms in that:

- No list forms can be created with Project Forms. Only detail forms are available.
- Project Forms cannot be designated input or output forms.
- Project Forms cannot be used in label editor or 4D import/export editors.

The demonstration database startup screen is an example of a Project Form.



Form Wizard Forms

Forms created with the Form Wizard are ideal for tables that the user has no access to. The Form Wizard saves you time by creating the Input and Output forms automatically based on templates you can create. That said, These generic forms should typically not be used as created by the Form Wizard to convey information to the user. Use them to start your forms; otherwise, you risk depriving your users from a whole range of features and conveniences available in 4D v11 SQL forms.

User Forms

There are cases where no matter how hard you try to create the ideal form, you do not succeed. In essence, it has to do with graphic elements of the form. For instance, a company logo, custom reports, etc. Keep in mind that in User Forms, the user cannot add object methods, variables or fields. Users can change the entry order or add active objects from a library. For more information about User Forms, refer to the User Forms section in the 4D v11 SQL Design Reference at the following address:

http://4d.com/support/documentation.html

Elements of a Form

Instead of describing every object in a form, which can be found in the 4D v11 SQL Design Reference, here is a description of a more general approach to the elements of a form. It may help to think of a form as a graphic layout; to visualize the amount of space and information that is necessary to present to the user. For the purposes of maintaining a unified and consistent look, divide the form into 3 different sections: Header, Body and Footer. Simple but effective, this way you can easily divide forms into types and create a template for each type taking into consideration those 3 different sections.

(1) The header can contain a company branding section, information about the record, information about the user, information about the section, navigation, etc.

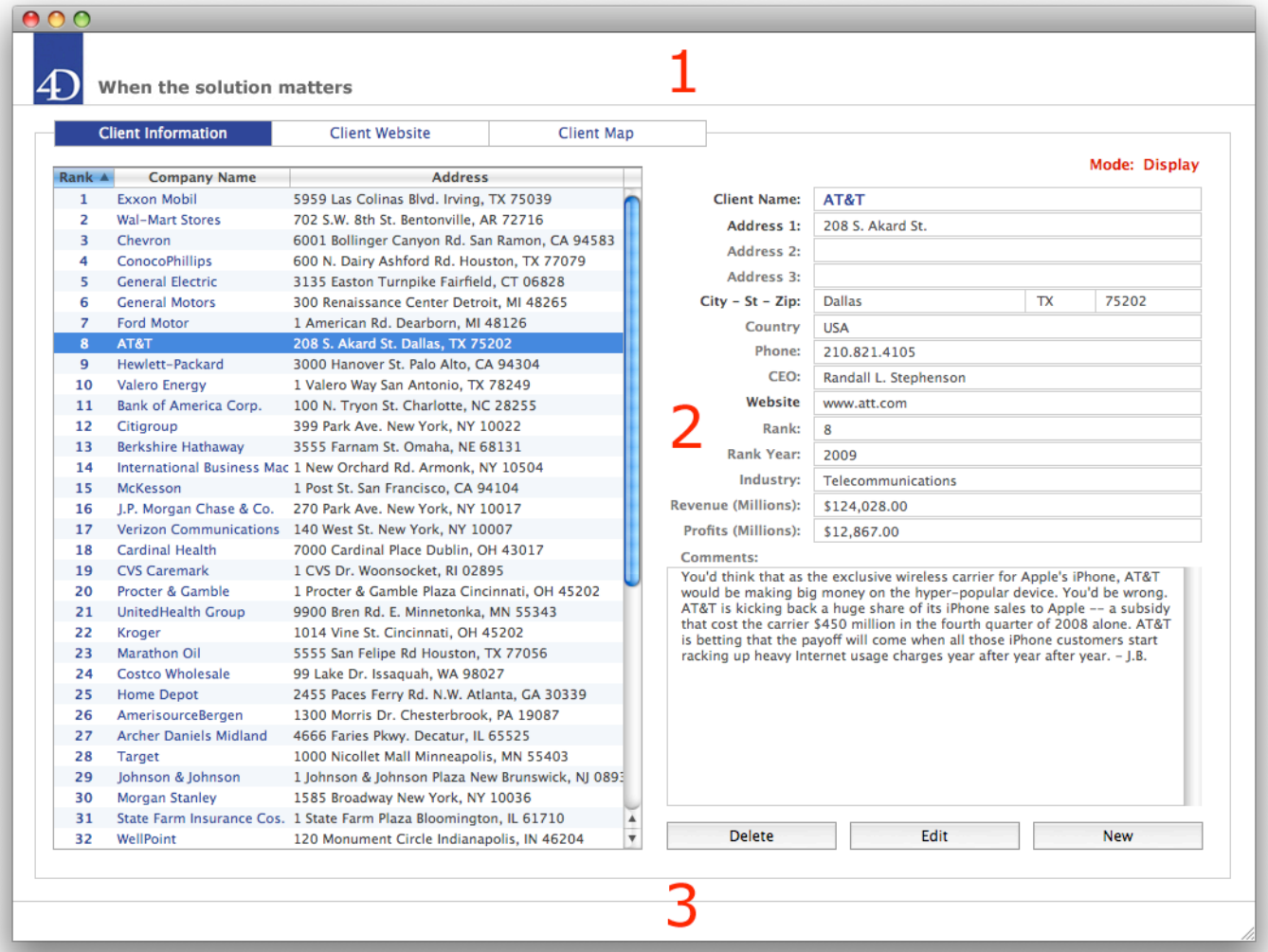
(2) The body can contain the record itself, information from other tables, information from other data sources, lists, tabs, hidden elements to be shown on command, etc.

