TIBCO WebFOCUS®

App Studio Maintain Data Getting Started

Release 8.2 Version 04 and Higher
March 2021
DN4501622.0321

Contents

Chapter 1

Introducing App Studio Maintain Data

The following topics offer introductory information about App Studio Maintain Data for new users.

These topics also provide an overview of the step-by-step process for developing an application, from creating the interface to running the application.

In this chapter:

Road Map: Where Should You Go?

Welcome to App Studio Maintain Data.

- ❑ If you are new to App Studio Maintain Data, read *What Is App Studio Maintain Data?* on page 7 and then work through the *App Studio Maintain Data Tutorial* on page 13.
- ❑ If you are interested in a quick overview of how to develop a App Studio Maintain Data application, read *Overview of Developing App Studio Maintain Data Applications* on page 9.

What Is App Studio Maintain Data?

App Studio Maintain Data is an application development tool that creates web-based data maintenance applications. Using App Studio Maintain Data and its multi-platform 4GL language, you can easily create, test, and deploy complex business applications that span the Internet, IBM mainframes, midrange servers, and workstations.

Challenge of Accessing Information

If your company is like most companies, you probably have several different computing systems in operation. For example, your accounting and payroll systems are running on a midrange UNIX computer, while your inventory system is running on an IBM mainframe. Most of the employees have personal computers on their desks and are accustomed to point-and-click interfaces.

With App Studio Maintain Data, you can develop applications with graphical front-ends that run on personal computers and access data on any computer system in your company.

How App Studio Maintain Data Works

Using the HTML canvas, you develop the application front-end (or user interface). Then, you develop the code that extracts data from your data sources and updates the data sources with new data.

At run time, end users start the application and access a Reporting Server where the data or procedures reside. The application extracts data from the data source, displays it for the end users to see, updates the data source with new information, and runs any procedures.

N-Tier Applications

App Studio Maintain Data applications are called *n-tier applications* because they are capable of distributing processing over many platforms. N-tier applications offer the following advantages:

- **You can access data on multiple platforms,** forming relationships among disparate data sources.
- **Your application logic runs on the machine most capable of performing it.** For example, PCs are ideally suited for displaying the user interface of your application. On the other hand, MVS machines may not be capable of displaying pretty pictures, but they pack plenty of processing power. An App Studio Maintain Data application front-end runs on a PC, whereas the data access code can run on an MVS machine.
- **You can speed up your applications.** Procedures that access data can run on the platform where the data resides, ensuring that any aggregation or screening takes place immediately. This means that your application is not shipping large quantities of data across a network to be aggregated or screened somewhere else. Less network traffic means increased application speed.

Leveraging the Power of the Reporting Server

The Reporting Server is available for every major operating system, MVS, UNIX, Windows, Open VMS, CICS, and VM/CMS. Using its data adapters, App Studio can access every major database management system, including Informix, Sybase, DB2, and Ingres. The Reporting Server can also run procedures written in another language. For example, COBOL, C, or other 3GL programs, CICS transactions, IMS transactions, RDBMS stored procedures, and FOCUS procedures.

Because Information Builders has already worked out the complications of different operating systems, communications protocols, and data access languages, the developer or end user does not need to worry about or even know where their data is coming from.

What App Studio Maintain Data Can Do for You

Using App Studio Maintain Data, you can:

- ❑ Easily develop web-based data maintenance applications with no prior knowledge of HTML, Java®, or complex 3GLs.
- ❑ Access and update data on every major operating system and every major database system (using the power of the Reporting Server).
- ❑ Access and update data from different platforms at the same time. For example, the inventory system on MVS and the accounting system on UNIX.
- ❑ Take advantage of the strengths of your computing systems while sidestepping the weaknesses by partitioning your application among the platforms that can support them.

What Is Next?

For more information on how to use App Studio Maintain Data, work through *App Studio Maintain Data Tutorial* on page 13.

If you want to develop your own App Studio Maintain Data applications immediately, read *Overview of Developing App Studio Maintain Data Applications* on page 9.

Overview of Developing App Studio Maintain Data Applications

Now that you know what a App Studio Maintain Data application is and how it works, you are ready for a step-by-step view of the development process. This section summarizes the steps.

Another way to get started immediately developing App Studio Maintain Data applications is by using Update Assist. All you need is a Master File for the data source for the application you want to create.

Step 1: Creating the Domain

Your first step is to create the Domain for the application.

To start App Studio, from the Start menu, click the *WebFOCUS App Studio* shortcut, under the Information Builders folder.

Step 2: Describing the Data

From the File/Folder Properties panel, set the Application Paths attribute to the location of the Master Files and data source files.

Step 3: Creating the Front End

Your next step in developing an application will probably be to create the front end, meaning the user interface with which the end user interacts. Your user interface is made up of forms, which you develop using the HTML canvas.

1. From the Requests & Data sources panel, create an external or embedded Maintain Data request.
2. Expand the node for the request.
3. Right-click *Forms* and select *New form*.

One of your procedures is designated as the starting procedure, which means that App Studio Maintain Data runs this procedure to run your application. You probably want the opening form of your application to be in this procedure (although it does not have to be here). You use the Properties dialog box to specify the starting procedure.

App Studio Maintain Data supplies you with a form (named Form1) in every procedure and supplies the code to display this form immediately after running the application. Using the name Form1 in the initial form of your application is recommended.

After you have created your forms, you can edit them using the HTML canvas.

You can also create tasks and actions that call Maintain Data functions (Cases).

Step 4: Creating the Data Access Logic

Your next step in developing the application is to create the data access logic, meaning the code that extracts data from the data source, manipulates it, and writes it back to the data source.

You code all of your data access logic using the Maintain Data language. The Maintain Data language is a robust yet simple, object-based 4GL. It is consistent across all platforms while incorporating the functionality of a 3GL and the data access capabilities of SQL.

You must specify which data sources you want a procedure to access before writing any code to read or write to that data source.

Procedure: How to Make a Procedure Access a Database

Specify which data sources you want a procedure to access.

1. If you want to put your data access code in a separate procedure (to take advantage of n-tier processing), create a procedure in your application.
2. In the Requests & Data Sources panel, right-click the procedure, and then click *Use data sources*.
3. Select the data sources this procedure will access and click *OK*.

Procedure: How to Write Maintain Language Code

1. Double-click the procedure to open the Maintain Data Editor.
2. Between the CASE Top and END keywords, specify the code that your procedure will run. You can do this in either of the following ways:
   - ❑ Right-click in the window and click *Language Wizard* in the shortcut menu. Then, follow the instructions to generate your Maintain Data language code.
   - ❑ Type the code that your function runs. For more information, see *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

You can obtain context-sensitive Help by selecting any keyword and pressing the F1 key. Notice also that your procedure code is color-coded.

Step 5: Setting Up Front End and Data Access Procedures to Call Each Other

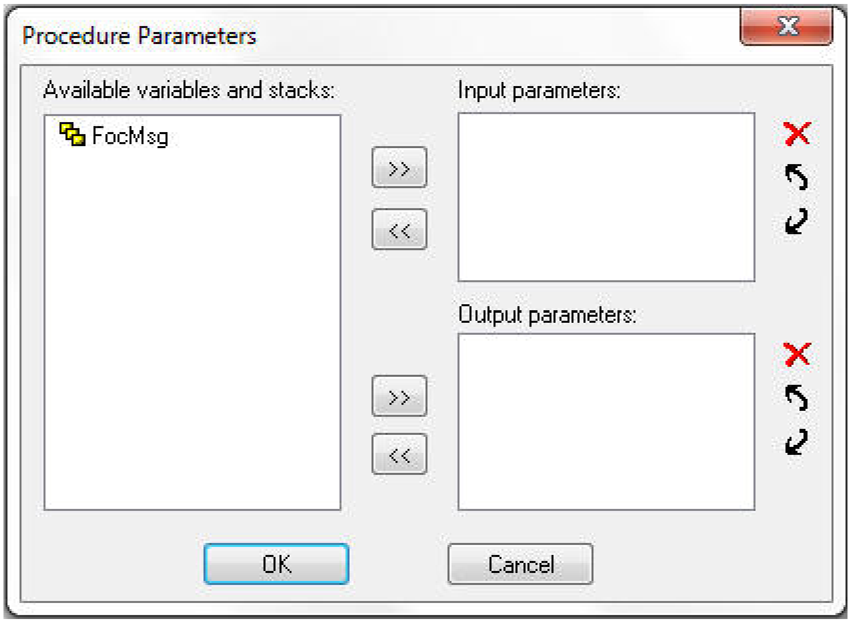
After you have written the front-end and data access procedures, you must set them up to call each other, and pass data back and forth. Since all variables are local, meaning defined only within the context of a procedure, you must pass these variables back and forth.

For example, suppose you have an application running on a Windows web server that accesses accounting data on UNIX and inventory data on MVS. A procedure called GetAccData, accesses the data on the UNIX machine and a procedure called GetInvData, accesses the data on the MVS machine. Both of these procedures pass data back to the Start procedure on the Windows machine, which displays a form with this data.

Procedure: How to Set Up Procedures to Call Each Other

1. Declare all variables in the calling procedure (parent) and called procedure (child).

2. In the called procedure, use the Procedure Parameters dialog box to declare what parameters it expects to receive and what data it passes back to the calling procedure. To open this dialog box, right-click the procedure and in the shortcut menu, click *Add parameters*, as shown in the following image.



3. In the calling procedure, use the CALL command. The parameters of the CALL command determine the data that gets passed to the called procedure and what data gets passed back. You can easily place a CALL statement in your procedure using the Language Wizard.

   For more information, see *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

Step 6: Run the Application

After you have created all of your application procedures, it is time to test them. Click the Run button to run the HTML page to see what it looks like.

If there are any errors, they are listed in the Output window.

Chapter 2

App Studio Maintain Data Tutorial

Welcome to the App Studio Maintain Data tutorial. In this chapter, you will be creating an App Studio FanClub application. This application enables you to maintain a data source of App Studio Maintain Data *fans*. You will be able to add fans, update fans, and view existing fans. This tutorial assumes no prior App Studio Maintain Data experience.

In this chapter:

Before You Begin

Before you begin working on the App Studio FanClub application, make sure you have completed the following steps:

- ❑ Installed and verified a Reporting Server.
- ❑ Installed App Studio on your machine.
- ❑ Verified the connection of your machine to the Reporting Server.

- Checked to see that you have the sample tutorial files required for the App Studio Maintain Data tutorial on your machine. This includes the following files:
  - The fannames data source (both the .mas and .foc files).
  - The addafan.gif, currfan.gif, fan.gif, and spiral.gif image files.

  Note: All of the FanClub files are installed under ibi/apps/maintain. The images are located under ibi/apps/maintain/images.

Creating a Domain

Before you can do any development work in App Studio Maintain Data, you must:

- Create a Domain folder.
- Using the Application Paths property in the File/Folder Properties panel, attach the Maintain Data app folder for Master Files and data files (.foc files).

Procedure: How to Create a Maintain Data Procedure

1. Right-click the Domain folder you created, click *New*, and then click *HTML/Document*.

   or

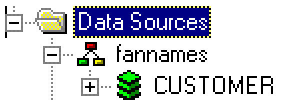
   Using the HTML/Document command in the Content group, create a new HTML file in the Domain folder.
2. In the Requests & Data Sources panel, from the New drop-down menu, select *External Request* and then select *Maintain Data: New*.
3. Name the procedure *Start*.
4. In the Requests & Data Sources panel, right-click the *Start* procedure.
5. Select *Use data sources* and select *fannames.mas* in the Data Sources dialog box.

   Note: If the fannames.mas file is not in the Data Sources dialog box, click the *New* icon and navigate to the maintain folder under the Data Servers directory.

Viewing the Structure of a Data Source in the Requests & Data Sources Panel

After you use the fannames data source in the Start procedure, its name appears in the Requests & Data Sources panel under the Data Sources folder, with a plus sign (+) next to it.

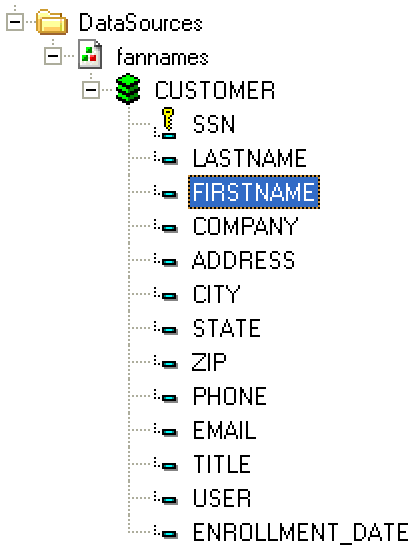
If you click the plus sign (+) next to fannames, App Studio Maintain Data displays the segments in the fannames data source, as shown in the following image.



A segment is a collection of fields that have a one-to-one relationship to each other. The fannames data source has one segment, called CUSTOMER.

Note: This data source has only one segment. Hierarchical data sources can have more than one segment. If you would like to see data sources with more than one segment, migrate the CAR data source description (one of the Information Builders standard sample files) into your folder and view it. You can then delete the CAR data source description from your folder by selecting it and clicking *Delete*.

If you click the plus sign (+) next to CUSTOMER, you see the fields in that segment, as shown in the following image.

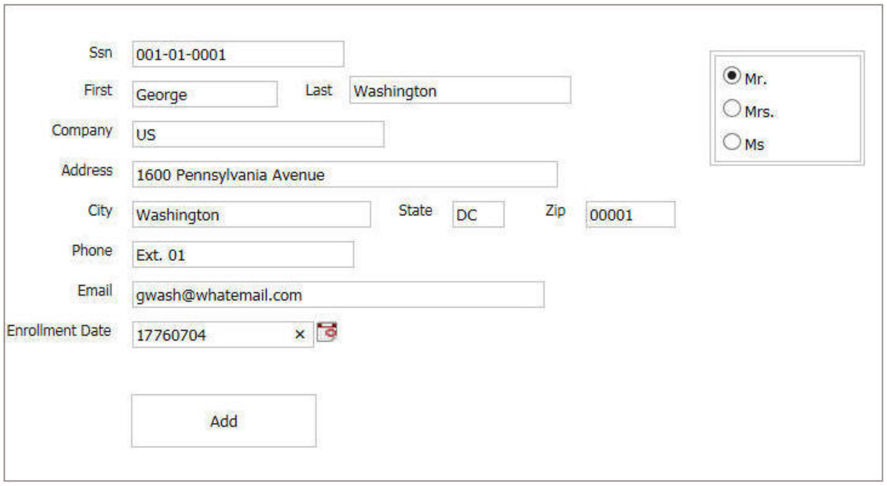


Since fannames has only one segment, these are the only fields in the fannames data source.

The key next to the SSN field indicates that SSN is a key field, which means it uniquely identifies the segment instance. In the fannames data source, each fan has a unique SSN, and no other fan should have the same number.

Designing a Form

An example of the first form that you will create for the FanClub application is shown in the following image.



Ssn	001-01-0001
First	George
Last	Washington
Company	US
Address	1600 Pennsylvania Avenue
City	Washington
State	DC
Zip	00001
Phone	Ext. 01
Email	gwash@whatemail.com
Enrollment Date	17760704

[✓ Mr.] Mrs. Ms

This form enables users to add a new fan to the data source.

The information typed into this form will be written to the fannames data source, so it should correspond to the fields in the data source. The fastest way to create the fields on the form is to use the existing data source fields. Therefore, your first task is to add data source fields to the form.

Procedure: How to Add Data Source Fields to a Form

1. In the Requests & Data Sources panel, expand *Forms* under the Start procedure.
2. Activate the Htmlpage1 tab.
3. Drag the Form1 form to the canvas.
4. Click *Yes* when prompted to create a multi-page container for better layout and presentation.
5. In the Requests & Data Sources panel, right-click *Stacks* under the Start procedure and select *New data source stack*.
6. In the Stack name field, type *AddFanStack*.
7. In the Stack Editor dialog box, expand *fannames*, then expand *CUSTOMER*, and then select *SSN* and use the arrow button to move the field to the stack column.
8. Click *OK*.

What Are Data Source Stacks?

App Studio Maintain Data procedures do not directly display or manipulate information in a data source. Instead, Maintain Data uses data source stacks as intermediaries between users and the data source.

A *stack* is a non-persistent (or in-memory) table where you can store and manipulate data from one or more data sources. App Studio Maintain Data procedures use stacks to hold values you read from the data source and to manipulate data before writing it back to the data source.

Since this is a new procedure, there are no stacks yet. You are going to create a stack named AddFanStack, instead of using the default, CustomerStk. The structure of the stack is based on the fields in the data source, in other words, the stack is going to have the columns SSN, LASTNAME, FIRSTNAME, and so on. However, this stack will be empty until you do something to put data in it.

Components of a Procedure in the Requests & Data Sources Panel

In the Requests & Data Sources panel, there have been some additions to the components of Start.

1. Switch to the HTML page tab, and in the Requests & Data Sources panel, expand *AddFanStack*.
2. Select all of the fields, except for TITLE, USER, FOCCOUNT, or FOCINDEX and, from the Requests & Data Sources panel, drag the fields to the canvas and drop them on the form.

Note: If you do not see AddFanStack, you need to force a parse by selecting *Parse* from the Maintain Data Editor context menu.

Moving Controls on a Form

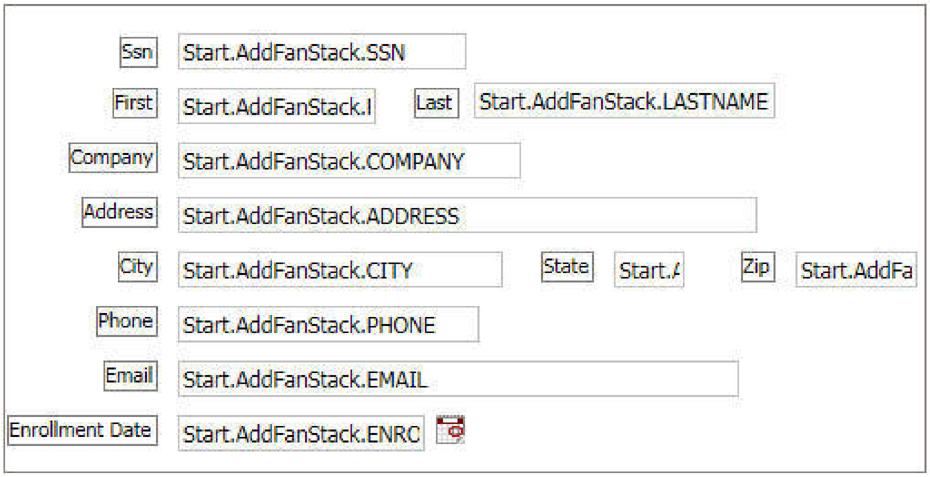
When you placed these fields on the form, App Studio Maintain Data placed them in a column. Rearrange them so that the Firstname and Lastname fields are next to each other in a row, and City, State, and Zip fields are also next to each other.

Procedure: How to Move Controls on a Form

1. Move *Lastname* (by clicking and dragging it to the new location) far enough to the right so that you can move Firstname up and to the left of it.
2. Move the *Firstname*, *Company*, *Address*, and *City* fields up.
3. Drag *State* to the right of *City*.
4. Drag *Zip* to the right of *State*.
5. Move the *Phone*, *Email*, and *Enrollment Date* fields up.

6. Select the form. On the Utilities tab, select *Tab Order* and reorder the fields to reflect the order in which they appear on your form.

In the following image, the Firstname and Lastname labels have been renamed to First and Last, respectively. You can put each of the labels in edit mode, one-by-one, and modify the text.



Ssn	Start.AddFanStack.SSN				
First	Start.AddFanStack.I	Last	Start.AddFanStack.LASTNAME		
Company	Start.AddFanStack.COMPANY				
Address	Start.AddFanStack.ADDRESS				
City	Start.AddFanStack.CITY	State	Start./	Zip	Start.AddFa
Phone	Start.AddFanStack.PHONE				
Email	Start.AddFanStack.EMAIL				
Enrollment Date	Start.AddFanStack.ENRC				

Note: You can use the Positioning tab to align objects on the form by selecting the appropriate positioning component in the Positioning group.

Saving Your Work

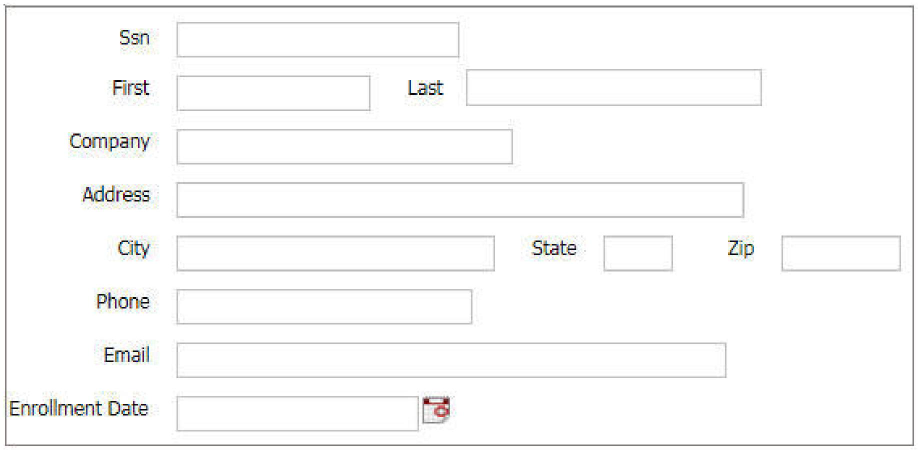
Before you go any further, save your work. From the Quick Access Toolbar, click the *Save* button.

Whenever you make changes to a component, App Studio Maintain Data places an asterisk (*) next to the component name in the Requests & Data Sources panel and in the title bar.

Running Your Page Locally

Run the page from the open canvas using the Run button on the Quick Access Toolbar to see how it looks.

App Studio Maintain Data opens the application, as shown in the following image.



This application enables you to type information into the fields displayed in the form (which places data into the data source stack AddFanStack). Next, you will improve the functionality and appearance of the application.

Using Radio Buttons

When you dragged the fields from the data source into the form, you deselected the TITLE and USER fields so they do not appear in the form.

By not copying TITLE into the form, you prevent users from typing an arbitrary title in the field. We will give them the option to select Mr., Mrs., or Ms., using a radio button control. This prevents them from typing arbitrary titles.

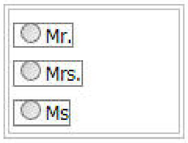
The task of using radio buttons can be separated into three steps:

1. Adding the group of radio buttons to your form.
2. Adding a tool tip text to the group of radio buttons. This step is optional, but it demonstrates how you can display useful information to your users.
3. Binding the results of the selection to a stack.

Procedure: How to Add a Group of Radio Buttons to the Form

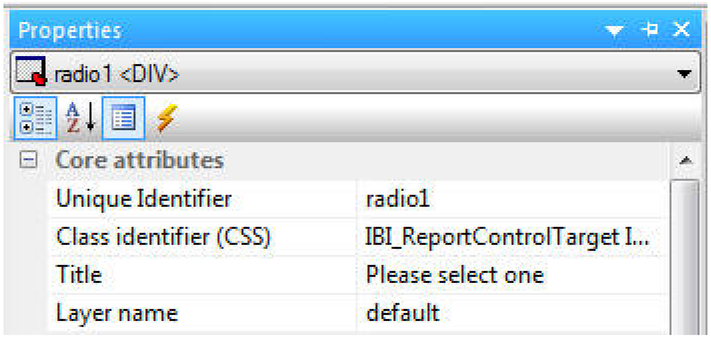
1. On the Controls tab, click the *Radio Button* control.
2. Draw a small rectangle on the form next to the SSN, First, and Last fields.
3. Select the control on the canvas and bring up the Settings panel.
4. Add the three values: Mr., Mrs., and Ms.

Your radio buttons should resemble the following image (you may need to adjust the size of your control to see all three of them).



Procedure: How to Add Tool Tip Text

1. Make sure you have selected your group of radio buttons.
2. In the Properties panel, select the *Title* property.
3. In the empty field to the right, type *Please select one*, as shown in the following image.



Core attributes

Unique Identifier	radio1
Class identifier (CSS)	IBI_ReportControlTarget I...
Title	Please select one
Layer name	default

You can move your cursor over the radio buttons at run time to see the tooltip.

Procedure: How to Bind the Results of the Selection to a Stack

1. Select your group of radio buttons.
2. Drag the TITLE field from the stack in the Requests & Data Sources panel to the Selection to input area in the Settings panel.

   If you wish, save your work and run your application again to see how it looks. Make sure to close the application before continuing the tutorial.

Stacks and Implied Columns

When you created AddFanStack using the Select Segment Fields dialog box (see *How to Add Data Source Fields to a Form* on page 16), you deselected the TITLE and USER fields. Since you deselected these fields, why are they showing up as columns in the data source stack?

These fields are showing up as columns in the data source stack because of the way App Studio Maintain Data defines data source stacks with the INFER command. When you use a field in the data source to define a column in a data source stack, App Studio Maintain Data defines columns based on the rest of the fields in that data source. If your data source is hierarchical, the rest of the fields in the segment and the key fields in any parent segment.

App Studio Maintain Data includes all of these fields in your data source stack so that when you update your data source from the data source stack, it knows the path to these fields.

These columns are called implied columns.

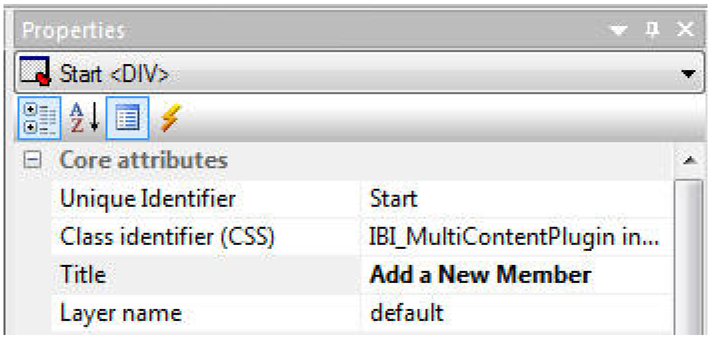
There is one field from the fannames data source that you have not placed on the form: the USER field. The user will not be entering this field. Instead, it will be generated by the application.

Giving Your Form a New Title

When you ran your application, you probably noticed that when hovering over the form with the mouse, the tooltip said Form1. You can change this so that your application displays a more useful title at run time.

Procedure: How to Give Your Form a Title

1. Deselect all controls in your form (that is, click anywhere in the form that is not a control).
2. In the Properties panel, change the *Title* property to *Add a New Member*, as shown in the following image.



Unique Identifier	Start
Class identifier (CSS)	IBI_MultiContentPlugin in...
Title	**Add a New Member**
Layer name	default

If you wish, save your work and run your application to see how it looks. Make sure to close the application before continuing the tutorial.

Writing Data to the Data Source

Your next step is to enable users to write the data that they type in this form to the data source. They do this by clicking a button that says *Add* at the bottom of the form. You add the button to your form and write the Maintain Data Language source code that inserts the data into the data source.

Procedure: How to Add a Button to Your Form

1. In the HTML canvas, on the Components tab, click *Button*.
2. Move the cursor to the bottom of your form and draw a rectangle where you want to place the button.
3. In the Properties panel, change the text in the value property for the button to *Add*. The text in the button is automatically selected when you create the button, so all you have to do is type your new text, as shown in the following image.

4. In the Properties panel, notice that the first property is the unique identifier and its value is button1. This is the internal program name of the button (how App Studio Maintain Data refers to it).

   By default, App Studio Maintain Data names forms and controls with their type and a unique number. If you are planning to refer to a form or control in other places in the procedure, giving it a more descriptive name is recommended.
5. Change the name of the button to *AddButton*. Change the unique identifier and Name properties.

Writing Functions

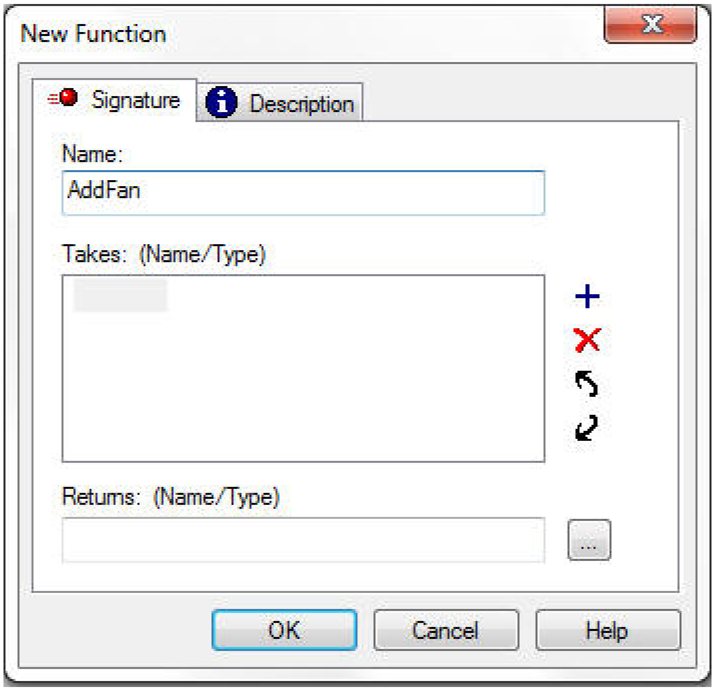
Now that you have created a button that the user will click, you must create the code to run when the user clicks the button.

You put this code in a function. A function is a series of commands in a procedure grouped together as a unit of control. A function accomplishes a task, such as calculating values, extracting data from a data source to place in a data source stack, or writing information to a data source.

Procedure: How to Write a Function

1. In the Requests & Data Sources panel, select the *Start* procedure.
2. Right-click *Start*.

3. In the shortcut menu, click *New function*.
4. In the New Function dialog box, give your function the name *AddFan*, as shown in the following image.