(3) The footer can contain data consolidation results, information about the record, status, navigation, etc.

As you can see, deciding what goes where is up to you, there is some overlap in terms of the 3 sections; however, once you decide what elements occupy each space, you should have the same basic layout throughout your application. The expectation of the user would be to find the same elements in the same place in every form you present to them. This does not mean that all of your forms have to

look the same. On the contrary, a variety of layouts is encouraged to meet the needs of different types of users. Just keep in mind that variety does not mean inconsistency or lack of standards. For example, if you decide to put your navigation in the footer of the form, all the forms should have the navigation in the footer.

You should be able to see a consistent look in all the forms of the sample database. Here's a quick layout breakdown:



Form Properties

The properties you assign to the form have a great impact on the way the form operates. Pay close attention to the way the form has been structured from the standpoint of its properties.

All form properties are important, but these are some of the form properties that are usually worth setting for every form.

- Form Name: Always name your forms

- Inherited Form Name: If possible, use an Inherited Form to save time and ensure consistency

- Size based on:

    - Set size for forms that you do not want the user to resize. With the Fixed Width and Fixed Height checked in the Window Size property.
    - Set Size for forms that you want the user to resize.
        - With The Fixed Width unchecked and a minimum and maximum sizes.
        - With the Fixed Height unchecked and a minimum and maximum sizes.

- Associated Menu Bar: Always associate a menu to your forms.
  Active Menu Bar: checked and use **SET MENU BAR** in your code before opening a window.

  Not associating a menu to a form leaves the form with no other choice but to acquire the current menu. This can create issues when the menus and menu items perform functions that do not correspond to the form. Your users can get easily confused if you provide them with menu choices that make no sense in the context of the form.

  The general rule in the case of menus associated with forms is to have a menu bar with menus and menu items that allow the user to perform an action that corresponds to the information provided in the form. When the menu and/or menu items have Methods or Standard Actions associated to them that do not make sense in the current form, you can disable the menu items until you need them again. You can also create menus with menu items that only have the functions that correspond to the form being displayed. In this case you should manage your menus so the by making sure a menu is associated in your form as well as assigned in the code for opening the window using **SET MENU BAR**.

  The demonstration database has four forms used to replace the default 4D v11 SQL **Confirm** and **Alert** dialogs: Two confirm dialogs - one for Windows and one for Mac - and two alert dialogs - one for Windows and one for Mac. These forms have the same menu associated as the forms calling the dialog. However, leaving all the menu items in the main menu enabled would be confusing. There are functions in these menu items that would open a new window or navigate to a different form. Therefore, when calling the dialogs, the menu items that do not apply get disabled. The other option would be to create and associate a separate menu for these dialogs and manage the menus for the different forms in the code.