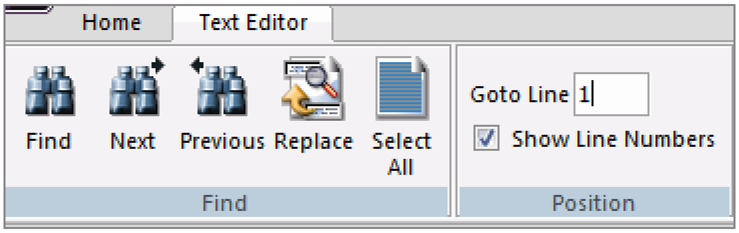
Name: AddFan

5. Click *OK*.

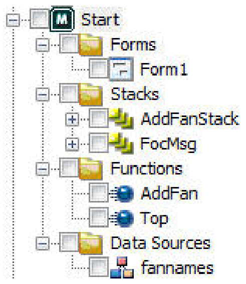
App Studio Maintain Data opens the source code for your function in the Maintain Data Editor.

About the Maintain Data Editor

Underlying many of the graphical elements of App Studio Maintain Data is Maintain Data language source code. The Maintain Data Text Editor enables you to create, view, and edit the source code for procedures, procedure components, and other types of files required by your Maintain Data applications. The Maintain Data Text Editor tab opens when you create a new function or edit an existing one. It contains two groups, Find and Position. The Maintain Data Text Editor tab is shown in the following image.



All App Studio Maintain Data procedures start with the keyword MAINTAIN (which must be in uppercase) and end with the keyword END (also in uppercase), as shown in the following image. When you create a procedure, App Studio Maintain Data includes these two keywords automatically.



```
MAINTAIN FILE fannames

$$Declarations

Case Top
Infer fannames.CUSTOMER.SSN into AddFanStack;
Winform Show Form1;
EndCase

Case AddFan
EndCase

END
```

Note: The text in your window may wrap differently.

FILE fannames tells App Studio Maintain Data the data sources this procedure is going to access. In the list of components, there is a Data Sources folder with a fannames data source below it.

Data source names (up to 15) follow the MAINTAIN FILE command and are separated by the word AND.

Following this line is a comment line beginning with $$. You can tell that this is a comment since green (the default) is the color for comments. This comment line is automatically generated when you create a procedure.

This particular comment, $$Declarations, is generated automatically by App Studio Maintain Data when you created the procedure. If you create any new variables using the Request & Data Sources panel, App Studio Maintain Data places the source code after this comment.

Case Top begins the definition of the Top function. This definition takes up several lines and ends with the keyword EndCase. The first statement, which begins with Infer, defines the AddFanStack data source stack.

The next line of the Top function, Winform Show Form1; is the code that displays Form1 at run time. This code was generated automatically when you created the procedure.

Finally, the Case AddFan and EndCase define an additional function in this procedure.

Finding or Replacing Text Using the Find Group

The Find group contains options that enable you to find or replace text.

The commands in the Find group are:

Find

Finds the specified text. You can also press Ctrl+F to activate the Find dialog box.

Next

Finds the next instance of the specified text.

Previous

Finds the previous instance of the specified text.

Replace

Replaces the specified text with different text.

Select All

Selects all of the text in the procedure.

Reference: Find Dialog Box

You can use the options on the Find dialog box to indicate how to search for information.

The options on the Find dialog box are:

Find What

Provides a text box where you can specify the text that you want to find.

Match case

Select this option to match the uppercase or lowercase value, as specified in the Find what field.

Match whole words only

Select this option to match the whole word only.

Find Next

Allows you to find the next instance of your search term.

Reference: Replace Dialog Box

You use the options on the Replace dialog box to indicate how to use the find feature to find and replace information.

The options on the Replace dialog box are:

Find What

Provides a text box where you can specify the text that you want to find.

Replace With

Provides a text box where you can specify the text that is going to replace the text for which you are searching.

Match case

Select this option to match the uppercase or lowercase value, as specified in the Find what field.

Match whole words only

Select this option to match the whole word only.

Find Next

Allows you to find the next instance of your search term.

Replace

Replaces the search information that you specified in the Find What field with the text or other information that you indicated in the Replace With field.

Placing the Cursor Using the Position Group

The Position group allows you to place your cursor at the desired line and allows you to turn off line numbers.

The commands in the Position group are:

Goto Line

Displays the current line your cursor is on. You can type a different line number into the text box to place your cursor on that line.

Show Line Numbers

When selected, displays line numbers. This option is selected by default.

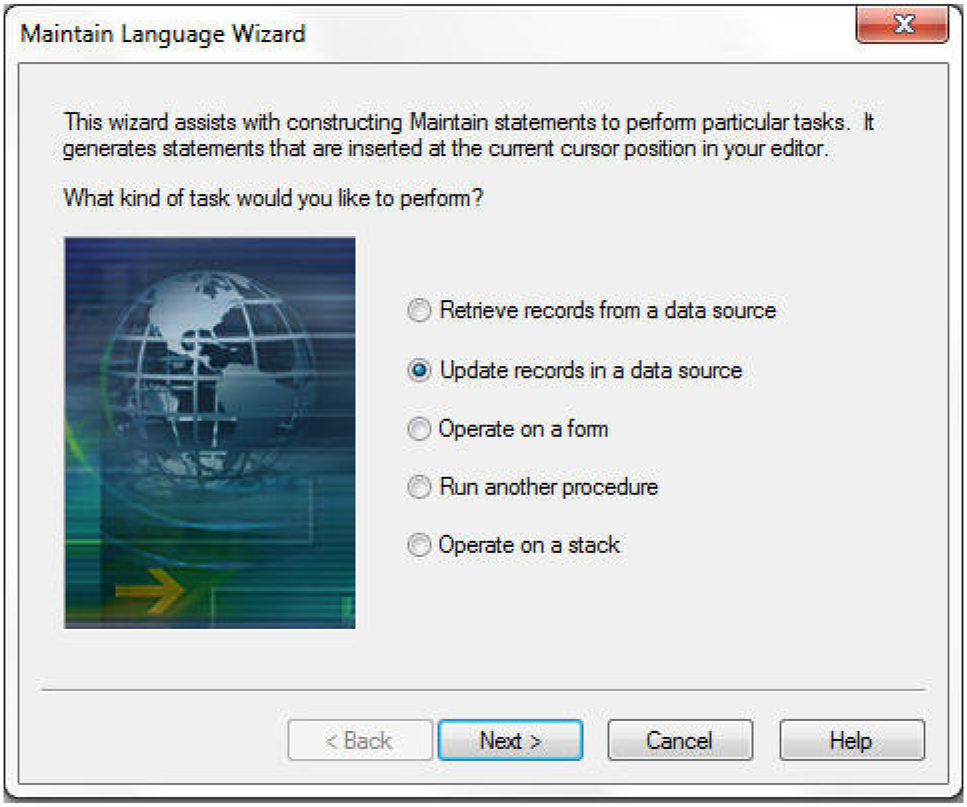
Procedure: How to Build Maintain Data Language Code Using the Language Wizard

1. Ensure that your cursor is in the line between Case AddFan and EndCase.
2. Right-click in the Maintain Data Editor.
3. From the shortcut menu, select *Language Wizard*.

   The Maintain Language Wizard opens. The Maintain Language Wizard helps you build Maintain Data language source code without typing the syntax yourself.

   The first Language Wizard window asks you to specify, in general, what kind of task you want to accomplish.

4. Select *Update records in a data source*, as shown in the following image.



Maintain Language Wizard

This wizard assists with constructing Maintain statements to perform particular tasks. It generates statements that are inserted at the current cursor position in your editor.

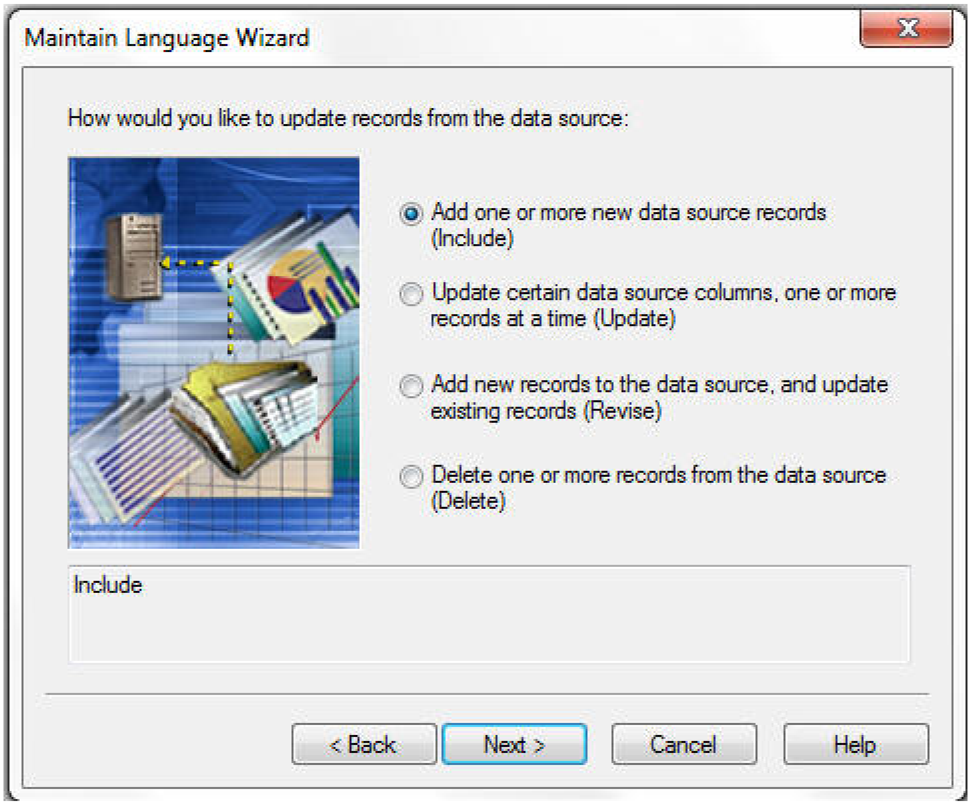
What kind of task would you like to perform?

- ( ) Retrieve records from a data source
- [✓ Update records in a data source]
- ( ) Operate on a form
- ( ) Run another procedure
- ( ) Operate on a stack

5. Click *Next*.

   Now that you have specified the general task you want to perform, the Language Wizard narrows the task further. Notice that after each task, there is a word in parentheses. This is the name of the Maintain Data language command that executes that task. Additionally, notice the box at the bottom of the window contains the Maintain Data language code being generated by the Language Wizard. As you move through the Language Wizard, you see more code.

6. Select *Add one or more new data source records (Include)*, as shown in the following image.



Maintain Language Wizard

How would you like to update records from the data source:

(●) Add one or more new data source records (Include)

( ) Update certain data source columns, one or more records at a time (Update)

( ) Add new records to the data source, and update existing records (Revise)

( ) Delete one or more records from the data source (Delete)

Include

7. Click *Next*.

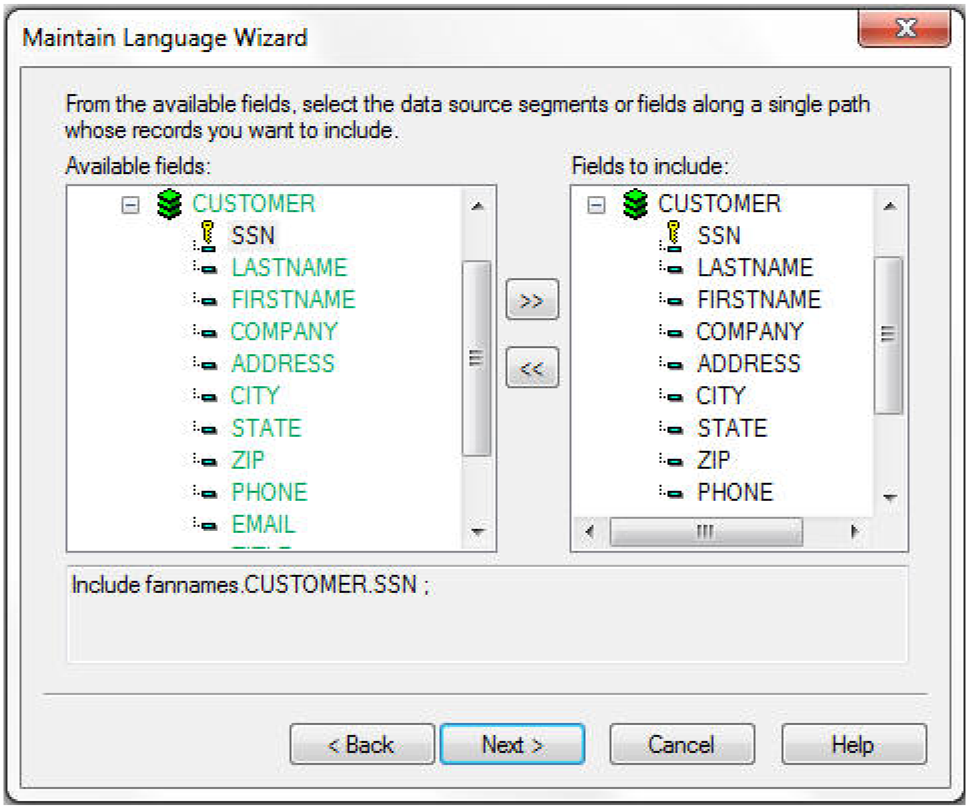
   Now that you have specified which command to use, the Language Wizard asks you to supply the parameters for that command. In this case, you must tell it which data source is being updated and from where.

   You first specify which data source is being updated.

   Note: The Available fields list contains the data sources that you are using in this procedure, not the list of data sources in the folder.

8. Expand the *fannames* data source.
9. Expand the *CUSTOMER* segment.
10. Copy the SSN field into the Fields to include box by clicking *SSN* and then clicking the right arrow, or by double-clicking *SSN*.

Notice that, as with stacks, all the other fields in the CUSTOMER segment are also copied, as shown in the following image. This is because the Maintain Data language assumes that if you are adding new data source records to a data source, you want to write information into all the fields in a segment. For more information, see *Stacks and Implied Columns* on page 20.