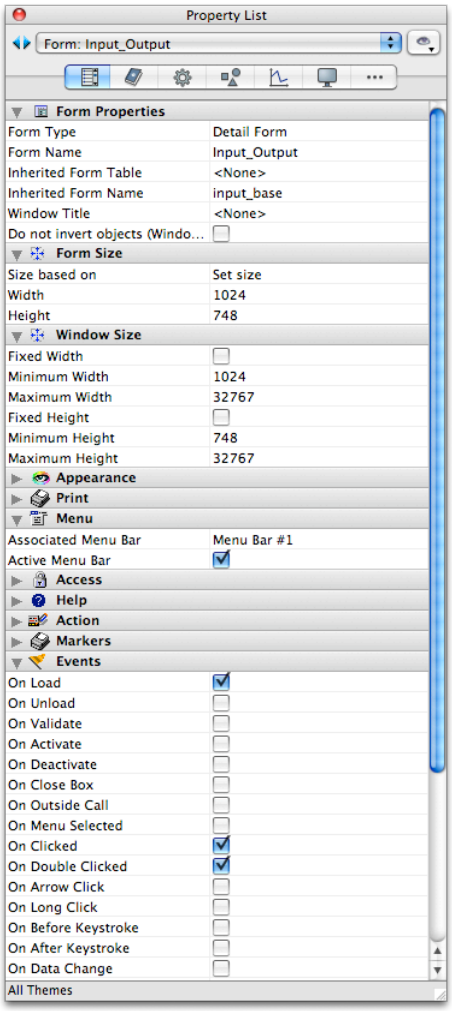
- Method: One option is to have a form Method that calls other methods. In the demonstration database, the Input/Output form has a form method called: FR_Input_Form ("Form Event")

  This way, any Object in the form can have the same Form Method with the appropriate call. For example, a Save button in the form could have a method: FR_Input_Form ("Save Record").

```
  `.....................................................................
  `   Project Method:  FR_Form
  `   Description:  handles  Input&Output Form Events
  `   Called by:   Add/Modify/Form
  `   Calls:  Nothing
  `   Parameters:  $1  string - event
  `   Returns:   Nothing
  `   Created by luis piñeiros
  `   2009
  `.....................................................................

C_INTEGER($i)
C_LONGINT($rec;$prod;$sales;$docs)
C_TEXT($website;$fedex;$url)

C_INTEGER($left;$top;$right;$bottom)
C_LONGINT($CurrentRec_l)

Case of
    : ($1="Form Event") & (Form event=On Load )
          FR_Tabs (1)
          FR_Set_Client_NonEnterable
          FR_Load_Clients
          FR_Form ("Display Blank Client")
          SORT LISTBOX COLUMNS(*;"LB_Clients_Ob";1;>)
          SELECT LISTBOX ROW(*;"LB_Clients_Ob";1;0)
          FR_DisplayClient

    : ($1="Form Event") & (Form event=On Resize )
          Case of
          : (Current form page=2) | (Current form page=3)
          FR_Set_Client_Min_Resize

          End case

    : ($1="Tab 1")
          FR_Tabs (1)

    : ($1="Tab 2")
          FR_Tabs (2)

    : ($1="Tab 3")
          FR_Tabs (3)

    : ($1="Phone Number")
          Case of
          : (Form event=On Getting Focus )
          SET VISIBLE(*;"Label_ClientPhone2";True)
```

```
          : (Form event=On Losing Focus )
          SET VISIBLE(*;"Label_ClientPhone2";False)

          End case

   : ($1="Delete Client")
          FR_DeleteClient

   : ($1="Save Client")
          FR_SaveClient

   : ($1="Cancel Client")
          FR_Set_Client_NonEnterable
          vClientMode_t:="Mode:  Display"
          FR_DisplayClient

   : ($1="New Client")
          FR_Set_Client_Enterable
          FR_Display_BlankClient
          vClientCountry_t:="USA"
          vClientMode_t:="Mode:  New"
          GOTO AREA(*;"vClientName_t")

   : ($1="Edit Client")
          FR_EditClient

   : ($1="Display Client")
          FR_DisplayClient

   : ($1="Display Blank Client")
          FR_Display_BlankClient

   : ($1="Display Map")
          FR_Display_Map

   : ($1="Display Website")
          FR_Display_Website

End case
```

- Events: Check all the events that apply to the Form and its Objects.



Ideal Size of a Form

What is the ideal size of a form? This is certainly not a question that can have only one answer. The ideal size of a form depends on different factors. It is also dependent on the amount of information you need to provide, on the type of environment and workstations that will run the application, etc. Again, this would be a good time to put a plan together to come up with the ideal size for each type of form in your application.