Include fannames.CUSTOMER.SSN ;

Notice that the Maintain Data language box at the bottom now reads as follows:

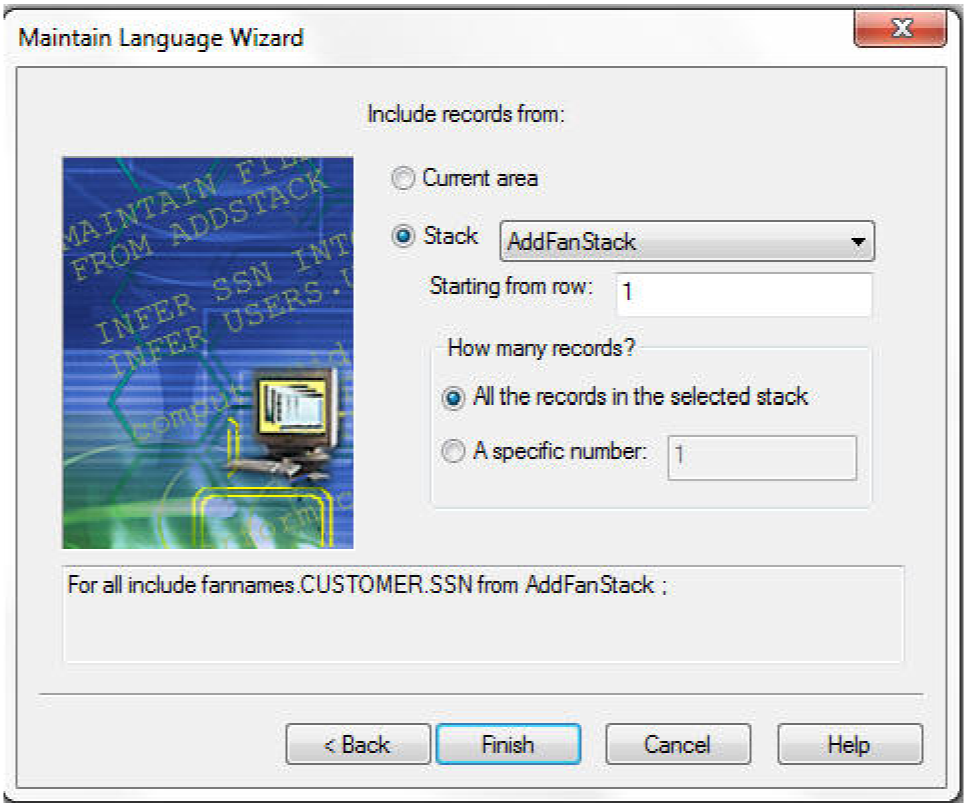
```
Include fannames.CUSTOMER.SSN;
```

11. Click *Next*.

    Your final step is to indicate where this data is being written from, which in this case, is AddFanStack.

12. Select *Stack*.
13. Choose *AddFanStack* from the list.

14. Leave the 1 in the Starting from row field and select the *All the records in the selected stack option*, as shown in the following image.



Stack: AddFanStack [✓ Stack]
Starting from row: 1
[✓ All the records in the selected stack]
A specific number: 1

For all include fannames.CUSTOMER.SSN from AddFanStack ;

Notice that the Maintain Data language box at the bottom now reads as follows:

```
For all include fannames.CUSTOMER.SSN from AddFanStack;
```

15. Click *Finish*.

App Studio Maintain Data places the source code that the Language Wizard generated in between the Case AddFan and EndCase lines.

Clearing Data From Stacks

Now that you have included the data from AddFanStack into the fannames data source, it is a good idea to clear the data from AddFanStack. Use the Language Wizard to write this code.

Procedure: How to Clear the Data From a Stack Using the Language Wizard

1. Place the insertion point after

```
For all include fannames.CUSTOMER.SSN from AddFanStack;
```

but before

```
EndCase
```

2. Right-click in the Maintain Data Editor and, from the shortcut menu, click *Language Wizard*.
3. Select *Operate on a stack* and click *Next*.
4. When the Language Wizard asks you which stack operation you would like to perform, select *Clear the contents of a stack* and click *Next*.

   The Language Wizard asks you to select one or more stacks to clear.
5. Select *AddFanStack* and click *Finish*. The syntax should read as follows:

   ```
   Stack clear AddFanStack;
   ```

Assigning the Function to the Add Button

Now that you have written the code that inserts user data into the data source, you need to designate that when the user clicks the *Add* button, this function is performed. You do this using the Tasks & Animations panel.

Procedure: How to Assign a Function to an Event

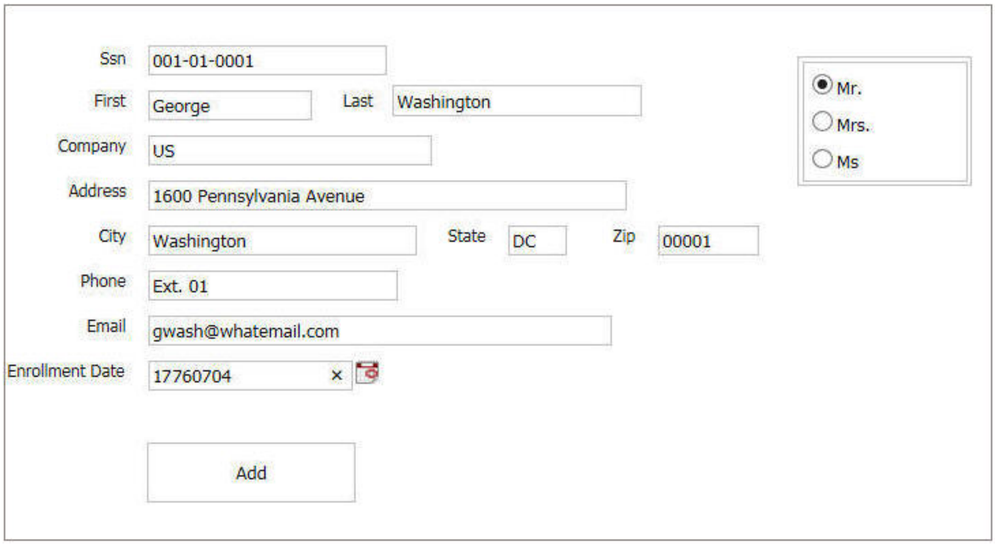
An event is something that a user performs, such as clicking a button or moving to a field. Events are done as tasks with Ajax calls. The target type and request is *mntname.case*.

1. In the Tasks & Animations panel, click *New* to create a new task. The Trigger Type is Click. The Trigger Identifier is AddButton.
2. In the Requests/Actions section, select *Run Request*, then select *Start*, and then select *Start.AddFan*.

   This will make the Add button call that case (function).

   Note: You need to have Start.Connect in the load task or this will not work.
3. Run your application to see how it looks.

4. Add your name to the fannames data source. You are now included in the App Studio FanClub application, as shown in the following image.



Ssn	001-01-0001				
First	George	Last	Washington		
Company	US				
Address	1600 Pennsylvania Avenue				
City	Washington	State	DC	Zip	00001
Phone	Ext. 01				
Email	gwash@whatemail.com				
Enrollment Date	17760704				

● Mr. ○ Mrs. ○ Ms

5. Close the application before continuing the tutorial.

Adding a Form to Display Data From a Data Source

The FanClub application can add names to the fannames data source, but users do not get much visual feedback from this task. The FanClub application needs to display the contents of the fannames data source.

This task is divided into the following steps:

1. Add a new form to the application.
2. Build the code to extract the data from the fannames data source using the Language Wizard.
3. Design the form to display the contents of fannames.
4. Create a link from Form1 to the new form.

Procedure: How to Add a New Form to Your Application

To create a new form, in the Maintain Data Editor, type *Winform Show formname* when you have the case created.

1. Create a case OnShowFanButton_Click and type the following:

```
Winform Show ShowFan;
EndCase
```

2. From the HTML canvas, right-click the outer frame of the form and select *Add Page*.
3. Right-click again and select *Show all pages*.
4. Drag ShowFan from the Requests & Data Sources panel to the new page container.
5. In the Properties panel, change the title of the form from ShowFan to *Show Fan Club Members*.

Extracting Data From a Data Source Into a Stack

Your next step is to create a function named GetFans, which extracts all the information in the fannames data source and places it into a stack named GetFanStack. Try this yourself or see the next section for instructions.

Tip: Create the function and then use the Language Wizard to generate the code.

Procedure: How to Extract Data From the Fannames Data Source Into a Stack

1. In the Requests & Data Sources panel, right-click the *Start* procedure and then click *New function*.
2. In the New Function dialog box, name your function *GetFans* and click *OK*.
3. Make sure your insertion point is placed after the statement

   ```
   Case GetFans
   ```

   but before

   ```
   EndCase
   ```

4. Right-click in the Maintain Data Editor and, from the shortcut menu, click *Language Wizard*.

   The Language Wizard opens.

5. Select *Retrieve records from a data source* and click *Next*.
6. Select *Starting from the current record position (Next)* and click *Next*.

   The Maintain Data language contains two commands to retrieve data from a *data source*. This window determines which one you want to use.

7. Select the *data source* segments or fields whose records you want to retrieve. In the *Available fields* box, expand the *fannames data source*, expand the *CUSTOMER* segment, move the *SSN* field to the *Fields to retrieve* box and click *Next*.

   This window is where the Language Wizard determines from which *data source* you are reading the data. For more information, see *Stacks and Implied Columns* on page 20.

8. Select *All the records in the selected segment* and make sure *Change the current data source position to the top* is selected and click *Next*.

This ensures that App Studio Maintain Data starts from the beginning of the *data source* when retrieving records.

9. Type *GetFanStack* in the text box to create a new stack, and make sure that *Clear the stack first* is selected. You can leave the default value 1 in the *Place the records into the stack* field. Click *Next*.

10. Leave the next window blank since you want to retrieve all records from the data source. Click *Finish*.

    The Language Wizard should generate the following code:

```
Reposition fannames.CUSTOMER.SSN;
Stack clear GetFanStack;
For all next fannames.CUSTOMER.SSN into GetFanStack;
```

Adding an HTML Table to Your Form

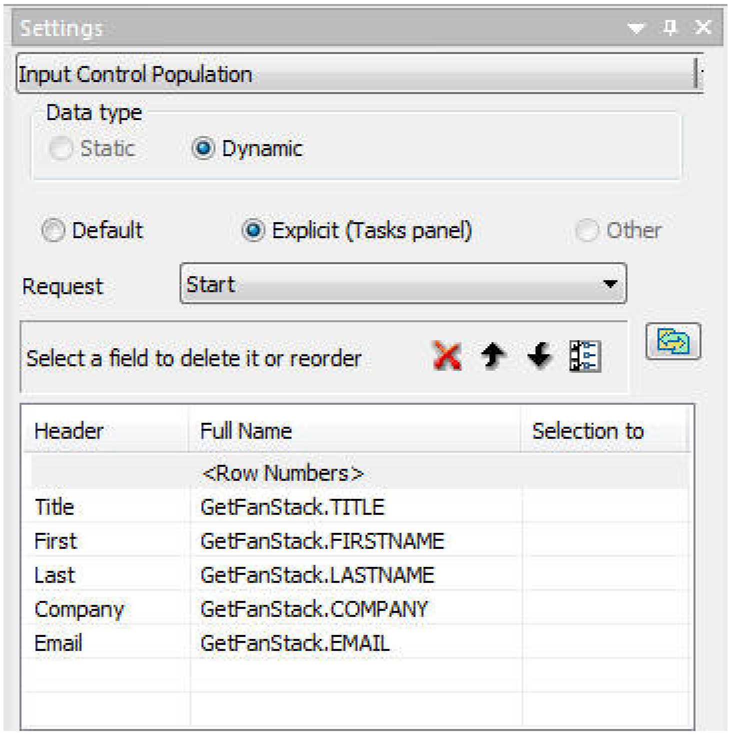
You are going to display the fans from the fannames data source using an HTML table. An HTML table displays the contents of a data source stack in a read-only grid.

Another option for displaying the fans from the fannames data source is to create a report procedure in App Studio and execute this code whenever you open this form.

Procedure: How to Add an HTML Table to Your Form

1. On the Components tab, click the *HTML Table* button.
2. Draw a rectangle on the form where you want your HTML table to go on your new ShowFan form.
3. On the Settings panel, be sure that the Explicit (Requests panel) radio button is selected.
4. From GetFanStack in the Requests & Data Sources panel, drag the *LASTNAME*, *FIRSTNAME*, *COMPANY*, *EMAIL*, and *TITLE* fields into the Table Columns list on the Settings panel.

Use the *Move up* and *Move down* buttons to rearrange these fields so that they are in the following order: TITLE, FIRSTNAME, LASTNAME, COMPANY, and EMAIL, as shown in the following image.

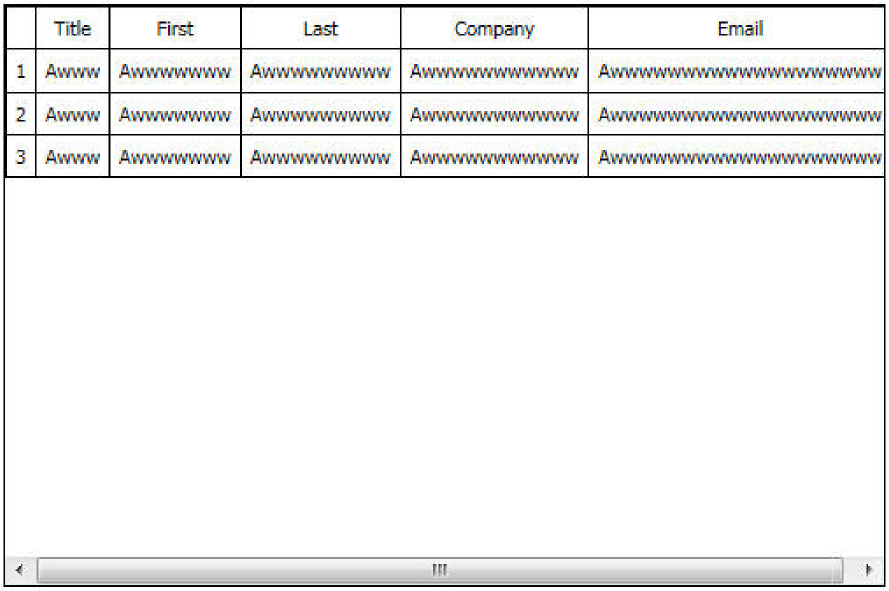


Header	Full Name	Selection to
	<Row Numbers>	
Title	GetFanStack.TITLE	
First	GetFanStack.FIRSTNAME	
Last	GetFanStack.LASTNAME	
Company	GetFanStack.COMPANY	
Email	GetFanStack.EMAIL	

You can change the appearance of any of these columns. For example, suppose you want to change the header titles for the FIRSTNAME and LASTNAME fields so that they read First and Last.

5. Click *Firstname* and change Firstname to *First*.
6. Repeat the process to change Lastname to *Last*.

Your form resembles the following image.



	Title	First	Last	Company	Email
1	Awww	Awwwwwwww	Awwwwwwwwww	Awwwwwwwwwwww	Awwwwwwwwwwwwwwwwwwwwwww
2	Awww	Awwwwwwww	Awwwwwwwwww	Awwwwwwwwwwww	Awwwwwwwwwwwwwwwwwwwwwww
3	Awww	Awwwwwwww	Awwwwwwwwww	Awwwwwwwwwwww	Awwwwwwwwwwwwwwwwwwwwwww

Creating a Link From One Form to Another

Your final step in creating your new form ShowFan is providing a way to open it from Form1. Do this by adding a button to Form1 that runs the GetFans function and opens the ShowFan form.