These are some considerations:

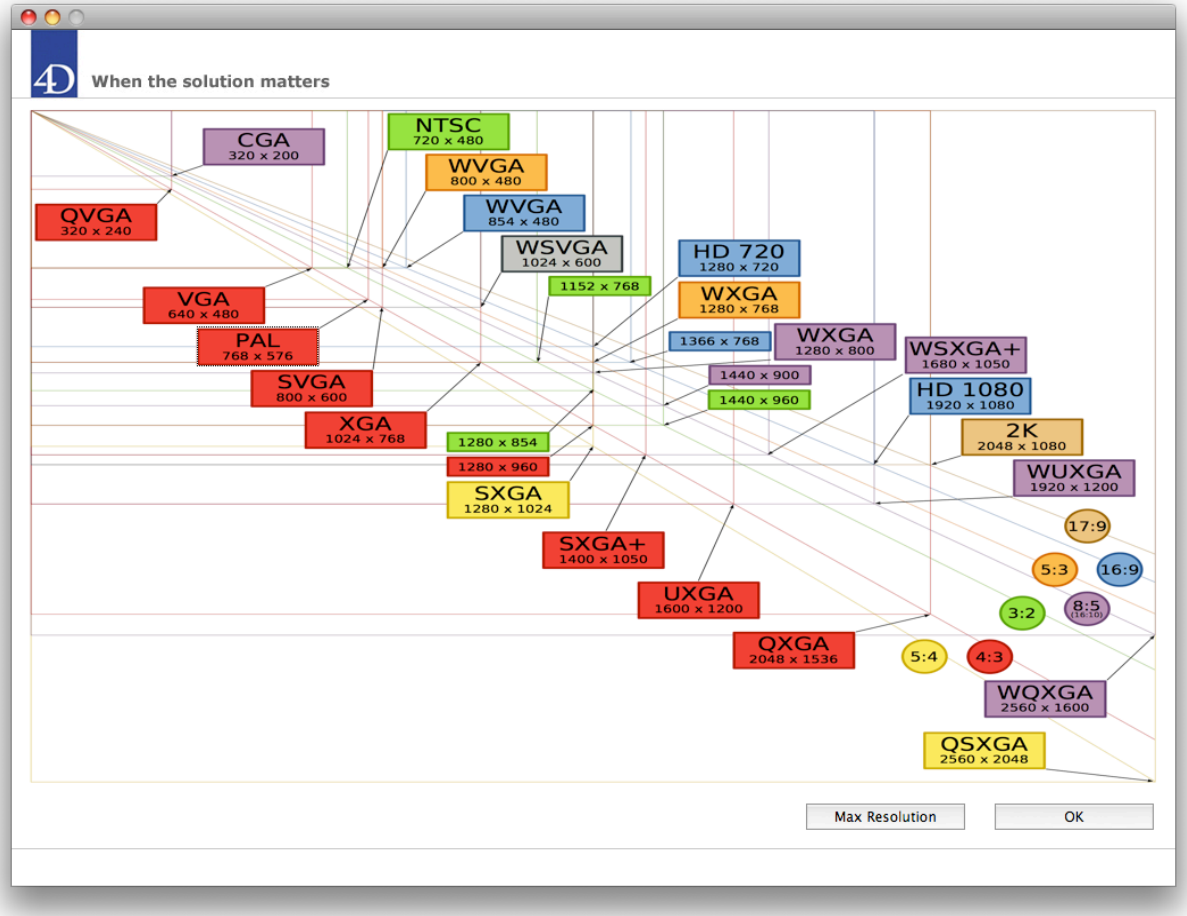
- Make the size of your main forms adjust to a known resolution for today's operating systems. In a Mac OS X or a Windows platform workstation running 4D v11 SQL, the sizes for displays can range anywhere from 800 x 600 to 1920 x 1200. Select a resolution that fits the available hardware and your application's needs. Do not confine your forms to a low resolution just because a few of the workstations may not offer the desired size. It is always easier and more cost effective to obtain the appropriate hardware than to