Procedure: How to Link From One Form to Another

1. Make Form1 the active window.
2. On the Components tab, click *Button*.
3. Draw a rectangle to the right of the Add button on Form1.
4. In the Properties panel, change the name and unique identifier for the button to *ShowFanButton* and change the value to *Show Fans*.
5. Create a new task.
6. In the Tasks & Animations panel, select *Click* for the Trigger Type and select *ShowFanButton* for the Trigger Identifier.
7. In the Tasks & Animations panel, from the Requests/Actions section, select *Run Request*, select *Start*, and then select *Start.OnShowFanButton_Click*.

   In the Maintain Data Editor, type Perform GetFans( ); in the case. The syntax should read as follows:

```
Case OnShowFanButton_Click
Perform GetFans( );
Winform Show ShowFan;
EndCase
```

8. Run your application to see how it looks.
9. Close the application before continuing the tutorial.

Adding Form Navigation Buttons

When you ran your application and displayed the Show Fan Club Members form, you may have noticed that the only thing you can do in this window is close the form by closing the browser.

Add some navigation buttons to the FanClub application:

- ❑ Add a Back button to the Show Fan Club Members form so that you can go back to the Add a New Member form.
- ❑ Add an Exit button to the Add a New Member form.

Procedure: How to Add a Back Button

1. Open the *ShowFan* form.
2. On the Components tab, click *Button*.
3. Draw a button under the HTML table.
4. Change the text on the button to *Back*.
5. Change the name and the unique identifier of the button to *BackButton*.
6. Create a new case called *OnBackButton_Click* and type *Winform Show Form1;* in it.
7. Add a task to call the case.
8. In the Tasks & Animations panel, select *Click* for the Trigger Type and select *BackButton* for the Trigger Identifier.
9. In the Tasks & Animations panel, from the Requests/Actions section, select *Run Request*, then select *Start*, and then select *Start.OnBackButton_Click*.

Procedure: How to Add an Exit Button

1. Open Form1.
2. Place an Exit button named *ExitButton* at the bottom of your form.
3. Create a new task for the click of the Exit button to call the disconnect request from a task.

4. In the Tasks & Animations panel, select *Click* for the Trigger Type and select *ExitButton* for the Trigger Identifier.
5. In the Tasks & Animations panel, from the Requests/Actions section, select *Run Request*, then select *Start*, and then select *Start.Disconnect*.
6. Run your application to see how it looks.
7. Click the *Exit* button to close the application.

Adding Images to Your Application

You can use images to improve the appearance and usability of App Studio Maintain Data applications. This section explains how to:

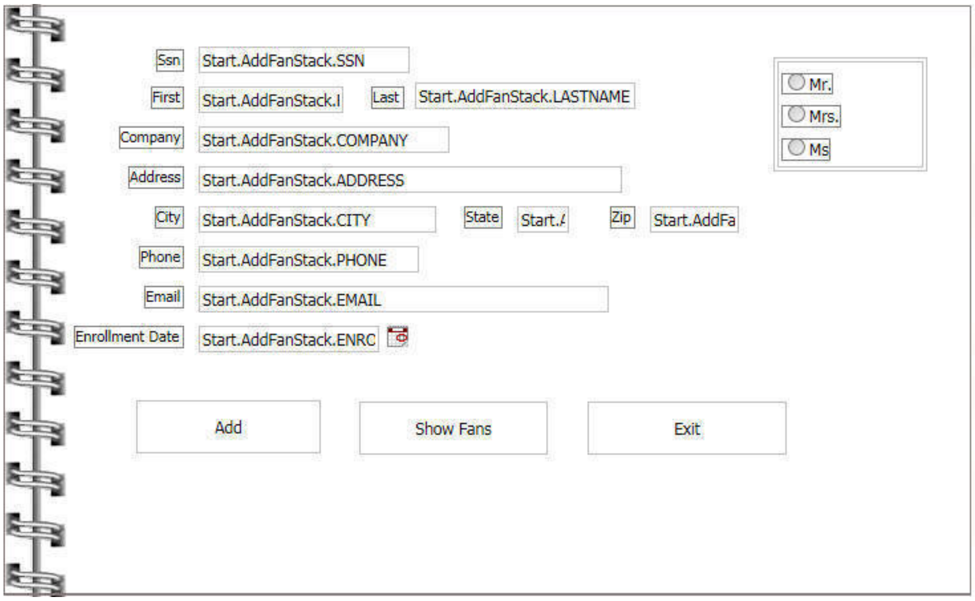
- ❑ Add an image to the background of your two forms. This image, spiral.gif, makes your forms look like pages in a spiral notebook.
- ❑ Add titles to your two forms.
- ❑ Add a fan graphic to your form.

Procedure: How to Add a New Background Image to Your Form

1. Open Form1.
2. Make sure all of the controls are deselected so that you can see the properties for the form.
3. In the Properties panel for the form, locate the *Background-image* property in the Background category.
4. Click the mouse in the input area to the right to display the ellipsis button.
5. Click the ellipsis button, browse to Data Servers, EDASERVE, Application, maintain, images, and select *spiral.gif*.

   Note: Spiral.gif is one of the sample Tutorial files that was placed on your hard drive at installation. It should be located in \ibi\apps\Maintain\images.
6. Use the Background-repeat property to set it to repeat-y. This makes the image repeat for the entire height of the form.

Your form resembles the following image.



Ssn Start.AddFanStack.SSN
First Start.AddFanStack.I Last Start.AddFanStack.LASTNAME
Company Start.AddFanStack.COMPANY
Address Start.AddFanStack.ADDRESS
City Start.AddFanStack.CITY State Start.A Zip Start.AddFa
Phone Start.AddFanStack.PHONE
Email Start.AddFanStack.EMAIL
Enrollment Date Start.AddFanStack.ENRC

You may need to move the controls on your form to the right so that they are not overlapping the spiral.

Procedure: How to Add an Existing Background Image to Your Form

1. Add the spiral image to the ShowFan form.
2. Run your application to see how it looks.
3. Click the *Exit* button to close the application before continuing the tutorial.

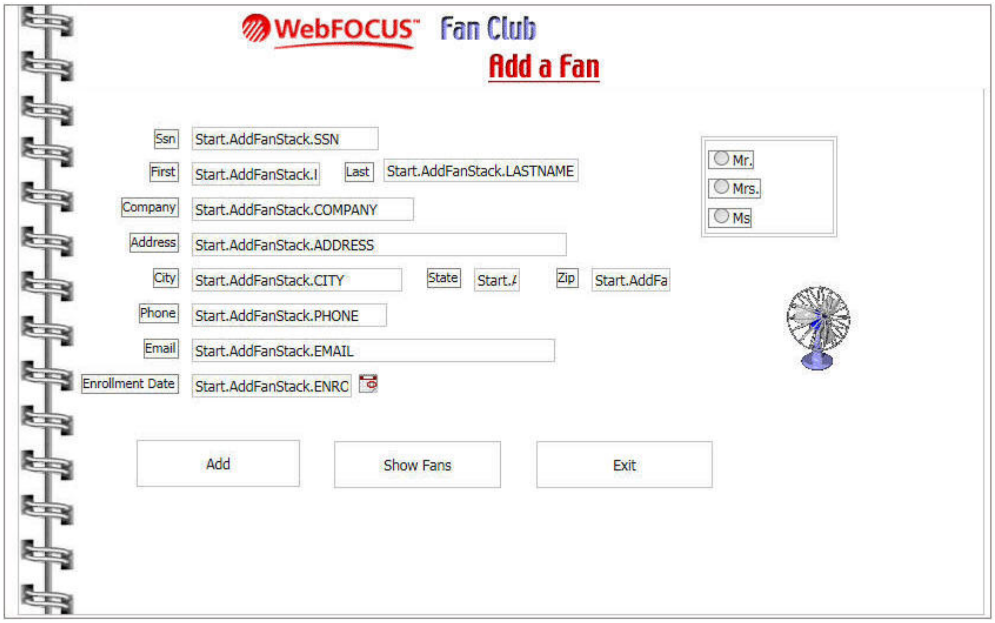
Procedure: How to Add an Image to Your Form

1. Open *Form1*.
2. Select all the controls on your form by pressing Ctrl+A and move them down so that you have roughly one inch at the top of your form.
3. On the Components tab, select the *Image* control.
4. Draw a box in the empty space at the top of the form.

   The HTML canvas opens the Open File dialog box.
5. Repeat the steps you followed in *How to Add a New Background Image to Your Form* on page 39 to add Addafan.gif to your form.

6. Add the fan.gif image to your form to the right of the entry boxes. (Repeat steps 3 through 5.)

   Your form resembles the following image.



Ssn	Start.AddFanStack.SSN
First	Start.AddFanStack.I
Last	Start.AddFanStack.LASTNAME
Company	Start.AddFanStack.COMPANY
Address	Start.AddFanStack.ADDRESS
City	Start.AddFanStack.CITY
State	Start.A
Zip	Start.AddFa
Phone	Start.AddFanStack.PHONE
Email	Start.AddFanStack.EMAIL
Enrollment Date	Start.AddFanStack.ENRC

7. Open ShowFan and add the image currfan.gif. (Repeat steps 3 through 5.)
8. Run your application to see how it looks.
9. When you are done, exit your application by clicking *Exit*.

Chapter 3

App Studio Maintain Data Concepts

The following topics provide an overview of basic App Studio Maintain Data concepts. To fully exploit the potential and productivity of App Studio Maintain Data, you should become familiar with concepts including:

- Processing data in sets by using stacks.
- Controlling the flow of an App Studio Maintain Data application and the procedures within it.
- Developing presentation logic using forms and s.
- Reading from and writing to data sources.
- Ensuring transaction integrity.
- Creating classes and objects.

In this chapter:

Set-based Processing

Maintain Data provides the power of set-based processing, enabling you to read, manipulate, and write groups of records at a time. You manipulate these sets of data using a data structure called a *data source stack*.

A data source stack is a simple temporary table. Generally, columns in a data source stack correspond to data source fields, and rows correspond to records, or path instances, in that data source. You can also create your own user-defined columns.

The intersection of a row and a column is called a cell and corresponds to an individual field value. The data source stack itself represents a data source path.

For example, consider the following Maintain Data command:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
    WHERE Employee.Emp_ID EQ SelectedEmpID;
```

This command retrieves Emp_ID and the other root segment fields, as well as the Pay_Date, Gross, Ded_Code, and Ded_Amt fields from the Employee data source and holds them in a data source stack named PayStack. Because the command specifies FOR ALL, it retrieves all of the records at the same time. You do not need to repeat the command in a loop. Because it specifies WHERE, it retrieves only the records you need, in this case, the payment records for the currently-selected employee.

You could just as easily limit the retrieval to a sequence of data source records, such as the first six payment records that satisfy your selection condition

```
FOR 6 NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
    WHERE Employee.Emp_ID EQ SelectedEmpID;
```

or even restrict the retrieval to employees in the MIS department earning salaries above a certain amount:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
    WHERE (Employee.Department EQ 'MIS') AND
          (Employee.Curr_Sal GT 23000);
```

Which Processes Are Set-based?

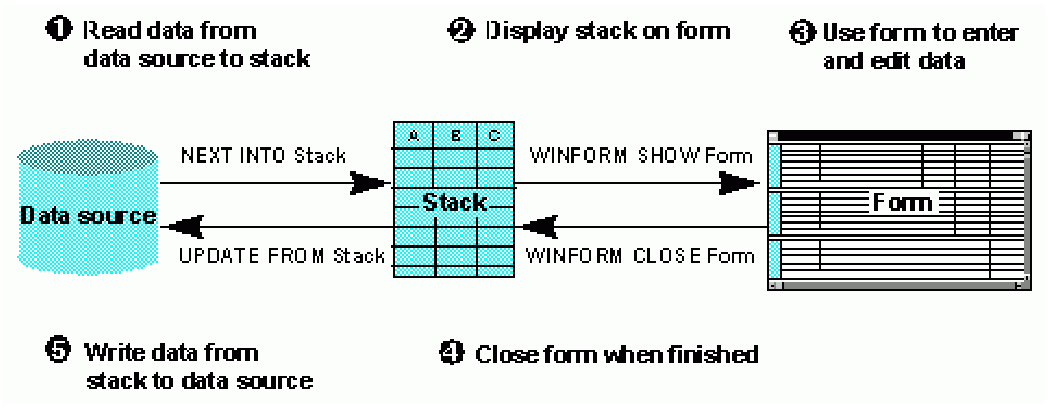
You can use set-based processing for the following types of operations:

- **Selecting records.** You can select a group of data source records at one time using the NEXT command with the FOR prefix. Maintain Data retrieves all of the data source records that satisfy the conditions you specified in the command and then automatically puts them into the data source stack that you specified.

- **Collecting transaction values.** You can use forms to display, edit, and enter values for groups of rows. The rows are retrieved from a data source stack, displayed in the form, and are placed back into a stack when the user is finished. You can also use the NEXT command to read values from a transaction file into a stack.

- **Writing transactions to the data source.** You can include, update, or delete a group of records at one time using the INCLUDE, UPDATE, REVISE, or DELETE commands with the FOR prefix. The records come from the data source stack that you specify in the command.
- **Manipulating stacks.** You can copy a set of records from one data source stack to another and sort the records within a stack.

The following diagram illustrates how these operations function together in a procedure:


❶ Read data from data source to stack
❷ Display stack on form
❸ Use form to enter and edit data
Data source
NEXT INTO Stack
UPDATE FROM Stack
A
B
C
Stack
WINFORM SHOW Form
WINFORM CLOSE Form
Form
❺ Write data from stack to data source
❹ Close form when finished

The diagram is explained in detail below:

1. The procedure selects several records from the data source and, for each record, copies the values for fields A, B, and C into the data source stack. It accomplishes this using the NEXT command.
2. The procedure displays a form on the screen. The form shows multiple instances of fields A, B, and C. The field values shown on the screen are taken from the stack. This is accomplished using a form and the Winform Show command.
3. The procedure user views the form and enters and edits data. As the form responds to the activity of the user, it automatically communicates with the procedure and updates the stack with the new data.
4. The procedure user clicks a button to exit the form. The button accomplishes this by triggering the Winform Close command.
5. The procedure writes the values for fields A, B, and C from the stack to the selected records in the data source. The procedure accomplishes this using the UPDATE command.

How Does Maintain Data Process Data in Sets?

Maintain Data processes data in sets using two features:

- ❑ **The command prefix FOR.** When you specify FOR at the beginning of the NEXT, INCLUDE, UPDATE, REVISE, and DELETE commands, the command works on a group of records, instead of on just one record.
- ❑ **Stacks.** You use a data source stack to hold the data from a group of data source or transaction records. For example, a stack can hold the set of records that are output from one command (such as NEXT or Winform) and provide them as input to another command (such as UPDATE). This enables you to manipulate the data as a group.

Creating and Defining Data Source Stacks: An Overview

Maintain Data makes working with stacks easy by enabling you to create and define a data source stack dynamically, simply by using it. For example, when you specify a particular stack as the destination stack for a data source retrieval operation, that stack is defined as including all of the fields in all of the segments referred to by the command. Consider the following NEXT command, which retrieves data from the VideoTrk data source into the stack named VideoTapeStack:

```
FOR ALL NEXT CustID INTO VideoTapeStack;
```

Because the command refers to the CustID field in the Cust segment, all of the fields in the Cust segment (from CustID through Zip) are included as columns in the stack. Every record retrieved from the data source is written as a row in the stack.