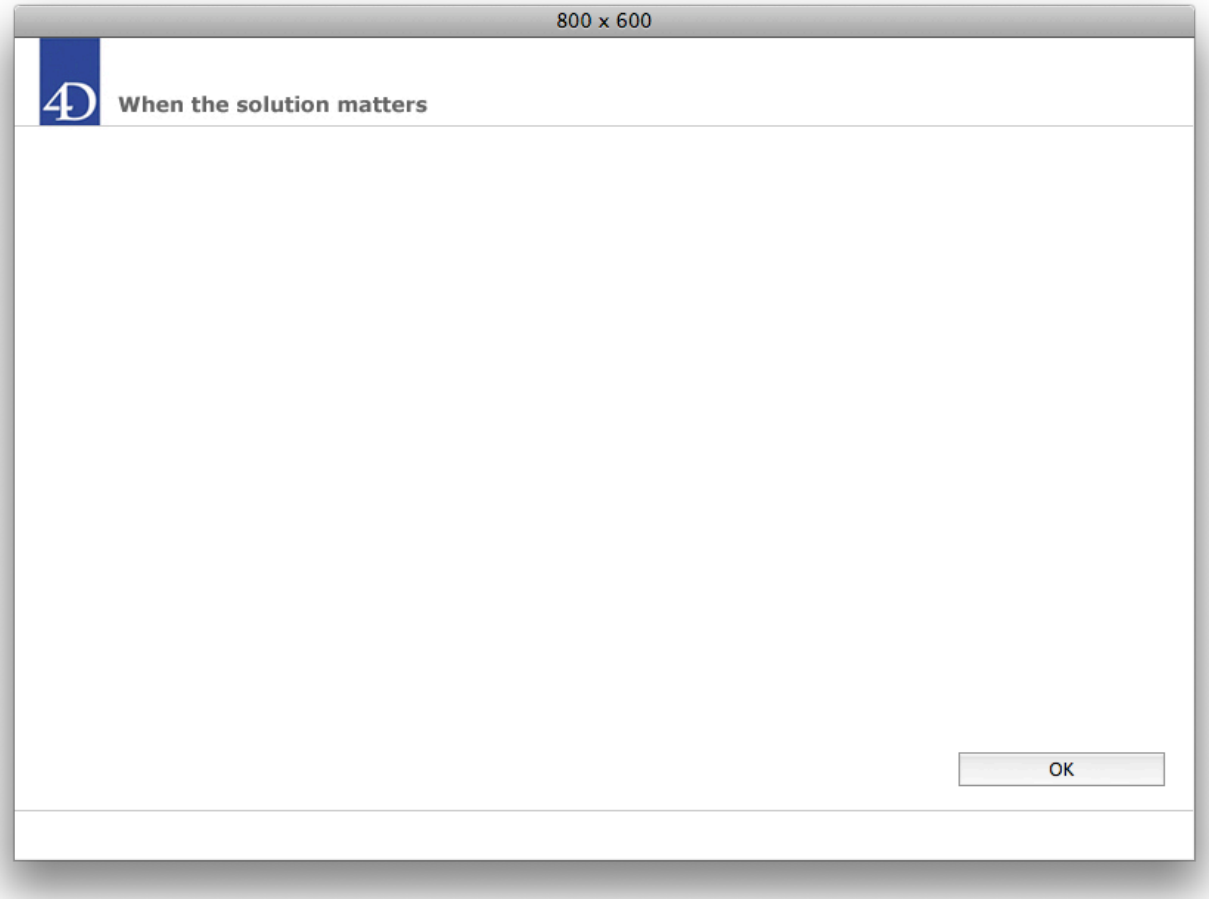
spend time, money and resources updating your application. Or worse yet, to have most of your users spending more time scrolling and navigating unnecessarily from form to form or from page to page.

- Secondary forms can be any size smaller than the principal forms in your application. Make sure to decide on what size the different types of forms should be to keep consistency.

- A certain amount of empty space should be considered in forms. Too much information in a reduced amount of space creates an environment more prone to user errors. When you feel that you are cramming too much into a form, consider additional pages and/or hidden elements that become visible on command.

The demonstration database contains a form with different display resolutions for you to have an idea of the amount of space available for your forms. Click on the resolution to open a window of the size selected. If your display does not support the size a message lets you know that the size cannot be accommodated in your workstation. To find the maximum resolution available in your display, click on Max Resolution.





Inherited Forms

The amount of common denominators in a 4D application is quite high. In particular when it comes to forms. You soon realize that you can reuse a lot of the elements from one form to another. The most efficient way to do this is to create a form and use it as a template in another form. 4D v11 SQL calls these templates inherited forms. You can have any number of objects in an inherited form. The advantage of using this technique is that when you change your inherited form, all the forms using it also reflect the changes.