Example: Creating and Populating a Simple Data Source Stack

If you are working with the VideoTrk data source, and you want to create a data source stack containing the ID and name of all customers whose membership expired after June 24, 1992, you could issue the following NEXT command:

```
FOR ALL NEXT CustID INTO CustNames WHERE ExpDate GT 920624;
```

The command does the following:

1. Selects (NEXT) all VideoTrk records (FOR ALL) that satisfy the membership condition (WHERE).
2. Copies all of the fields from the Cust segment (referenced by the CustID field) from the selected data source records into the CustNames stack (INTO).

The resulting CustNames stack looks like this (some intervening columns have been omitted to save space):

CustID	LastName	...	Zip
0925	CRUZ	...	61601
1118	WILSON	...	61601
1423	MONROE	...	61601
2282	MONROE	...	61601
4862	SPIVEY	...	61601
8771	GARCIA	...	61601
8783	GREEN	...	61601
9022	CHANG	...	61601

Creating a Data Source Stack

You create a data source stack:

- ❑ Implicitly, by specifying it in a NEXT or MATCH command as the destination (INTO) stack, or by associating it in the HTML canvas.

  Forms are introduced in *Forms and Event-driven Processing* on page 64. The HTML canvas is used to design and create forms.

- ❑ Explicitly, by declaring it in an INFER command.

For example, this NEXT command creates the EmpAddress stack:

```
FOR ALL NEXT StreetNo INTO EmpAddress;
```

Defining Data Source Columns in a Data Source Stack

When you define a data source stack, you can include any field along a data source path. Maintain Data defines the data source columns of a stack by performing the following steps:

1. Scanning the procedure to identify all the NEXT, MATCH, and INFER commands that use the stack as a destination and all the controls that use the stack as a source or destination.

2. Identifying the data source fields that these commands move in or out of the stack:
    - ❑ **NEXT commands.** Moves the fields in the data source field list and WHERE phrase.
    - ❑ **MATCH commands.** Moves the fields in the data source field list.
    - ❑ **INFER commands.** Moves all the fields specified by the command.
3. Identifying the data source path that contains these fields.
4. Defining the stack to include columns corresponding to all the fields in this path.
    - ❑ **NEXT commands.** Moves the fields in the data source field list and WHERE phrase.
    - ❑ **MATCH commands.** Moves the fields in the data source field list.
    - ❑ **INFER commands.** Moves all the fields specified by the command.

You can include any number of segments in a stack, as long as they all come from the same path. When determining a path, unique segments are interpreted as part of the parent segment. The path can extend through several data sources that have been joined together. Maintain Data supports joins that are defined in a Master File. For information about defining joins in a Master File, see the *Describing Data With WebFOCUS Language* manual. (Maintain Data can read from joined data sources, but cannot write to them.)

The highest specified segment is known as the anchor and the lowest specified segment is known as the target. Maintain Data creates the stack with all of the segments needed to trace the path from the root segment to the target segment:

- ❑ It automatically includes all fields from all of the segments in the path that begins with the anchor and continues to the target.
- ❑ If the anchor is not the root segment, it automatically includes the key fields from the anchor's ancestor segments.

Example: Defining Data Source Columns in a Data Source Stack

In the following source code, a NEXT command refers to a field (Last_Name) in the EmpInfo segment of the Employee data source, and reads that data into EmpStack. Another NEXT command refers to a field (Salary) in the PayInfo segment of Employee and also reads that data into EmpStack.

```
NEXT Last_Name INTO EmpStack;
.
.
.
FOR ALL NEXT Salary INTO EmpStack;
```

Based on these two NEXT commands, Maintain Data defines a stack named EmpStack, and defines it as having columns corresponding to all of the fields in the EmpInfo and PayInfo segments:

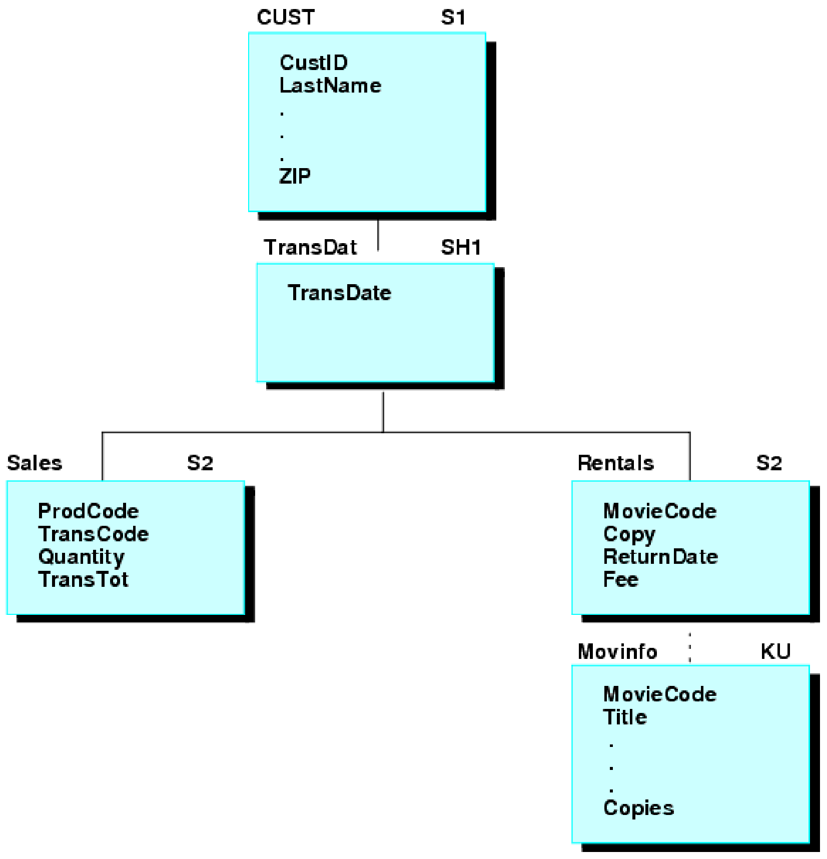
Emp_ID	Last_Name	...	Ed_Hrs	Dat_Inc	...	Salary	JobCode
071382660	STEVENS	...	25.00	82/01/01	...	$11,000.00	A07
071382660	STEVENS	...	25.00	81/01/01	...	$10,000.00	A07

Example: Establishing a Path Using Keys and Anchor and Target Segments

The following code populates CustMovies, a data source stack that contains video rental information for a given customer. The first NEXT command identifies the customer. The second NEXT command selects a field (TransDate) from the second segment and a field (Title) from the bottom segment of a path that runs through the joined VideoTrk and Movies data sources:

```
NEXT CustID WHERE CustID IS '7173';
FOR ALL NEXT TransDate Title INTO CustMovies
   WHERE Category IS 'COMEDY';
```

The structure of the joined VideoTrk and Movies data sources looks like this:



In this NEXT command, the TransDat segment is the anchor and the MovInfo segment is the target. The resulting CustMovies stack contains all the fields needed to define the data source path from the root segment to the target segment:

- The ancestor segment of the anchor, Cust (key field only).
- All segments from the anchor through the root: TransDat, Rentals, MovInfo (all fields).

The stack looks like this:

CustID	TransDate	MovieCode	...	Title	...	Copies
7173	91/06/18	305PAR	...	AIRPLANE	...	2

CustID	TransDate	MovieCode	...	Title	...	Copies
7173	91/06/30	651PAR	...	MY LIFE AS A DOG	...	3

Creating Data Source Stack User-Defined Columns

In addition to creating data source stack columns that correspond to data source fields, you can also create data source stack columns that you define yourself. You can define these columns in two ways:

- **Within a procedure.** You can create a stack column, as well as user-defined variables, by issuing a COMPUTE command. You can also use the COMPUTE command to assign values to stack cells.

  Because all Maintain Data variables are local to a procedure, you must redefine variables in each procedure in which you use them. For user-defined stack columns, you accomplish this by simply reissuing the original COMPUTE command in each procedure to which you are passing the stack. You only need to specify the format of the variable. You do not need to specify its value, which is passed with the stack.

- **Within the Master File.** You can define a virtual field in a Master File by using the DEFINE attribute. The field is then available in every procedure that accesses the data source. The virtual field is treated as part of the data source segment in which it is defined, and Maintain Data automatically creates a corresponding column for it, a virtual column, in every stack that references that segment.

  Virtual fields must be derived, directly or indirectly, from data source values. They cannot be defined as a constant. The expression assigned to a virtual field in the Master File can reference fields from other segments in the same data source path as the virtual field.

  Unlike other kinds of stack columns, you cannot update a virtual column or field, and you cannot test it in a WHERE phrase.

Example: Creating a User-Defined Column

Consider a data source stack named Pay that contains information from the Employee data source. If you want to create a user-defined column named Bonus and set its value to 10% of the current salary of each employee, you could issue the COMPUTE command to create the new column, and then issue another COMPUTE to derive the value. You place the second COMPUTE within a REPEAT loop to run it once for each row in the stack:

```
COMPUTE Pay.Bonus/D10.2;
REPEAT Pay.FocCount  Row/I4=1;
    COMPUTE Pay(Row).Bonus = Pay(Row).Curr_Sal * .10;
ENDREPEAT  Row=Row+1;
```

Copying Data Into and Out of a Data Source Stack

You can copy data into and out of a data source stack in the following ways:

- **Between a stack and a data source.** You can copy data from a data source into a stack using the NEXT and MATCH commands. You can copy data in the opposite direction, from a stack into a data source, using the INCLUDE, UPDATE, and REVISE commands. In addition, the DELETE command, while not actually copying a stack data, reads a stack to determine which records to remove from a data source. For more information about these commands, see *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

- **Between a stack and a form.** You can copy data from a stack into a form, and from a form into a stack, by specifying the stack as the source or destination of the data displayed by the form. This technique makes it easy for an application user to enter and edit stack data at a personal computer.

- **From a transaction file to a stack.** You can copy data from a transaction file to a stack using the NEXT command. For more information about this command, see the *App Studio Maintain Data Language Reference* manual.

- **Between two stacks.** You can copy data from one stack to another using the COPY and COMPUTE commands. For more information about these commands, see *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

You can use any of these commands to copy data by employing the command INTO and FROM phrases. FROM specifies the command data source (the source stack), and INTO specifies the command data destination (the destination stack).

Example: Copying Data Between a Data Source Stack and a Data Source

In this NEXT command

```
FOR ALL NEXT CustID INTO CustStack;
```

the INTO phrase copies the data (the CustID field and all of the other fields in that segment) into CustStack. The following UPDATE command

```
FOR ALL UPDATE ExpDate FROM CustStack;
```

uses the data from CustStack to update records in the data source.

Referring to Specific Stack Rows Using an Index

Each stack has an index that enables you to refer to specific rows. For example, by issuing a NEXT command, you create the CustNames stack to retrieve records from the VideoTrk data source:

```
FOR ALL NEXT CustID LastName INTO CustNames
   WHERE ExpDate GT 920624;
```

The first record retrieved from VideoTrk becomes the first row in the data source stack, the second record becomes the second row, and so on.

	CustID	**LastName**	**...**	**Zip**
1	0925	CRUZ	...	61601
2	1118	WILSON	...	61601
3	1423	MONROE	...	61601
4	2282	MONROE	...	61601
5	4862	SPIVEY	...	61601
6	8771	GARCIA	...	61601
7	8783	GREEN	...	61601
8	9022	CHANG	...	61601

You can refer to a row in the stack by using a subscript. The following example refers to the third row, in which CustID is 1423:

```
CustNames(3)
```

You can use any integer value as a subscript: an integer literal (such as 3), an integer field (such as TransCode), or an expression that resolves to an integer (such as TransCode + 2).

You can even refer to a specific column in a row (that is, to a specific stack cell) by using the stack name as a qualifier:

```
CustNames(3).LastName
```

If you omit the row subscript, the position defaults to the first row. For example,

```
CustNames.LastName
```

is equivalent to

```
CustNames(1).LastName
```

Maintain Data provides two system variables associated with each stack. These variables help you to navigate through a stack and to manipulate single rows and ranges of rows:

- **FocCount.** This value of the variable is always the number of rows currently in the stack and is set automatically by Maintain Data. This is very helpful when you loop through a stack, as described in the following section, *Looping Through a Stack* on page 54. FocCount is also helpful for checking if a stack is empty:

  ```
  IF CustNames.FocCount EQ 0 THEN PERFORM NoData;
  ```

- **FocIndex.** This variable points to the current row of the stack. When a stack is displayed in a form, the form sets FocIndex to the row currently selected in the grid or browser. Outside of a form, the developer sets the value of FocIndex. By changing its value, you can point to any row you wish. For example, in one function you can increment FocIndex for the Rental stack:

  ```
  IF Rental.FocIndex LT Rental.FocCount
     THEN COMPUTE Rental.FocIndex = Rental.FocIndex + 1;
  ```

  You can then invoke a second function that uses FocIndex to retrieve desired records into the MovieList stack:

  ```
  FOR ALL NEXT CustID MovieCode INTO MovieList
    WHERE VideoTrk.CustID EQ Rental(Rental.FocIndex).CustID;
  ```

  The syntax "*stackname*(*stackname*.FocIndex)" is identical to "*stackname*() ", so you can code the previous WHERE phrase more simply as follows:

  ```
  WHERE VideoTrk.CustID EQ Rental().CustID
  ```

Looping Through a Stack

The REPEAT command enables you to loop through a stack. You can control the process in different ways, so that you can loop according to several factors:

- The number of times specified by a literal, or by the value of a field or expression.
- The number of rows in a stack, by specifying the FocCount variable of the stack.
- While an expression is true.
- Until an expression is true.

- Until the logic within the loop determines that the loop should be exited.

You can also increment counters as part of the loop.

Example: Using REPEAT to Loop Through a Stack

The following REPEAT command loops through the Pay stack once for each row in the stack and increments the temporary variable Row by one for each loop:

```
REPEAT Pay.FocCount  Row/I4=1;
    COMPUTE Pay(Row).NewSal = Pay(Row).Curr_Sal * 1.10;
ENDREPEAT  Row=Row+1;
```

Sorting a Stack

You can sort the row of a stack using the STACK SORT command. You can sort the stack by one or more of its columns and sort each column in ascending or descending order. For example, the following STACK SORT command sorts the CustNames stack by the LastName column in ascending order (the default order):

```
STACK SORT CustNames BY LastName
```

Editing and Viewing Stack Values

There are multiple ways in which you can edit and/or view the values of a stack.

- **Forms.** You can display a stack in an HTML table or a grid on a form. A grid enables you to edit the fields of a stack directly on the screen. You cannot edit a stack in an HTML table.