The order in which the objects are loaded in a form using an inherited form is:

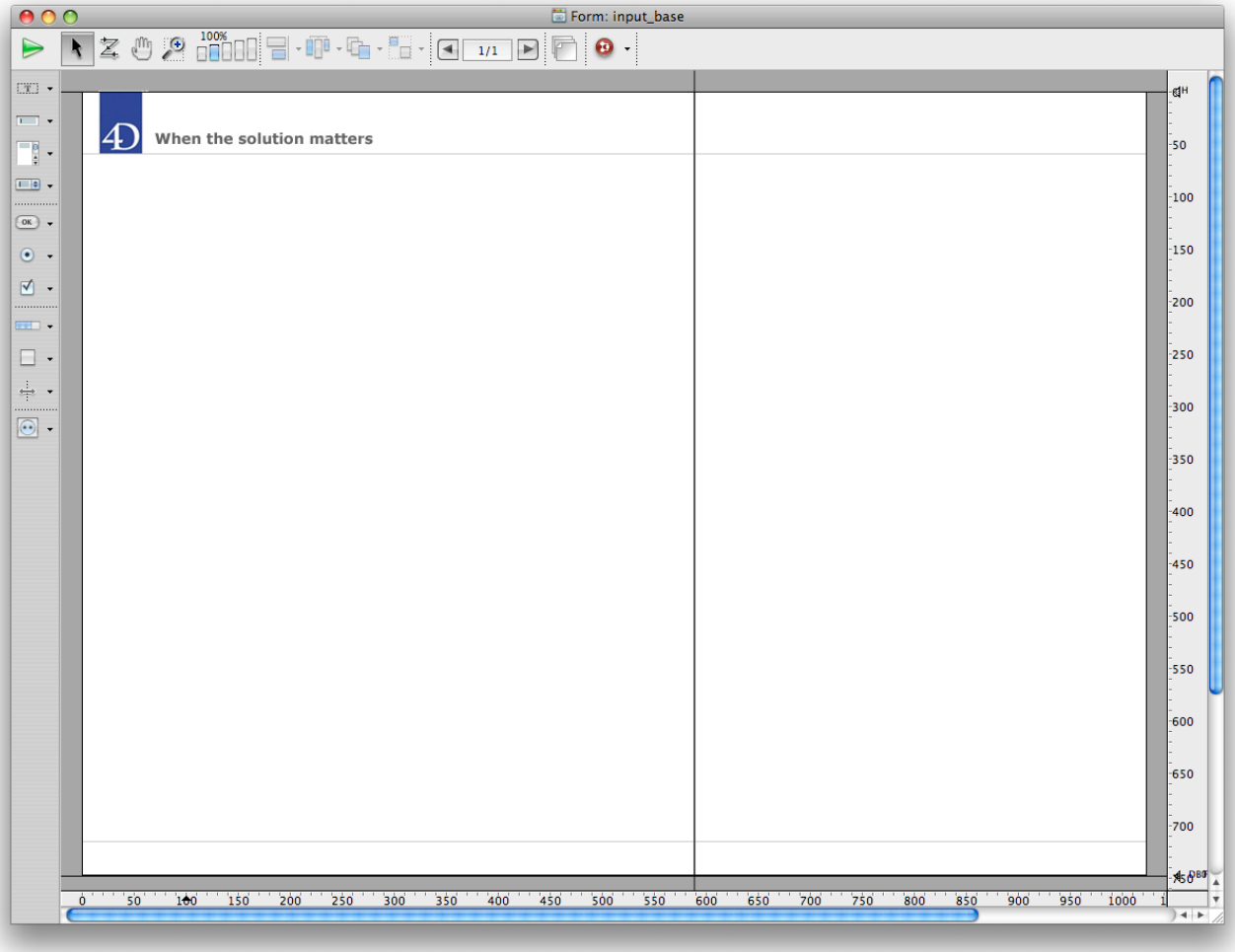
1. Page zero of the inherited form
2. Page 1 of the inherited form
3. Page zero of the open form
4. Current page of the open form

Properties of an inherited form are not considered when using it in another form. However, the methods of the Objects in an inherited form execute in a form calling it. For more information about Inherited Forms, please see the 4D v11 SQL Design Reference documentation.