- **COMPUTE command.** You can use the COMPUTE command to assign a value to any of the cells of a stack. When assigning a value, the COMPUTE keyword is optional, as described in *Command Reference* in the *App Studio Maintain Data Language Reference* manual. For example, the following command assigns the value 35000 to the cell at the intersection of row 7 and column NewSal in the Pay stack:

  ```
  COMPUTE Pay(7).NewSal = 35000;
  ```

It is important to note that if you do not specify a row when you assign values to a stack, Maintain Data defaults to the first row. Thus, if the Pay stack has 15 rows and you issue the following command

```
COMPUTE Pay.NewSal = 28000;
```

the first row receives the value 28000. If you issue this NEXT command

```
FOR 6 NEXT NewSal INTO Pay;
```

the current row of Pay defaults to one, and so the six new values are written to rows one through six of Pay. Any values originally in the first six rows of Pay will be overwritten.

If you wish to append the new values to Pay, that is, to add them as new rows 16 through 21, you would issue this NEXT command, which specifies the starting row:

```
FOR 6 NEXT NewSal INTO Pay(16);
```

You can accomplish the same thing without needing to know the number of the last row by of the stack using the FocCount variable:

```
FOR 6 NEXT NewSal INTO Pay(Pay.FocCount+1);
```

If you want to discard the original contents of Pay and substitute the new values, it is best to clear the stack before writing to it using the following command:

```
STACK CLEAR Pay;
FOR 6 NEXT NewSal INTO Pay;
```

Default Data Source Stack: The Current Area

For all data source fields referenced by a Maintain Data procedure, Maintain Data creates a corresponding column in the default data source stack known as the *Current Area*.

The Current Area is always present and is global to the procedure. It has one row, and functions as a kind of data source buffer. Each data source field, that is, each field described in a Master File that is accessed by a Maintain Data procedure, has a corresponding column in the Current Area. When a data source command assigns a value, either to a field using INCLUDE, UPDATE, or REVISE, or from a field to a stack using NEXT or MATCH, Maintain Data automatically assigns that same value to the corresponding column in the single row of the Current Area. If a set-based data source command writes multiple values to or from a stack column, the last value that the command writes is the one that is retained in the Current Area.

Note: Stacks are a superior way of manipulating data source values. Since the Current Area is a buffer, it does not function as intuitively as stacks do. It is recommended that you use stacks instead of the Current Area to manipulate data source values.

For example, if you write 15 values of NewSal to the Pay stack, the values will also be written to the NewSal column in the Current Area; since the Current Area has only one row, its value will be the fifteenth (that is, the last) value written to the Pay stack.

The Current Area is the default stack for all FROM and INTO phrases in Maintain Data commands. If you do not specify a FROM stack, the values come from the single row in the Current Area. If you do not specify an INTO stack, the values are written to the single row of the Current Area, so that only the last value written remains.

The standard way of referring to a stack column is by qualifying it with the stack name and a period:

```
stackname.columnname
```

Because the Current Area is the default stack, you can explicitly reference its columns without the stack name, by prefixing the column name with a period:

```
.columnname
```

Within the context of a WHERE phrase, an unqualified name refers to a data source field (in a NEXT command) or a stack column (in a COPY command). To refer to a Current Area column in a WHERE phrase you should reference it explicitly by qualifying it with a period. Outside of a WHERE phrase it is not necessary to prefix the name of a Current Area column with a period, as unqualified field names will default to the corresponding column in the Current Area.

For example, the following NEXT command compares Emp_ID values taken from the Employee data source with the Emp_ID value in the Current Area:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Code INTO PayStack
   WHERE Employee.Emp_ID EQ .Emp_ID;
```

If the Current Area contains columns for fields with the same field name but located in different segments or data sources, you can distinguish between the columns by qualifying each one with the name of the Master File and/or segment in which the field is located:

```
masterfile_name.segment_name.column_name
```

If a user-defined variable and a data source field have the same name, you can qualify the name of the Current Area column of the data source field with its Master File and/or segment name; an unqualified reference will refer to the user-defined variable.

Maximizing Data Source Stack Performance

When you use data source stacks, there are several things you can do to optimize performance:

- ❑ **Filter out unnecessary rows.** When you read records into a stack, you can prevent the stack from growing unnecessarily large by using the WHERE phrase to filter out unneeded rows.
- ❑ **Clear stacks when done with data.** Maintain Data automatically releases a stack memory at the end of a procedure, but if in the middle of a procedure you no longer need the data stored in a stack, you can clear it immediately by issuing the STACK CLEAR command. Clearing the data frees the stack memory for use elsewhere.

- **Do not reuse a stack for an unrelated purpose.** When you specify a stack as a data source or destination in certain contexts (in the NEXT, MATCH, and INFER commands, and in the HTML canvas for controls), you define the columns that the stack will contain. If you use the same stack for two unrelated purposes, it will be created with the columns needed for both, making it unnecessarily wide.

Controlling the Flow of a Procedure

Maintain Data provides many different ways of controlling the flow of execution within a procedure. You can:

- Nest a block of code. In commands in which you can nest another command, such as in an IF command, you can nest an entire block of commands in place of a single one by defining the block using the BEGIN command.
- Loop through a block of code a set number of times, while a condition remains true, or until it becomes true, using the REPEAT command.
- Branch unconditionally to a block of code called a Maintain Data function. You define the function using the CASE command, and can invoke it in a variety of ways. When the function terminates, it returns control to the command following function invocation.

  Alternatively, you can branch to a function, but not return upon termination, by invoking the function using the GOTO command.
- Branch conditionally using the IF command. If the expression you specify in the IF command is true, the command executes a PERFORM or GOTO command nested in the THEN phrase, which branches to a Maintain Data function.

  Alternatively, you can nest a different command, such as a BEGIN command defining a block of code, to be conditionally run by the IF command.
- Trigger a task in response to a user action. When users perform an action in a form at run time, the action triggers the task, a function, or URL link, that you have specified.

For more information on the commands listed here, see *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

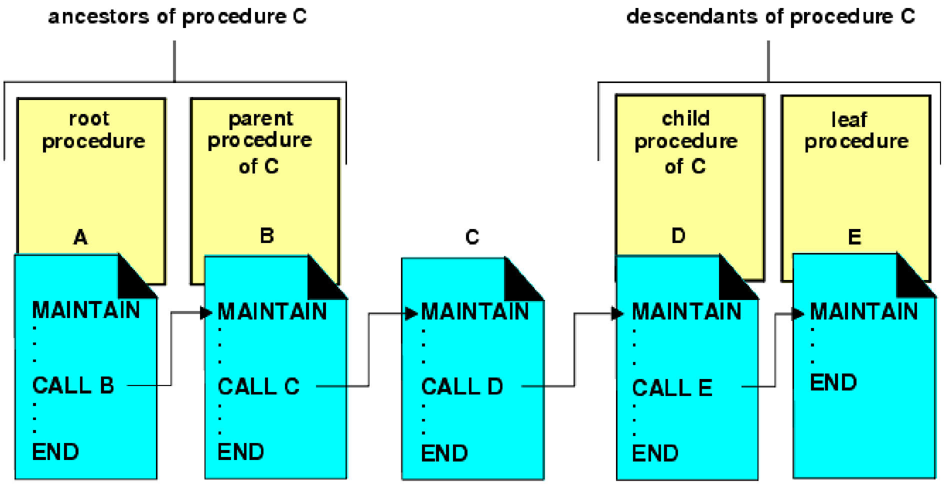
Executing Other Maintain Data Procedures

You can call one Maintain Data procedure from another with the CALL command. *Maintain Data procedure* here means any procedure of Maintain Data language commands. CALL simplifies the process of modularizing an application. Software designed as a group of related modules tends to be easier to debug, easier to maintain, and lends itself to being reused by other applications, all of which increase your productivity.

CALL also makes it easy to partition an application, deploying each type of logic on the platform on which it will run most effectively.

The following diagram illustrates how to describe the relationship between called and calling procedures. It describes a sequence of five procedures from the perspective of the middle procedure, which is named C.

Note: A root procedure is also called the starting procedure.



Calling a Maintain Data Procedure on a Different Server

If parent and child Maintain Data procedures reside on different servers, you identify the location of the child procedure by supplying the AT *server* phrase in the CALL command.

Example: Calling the EmpUpdat Procedure on a Different Server

Consider the EmpUpdat procedure:

```
MAINTAIN FILE Employee
FOR ALL NEXT Emp_ID INTO EmpStack;
.
.
.
CALL NewClass;
.
.
.
END
```

This calls the NewClass procedure on the EducServ Reporting Server:

```
MAINTAIN
.
.
.
END
```

In this example, EmpUpdat is the parent procedure and NewClass is the child procedure. When the child procedure, and any procedures that it has invoked, have finished executing, control returns to the parent.

Passing Variables Between Procedures

All user variables (both stacks and simple, or scalar variables) are global to a function or procedure, but not global to the application. In other words, to protect them from unintended changes in other parts of an application, you cannot directly reference a variable outside of the procedure in which it is found (with the exception of the FocError transaction variable). However, you can access the variable data in other procedures, simply by passing it as an argument from one procedure to another.

To pass variables as arguments, you only need to name them in the CALL command, and then again in the corresponding MAINTAIN command, using the FROM phrase for input arguments and INTO phrase for output arguments. Some variable attributes must match in the CALL and MAINTAIN commands:

- **Number.** The number of arguments in the parent and child procedures must be identical.
- **Sequence.** The order in which you name stacks and simple variables must be identical in the CALL and corresponding MAINTAIN commands.
- **Data type.** Stack columns and simple variables must have the same data type (for example, integer) in both the parent and child procedures.
- **Stack column names.** The names of stack columns need to match. If a column has different names in the parent and child procedures, it is not passed.

Other attributes do not need to match:

- **Stack and scalar variable names.** The names of stacks and simple variables specified in the two commands do not need to match.
- **Other data attributes.** All other data attributes, such as length and precision, do not need to match.
- **Simple variables.** If you pass an individual stack cell, you must receive it as a simple variable, not as a stack cell.

After you have passed a variable to a child procedure, you need to define it in that procedure. How you define it depends upon the type of variable:

- **User-defined columns and fields.** You must redefine each user-defined variable using a DECLARE or COMPUTE command. You only need to specify the variable format, not its value. For example, the following DECLARE command redefines the Counter field and the FullName column:

  ```
  DECLARE Counter/A20;
          EmpStack.FullName/A15;
  ```

- **Data source and virtual stack columns.** You can define the data source columns and virtual columns of the stack in one of two ways. You can define them implicitly, by referring to the stack columns in a data source command, or explicitly, by referring to them using the INFER command. For example:

  ```
  INFER Emp_ID Pay_Date INTO EmpStack;
  ```

  The INFER command declares data source fields and the stack with which they are associated. You can specify one field for each segment you want in the stack or simply one field each from the anchor and target segments of a path you want in the stack.

  While INFER reestablishes the definition of the stack, it does not retrieve any records from the data source.

After a variable has been defined in the child procedure, its data becomes available. If you refer to stack cells that were not assigned values in the parent procedure, they are assigned default values, such as spaces or zeros (0), in the child procedure, and a message is displayed warning that they have not been explicitly assigned any values.

When the child procedure returns control back to the parent procedure, the values of stacks and simple variables specified as output arguments are passed back to the parent. The values of stacks and simple variables specified only as input arguments are not passed back.

Example: Passing Data Between Maintain Data Procedures

This procedure

```
MAINTAIN FILE Employee
FOR ALL NEXT Emp_ID INTO EmpStack;
.
.
.
CALL NewClass FROM EmpStack CourseStack INTO CourseStack;
.
.
.
END
```

calls the NEWCLASS procedure:

```
MAINTAIN FROM StudentStack CourseStack INTO CourseStack
.
.
.
END
```

EmpStack and CourseStack in the parent procedure correspond to StudentStack and CourseStack in the child procedure.

Accessing Data Sources in the Child Procedure

If a child procedure accesses a data source, whether retrieving or writing records, you must specify the data source in the MAINTAIN command. This is done the same way as for a stand-alone procedure. For example, the procedure below specifies the Employee and EducFile data sources:

```
MAINTAIN FILES Employee AND EducFile FROM StuStk INTO CoursStk
.
.
.
END
```

Data Source Position in Child Procedures

Each Maintain Data procedure tracks its own position in the data source. When you first call a procedure, Maintain Data positions you at the beginning of each segment in each data source accessed within that procedure. After navigating through a data source, you can reposition to the beginning of a segment by issuing the REPOSITION command. The data source positions are independent of the positions established in other procedures.

When a child procedure returns control to its parent, by default it clears its data source positions. You can specify that it retain its positions for future calls by using the KEEP option, as described in *Optimizing Performance: Data Continuity and Memory Management* on page 63.

Advantages of Modularizing Source Code Using CALL

Modularizing source code into several procedures has many advantages. One benefit is that you can use multiple procedures, run using the CALL command, to share common source code among many developers, speeding up both development and maintenance time. For example, a generalized error message display procedure could be used by all App Studio Maintain Data developers. After passing a message to the generalized procedure, the procedure would handle message display. The developers do not need to worry about how to display the message, and the error messages will always look consistent to end users.

Another advantage of modular design is that you can remove infrequently-run source code from a procedure and move it into its own procedure. This reduces the size of the original procedure, simplifying its logic, making maintenance easier, and using less memory if the new procedure is not called.

Optimizing Performance: Data Continuity and Memory Management

By default, when you terminate a child procedure, Maintain Data clears its data from memory to save space. You can optimize your application performance by specifying, each time you terminate a child procedure, how you want Maintain Data to handle the procedure data. You have two options, based on how often you will call a given procedure over the course of an application. If you will call the procedure:

- Frequently, use the KEEP option to make the procedure run faster by retaining its data between calls.

  This option provides data continuity. The procedure data carries over from the end of one invocation to the beginning of the next. The next time you call the procedure, its variables and data source position pointers start out with the same values that they held when the procedure was last terminated. You can use these values or reinitialize them using the DECLARE (or COMPUTE) and REPOSITION commands.

  Of course, variables passed by the parent procedure are not affected by data continuity since the child procedure receives them directly from the parent procedure at the beginning of each call.

  KEEP affects transaction integrity in the following way. The KEEP option does not issue an implied COMMIT command at the end of a child procedure. When a child procedure with an open logical transaction returns to its parent procedure and specifies KEEP, the transaction continues into the parent.

- Rarely, use the RESET option to reduce memory consumption by freeing the procedure data at the end of each call.

This option does not provide data continuity. All of the procedure variables and data source position pointers are automatically initialized at the beginning of each procedure.

RESET affects transaction integrity in the following way. The RESET option issues an implied COMMIT command at the end of a child procedure. When a child procedure with an open logical transaction returns to its parent procedure using RESET, the transaction is closed at the end of the child procedure.

You can specify how a procedure will handle its data in memory by terminating it with the GOTO END command qualified with the appropriate memory-management phrase. The syntax is