Ideal elements of a form to be included in an inherited form are:

- Company Identity information (Logos, tag lines, etc)
- Header elements (Lines, borders, etc)
- Footer Elements (Lines, borders, navigation, etc.)
- Backgrounds

The demonstration database makes use of an inherited form. It has been setup with the size of the main window in mind. However, the elements of the inherited form are resized based on the size of the form. The white background is a rectangle with Horizontal Sizing and Vertical Sizing set to Grow. The line under the 4D logo as well as the line on the footer are set with Horizontal Sizing to Grow and Vertical Sizing to None.



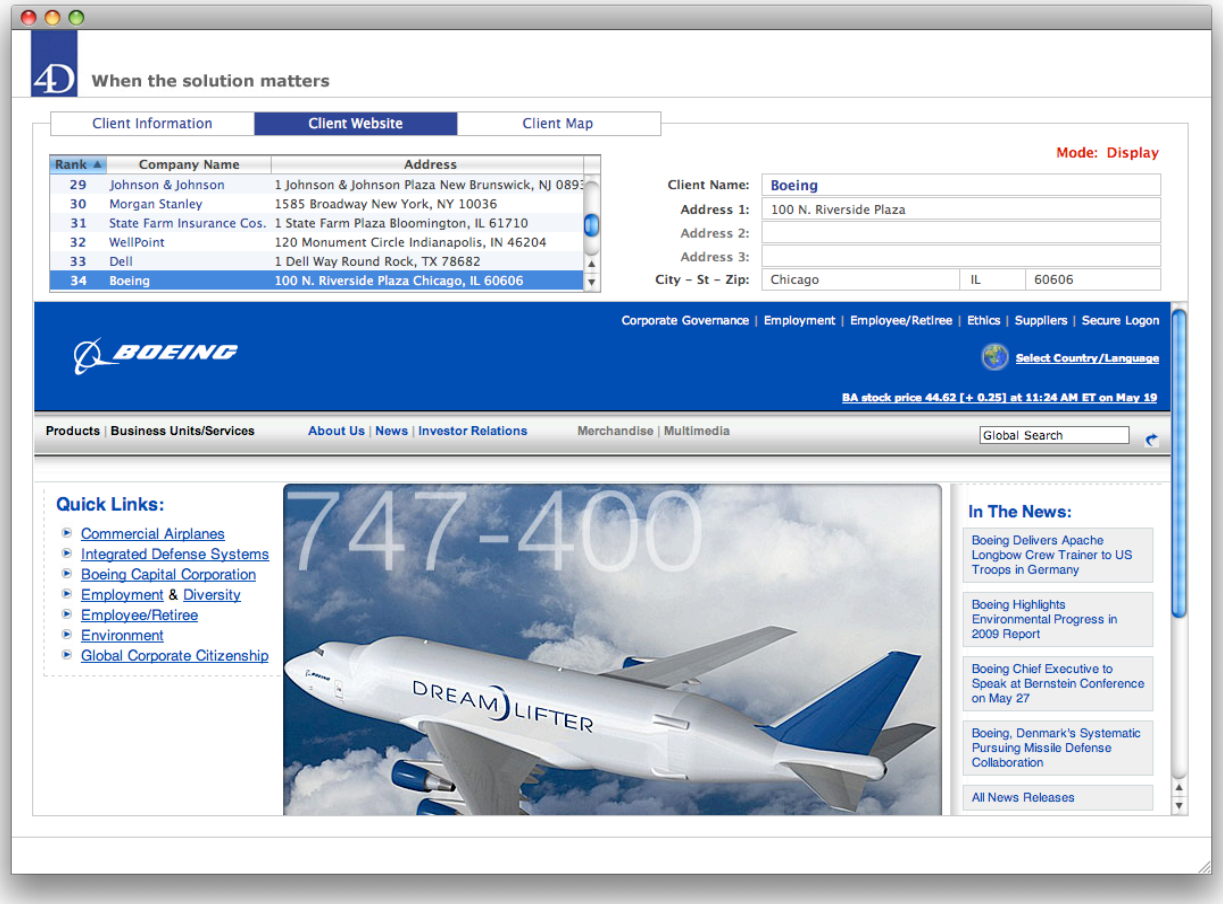
Multi-Page Forms

How do you decide whether to use a multi-page form or multiple forms? A good way to decide is to analyze the context of the information being presented to the user. Good candidates for a multi-page form are:

- If the information can be divided into sections with a level of usage or priority can be assigned to each section
- Tabbed information
- Preferences or Administration
- Reports with multiple Graphs
- Footer Elements (Lines, borders, navigation, etc.)
- Backgrounds

Multi-page forms can also benefit from Inherited Forms. In addition, as with any other form, you can make good use of page zero of the form. If you can visualize your form as having layers, the first layer on the bottom of the page is page zero. This layer is transparent, which means that its contents will be reflected in every page of your form. Any objects that you want to share in your page can be placed in page zero of your form.

The demonstration database makes use of a multi-page form. The Input/Output form has 3 pages. Each has been designated with a tab. Some elements are shared by all the pages and others are unique to each page.



Style Sheets

It would difficult to accomplish creating efficient forms without making use of style sheets. Whether you plan to deploy your application on one operating system or multiple ones, you must consider using style sheets. 4D v11 SQL offers support for Windows XP, Windows Vista, Mac OS X and Windows 2000.

Style sheets allow you to define the font type, size and style for each platform. Once you have defined the different style sheets that you plan to use in your application, applying them in your forms provides you with the following advantages:

- Maintaining your forms is as simple as changing a style sheet. If you change your mind about a font or a size, all you have to do is change it in the style sheet and all the objects using it will reflect the change.
- Saves you time when creating objects since you can apply a style sheet to multiple objects at the same time.

- When deploying in multiple platforms, 4D v11 SQL automatically adjusts to the user's platform. Just make sure the Object has the System appearance property set.
- It helps you keep consistency in your forms. You should always create style sheets for all your text-based objects:
  - Labels
  - Fields
  - Headers
  - Footers
  - Printing Labels
  - Printing fields
  - Printing Headers
  - Printing Footers
  - Buttons
  - Variables
  - List Boxes

Resizing Forms

Providing a base size for all of your forms does not insure that your users are not going to resize them. Furthermore, it should be encouraged whenever possible or applicable to do so. If the user has a display or multiple displays that handle a higher resolution than your base forms, make sure you provide a way to resize the forms.

Resizing forms should take into consideration all the objects in the form. You want the transition from one size to another to be smooth and logical. Do not let objects move and/or resize without controlling the final result. The best way to test is to try different sizes in different displays. Test resizing in a workstation connected to multiple displays.

Repositioning the form may also play a role in the resizing process. Sometimes, moving the form creates a situation that might confuse the user. If you need to move the form, make sure you use the main display to do so. Position the form centered within a space that conveys a clear message.

A good example of repositioning and resizing elements in a form is the list of Clients when the Input/Output form is in page 2 or page 3. The list has been assigned Horizontal Sizing None and Vertical Sizing Grow. However, when in pages 2 and 3, the Web Area covers part of the list. Therefore, if we let the list grow as expected, part of the list would be invisible and the scroll bar will not be accessible. To compensate, what we do is in pages 2 and 3 we resize the list programmatically to stay within the boundaries of the fields on the right side.

Hiding and Showing Elements

Multi-page forms, resizing and inherited forms are all very useful tools to display information in a form. Let's add one more tool to our toolbox. How about hiding and showing elements? You may encounter situations where an additional page does not quite meet the requirements of a form. You want to have information displayed in the same page but not all the time. The user decides when to display the information. This is the ideal scenario for hiding and showing elements of your form. You can select from making objects invisible and visible on command or to move objects to the visible area of a form.

User interface (UI) elements can also be hidden and shown on command. It is disconcerting to the user to have options that do not apply showing when they are not required. For example, if you have a form to create and edit records, when creating a record a DELETE button does not make sense. You should only show an option to delete a record when editing. So, why not hide the DELETE button until the user is editing the record. Does it make sense?

The demonstration database provides an example of hiding and showing elements. In the Input/Output form, the **New**, **Edit** and **Delete** buttons are visible when the Clients records are in **Display** mode and the **Save** and **Cancel** buttons are invisible. On the other hand, when a Client record is in **Edit** or **New** mode, the **Save** and **Cancel** buttons are visible and the **New**, **Edit** and **Delete** buttons are invisible.

Conclusion

Forms are the link between your data and your users. One of the main goals of 4D v11 SQL applications is to maximize the value of your information. Creating forms that are easy to use and powerful at the same time takes you a step closer to that goal.

Before creating forms for your application, have a well-defined and structured blueprint. Provide the users with the information required, do not make things more difficult than they have to be. Remember, complex does not mean powerful. Simplicity can have valuable long-terms effects in the quality of the experience for the user, as well as the information of your system.