```
GOTO END [KEEP|RESET];
```

where:

KEEP

Terminates the procedure, but keeps its data, the values of its variables and data source position pointers, in memory. It remains in memory through the next invocation, or (if it is not called again) until the application terminates. The procedure does not issue an implied COMMIT command to close an open logical transaction.

RESET

Terminates the procedure, clears its data from memory, and issues an implied COMMIT command to close an open logical transaction. RESET is the default value.

You can use both options in the same procedure. For example, when you are ready to end a child procedure, you could evaluate what logic the procedure will need to perform when it is next called and then branch accordingly either to keep data in memory, saving time and providing data continuity, or else to clear data from memory to conserve space.

Forms and Event-driven Processing

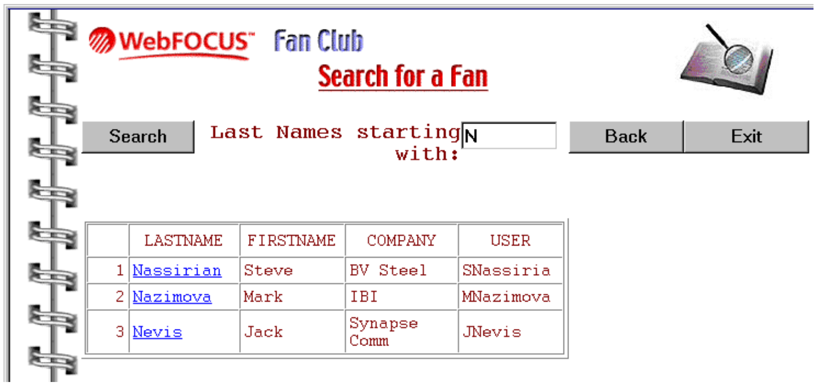
Forms are the visual interface to an App Studio Maintain Data application, giving it a dynamic and attractive face while enabling you to make the application flexible and to place its power at the fingertips of the application end users.

You can design forms that enable end users to:

- ❏ Enter and edit data.
- ❏ Select options.
- ❏ Perform business logic, such as searching a data source for a customer order.
- ❏ Send email.

- Navigate the World Wide Web.
- Read application-specific help information.
- Control the flow of an application using an event-driven paradigm.

You develop these forms and the associated logic using the Form Editor. This is a sample form:



	LASTNAME	FIRSTNAME	COMPANY	USER
1	Nassirian	Steve	BV Steel	SNassiria
2	Nazimova	Mark	IBI	MNazimova
3	Nevis	Jack	Synapse Comm	JNevis

Forms are event-driven, and enable:

- **Event-driven processing.** Forms are responsive to the needs of users because they recognize user activity on the screen, that is, different types of screen events. For example, a form recognizes what the user does on the screen with the keyboard and mouse. It knows when users click a button or change a field value.

  Forms also enable you to assign tasks to these events. Each time a specified event occurs, Maintain Data automatically triggers the corresponding task. If you use events to trigger the application business logic, you can give the user more freedom, for example, over which editing tasks to perform, and in which order. You can also give the user access to more functionality, and more types of data, on a single screen. Event-driven processing gives the user more flexibility over the application, even as it gives the application more control over the user interface.

- **Event-driven development.** App Studio Maintain Data provides you with a simple way of developing event-driven applications, event-driven development. Because much of an application flow can be controlled from forms, you can develop the application as you paint its forms. You can first design the visual layout, then create controls, and finally code tasks, all from the HTML canvas. App Studio Maintain Data also offers the Language Wizard which generates code for you, making it faster and easier to develop effective interfaces and powerful applications.

For an introduction to using forms and developing event-driven applications, see *How to Use Forms* on page 66, *Designing Event-driven Applications* on page 67, and *Creating Event-driven Applications* on page 67.

How to Use Forms

Forms are deployment-independent. You design a form to meet the needs of your application. App Studio Maintain Data automatically implements the form as a webpage. This enables you to focus on logic, and leave implementation details to App Studio Maintain Data.

Forms have standard form features, including:

- A title bar that identifies the form.
- Scroll bars that enable you to move the contents of a control vertically and horizontally if they extend beyond the control border.

Forms are displayed one at a time in one web browser session.

You can transfer control from one form to another, from a form to another Maintain Data procedure or to an App Studio procedure, and from a form to any Internet resource, such as an email client, a webpage, or an FTP server.

Designing a Form

Forms offer a diverse set of ways by which an application end user can select options, invoke procedures, display and edit fields, and get helpful information. For example, if you want the user to select an option or procedure, you can use any of the following controls:

- **Buttons.** You can specify a function to be performed when the end user clicks a button or image. Common examples are Done and Cancel buttons.
- **Radio buttons.** The end user can select one of a mutually exclusive group of options. For example, an employee could identify his or her department.
- **Check boxes.** The end user can select or deselect a single option or characteristic. For example, an applicant could indicate if this is the first time that he or she has applied.
- **Combo boxes and list boxes.** The end user can select one or more options from a dynamic list of choices.
- **Menus.** The end user can select an option from a drop-down menu or submenu.

If you want to display or edit data, you can use these controls:

- **Edit boxes.** You can use this control to edit a single value.

- **Multi-line edit boxes.** You can use this control to edit a long value wrapping onto multiple lines.
- **HTML tables.** You can use this control to view a data source stack.
- **Grids.** You can use this control to edit and view a data source stack.

Designing Event-driven Applications

The flow of control in conventional processing is mostly pre-determined, that is, the programmer determines the few paths that the user will be able to take through the procedure.

To make your application user interface more responsive to the user, App Studio Maintain Data offers event-driven processing. Each time that an event occurs, it invokes, or *triggers*, the assigned task. In App Studio Maintain Data, the event is something the application end user does in a form, and the task is a function or a URL. For example, you might create a button that, when clicked by a user, triggers a task that reads a data source and displays the data in the form.

Creating Event-driven Applications

Developing a procedure by writing out sequential lines of source code may be sufficient for conventional linear processing, but event-driven processing demands event-driven development. Developing an application in this way lets you build much of the application logic around the user interface. In effect, you develop the application as you develop the interface in the HTML canvas. For example, you could start by creating a form, creating a control, and then coding a task for one of the control events. App Studio Maintain Data also provides you with a number of automated tools for developing applications. For example, you can use the Language Wizard to generate source code for various operations such as retrieving and updating data.

Reading From a Data Source

Most applications need to read data from a data source. The most common method is to read data from a data source into a data source stack. Before reading, you first need to select the record in which the data resides. There are five ways of selecting records:

- **By field value for an entire set of records.** Use the NEXT command. The WHERE phrase enables you to select by value, and the FOR ALL phrase selects the entire set of records that satisfy the WHERE selection condition. The basic syntax for this is:

```
FOR ALL NEXT fields INTO stack WHERE selection_condition;
```

- **By field value for a sequence (subset) of records.** Use the NEXT command. This is similar to the technique for a set of records, except that it employs the FOR *n* phrase, selecting, at the current position in the data source, the first *n* records that satisfy the WHERE condition. The basic syntax for this is:

  ```
  FOR n NEXT fields INTO stack WHERE selection_condition;
  ```

- **By field value, one segment at a time, one record at a time.** Use the MATCH command. The basic syntax for this is:

  ```
  MATCH fields [FROM stack] [INTO stack];
  ```

- **Sequentially for a sequence (subset) of records.** Use the NEXT command. This technique employs the FOR *n* phrase to select the next *n* records. The basic syntax for this is:

  ```
  FOR n NEXT fields INTO stack;
  ```

- **Sequentially, one segment instance at a time, one record at a time.** Use the NEXT command. The basic syntax for this is:

  ```
  NEXT fields [INTO stack];
  ```

You can read from individual data sources, and from those that have been joined. Maintain Data supports joins that are defined in a Master File. For information about defining joins in a Master File, see the *Describing Data With WebFOCUS Language* manual. Maintain Data can read from joined data sources, but cannot write to them.

You can evaluate the success of a command that reads from a data source by testing the FocError system variable, as described in *Evaluating the Success of a Simple Data Source Command* on page 70.

The NEXT and MATCH commands are described in detail in *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

Repositioning Your Location in a Data Source

Each time you issue a NEXT command, Maintain Data begins searching for records from the current position in the data source. For example, if your first data source operation retrieved a set of records

```
FOR ALL NEXT CustID INTO SmokeStack
    WHERE ProdName EQ 'VCR DUST COVER';
```

then Maintain Data will have searched sequentially through the entire data source, so the current position marker will now point to the end of the data source. If you then issue another NEXT command

```
FOR ALL NEXT LastName FirstName INTO CandyStack
    WHERE ProdName EQ 'CANDY';
```

Maintain Data searches from the current position to the end of the data source. Since the current position is the end of the data source, no records are retrieved.

When you want a NEXT command to search through the entire data source (as is often the case when you wish to retrieve a set of records) you should first issue the REPOSITION command to move the current position marker to the beginning of the data source.

Example: Repositioning to the Beginning of the Data Source

The following REPOSITION command specifies the CustID field in the root segment, and so moves the current position marker for the root segment chain and all of its descendant chains back to the beginning of the chain (in effect, back to the beginning of the data source):

```
REPOSITION CustID;
FOR ALL NEXT LastName FirstName INTO CandyStack
    WHERE ProdName EQ 'CANDY';
```

Writing to a Data Source

Writing to a data source is the heart of transaction processing applications. Maintain Data provides the following commands to write to a data source:

- **INCLUDE.** Adds the specified new segment instances to a data source.
- **UPDATE.** Updates the specified fields in a data source.
- **REVISE.** Adds new segment instances and updates the specified fields in existing segment instances.
- **DELETE.** Removes the specified segment instances from a data source.

These commands are described in detail in *Command Reference* in the *App Studio Maintain Data Language Reference* manual.

Maintain Data requires that data sources to which it writes have unique keys.

Evaluating the Success of a Simple Data Source Command

When you issue a command that reads or writes to a data source, you should determine if the command was successful. Reasons for a data source command not being successful include attempting to insert a record that already exists, to update a record that does not exist, to delete a record that does not exist, to read a record that does not exist, and being interrupted by a system failure.

When you issue a command that reads or writes to a data source, if the command is:

- ❑ Successful, Maintain Data automatically sets the transaction variable FocError to 0 (zero), and writes to the data source.
- ❑ Unsuccessful, Maintain Data sets FocError to a non-zero value, and does not write to the data source.

Example: Evaluating the Success of an UPDATE Command

The following function updates the VideoTrk data source for new video rentals. If the UPDATE command is unsuccessful, the application invokes a function that displays a message to the user. The line that evaluates the success of the command is highlighted below:

```
CASE UpdateCustOrder
    UPDATE ReturnDate Fee FROM RentalStack;
    IF FocError NE 0 THEN PERFORM ErrorMessage;
ENDCASE
```

Evaluating the Success of a Stack-based Write Command

When you write a set of stack rows to a data source, if you specify more rows than there are matching data source records, this does not invalidate the write operation. Maintain Data attempts to write all the matching rows to the data source. For example, the following UPDATE command specifies 15 rows, but there are only 12 matching rows. All 12 are written to the data source.

```
FOR 15 UPDATE Curr_Sal FROM NewSalaries;
```

When you write a set of stack rows to a data source, if one row fails, the following happens:

- ❑ The rows preceding the failed row are written to the data source.
- ❑ The rows following the failed row are ignored.
- ❑ FocError is set to a non-zero value, signaling an error.
- ❑ FocErrorRow is set to the number of the failed row.

Data source logic errors include attempting to insert an existing record, to update a nonexistent record, and to delete a nonexistent record.

To determine if an entire stack was successfully written to the data source, test FocError immediately following the data source command. If FocError is not zero (0), you can determine which row caused the problem by testing FocErrorRow; you can then reprocess that row. If you will be passing control to a different procedure and reprocessing the row there, consider first setting the stack's FocIndex variable to the value of FocErrorRow in the current procedure, so that after you pass control the stack is already positioned at the problem row.

If you do not wish to take advantage of this flexibility, and instead prefer to invalidate *all* the rows of the stack if any of them are unsuccessful, you can bracket the data source command in a logical transaction that you can then roll back. Logical transactions are discussed in *Transaction Processing* on page 71.

Row failure when committing to a data source. If a stack-based write command is part of a logical transaction, and the write command succeeds when it is issued but fails when the application tries to commit the transaction, Maintain Data rolls back all of the write command rows, along with the rest of the transaction. For example, a write command might fail at commit time because another user has already changed one of the records to which the command is writing. Transaction processing is described in *Transaction Processing* on page 71.

Example: Evaluating a Stack-based Update Command

The NewSalaries stack has 45 rows. The following command updates the Employee data source for all the rows in NewSalaries:

```
FOR ALL UPDATE Curr_Sal FROM NewSalaries;
```

If there is no data source record that matches the thirtieth row of NewSalaries, Maintain Data updates the data source records matching the first 29 rows and ignores records that match rows 30 and higher.

Transaction Processing

You are familiar with individual data source operations that insert, update, or delete data source segment instances. However, most applications are concerned with transactions, such as transferring funds or fulfilling a sales order, that each require several data source operations. These data source operations may access several data sources, and may be issued from several procedures. Such a collection of data source operations is called a *logical transaction* (and is also known as a logical unit of work.)

The advantage of describing a group of related data source commands as one logical transaction is that the transaction is written to the data source only if all of its component commands are successfully written to the data source. Transaction integrity is an uncompromising proposition: if even one part of the transaction fails when you try to write it (by issuing the COMMIT command), Maintain Data automatically rolls back the entire transaction, leaving the data source unchanged.

Transaction integrity also ensures that when several users share access to the same data source, concurrent transactions run as if they were isolated from each other. The changes caused by a transaction in a data source are concealed from all other transactions until that transaction is committed. This prevents each transaction from being exposed to interim inconsistent images of the data source, and so protects the data from corruption.

There are many strategies for managing concurrent data source access. No matter which type of data source you use, Maintain Data respects the DBMS concurrency strategy and lets it coordinate access to its own data sources.

Transaction processing is described in greater detail in the *App Studio Maintain Data Language Reference* manual.

Example: Logical Transactions in a Bank

A banking application would define a transfer of funds from a checking account to a savings account as one logical transaction comprising two update operations:

- ❑ Subtracting the funds from the source account (UPDATE Checking FROM SourceAccts).
- ❑ Adding the funds to the target account (UPDATE Savings FROM TargetAccts).

If the application had not been able to subtract the funds from the checking account, because someone had cleared a check against that account a few moments earlier and depleted its funds, but the application had added the funds to the savings account, the bank accounts would become unbalanced.

The two update commands (subtracting and adding funds) must be described as parts of a single logical transaction, so that the subtraction and addition updates are not written to the data source independently of each other.

Classes and Objects

Most application development is modular, the developer creates complex systems comprised of smaller parts. In conventional development, these modules are processes (such as procedures). In object-oriented development, the modules are models of real-world objects (such as a customer or a shipping order). Each object encapsulates both data and processes.

For example, if you are developing an order fulfillment system for a mail-order clothing business, the objects might include customers, orders, and stock items. The customer data might include the ID code, phone number, and order history. the customer processes might include a function that adds the customer to a new mailing list, a function that updates the customer contact information, and a function that places an order for the customer.

Object-oriented development, because it models the real-world objects with which your enterprise deals, and encourages you to reuse application logic in a variety of ways, is a more efficient way of developing applications. App Studio Maintain Data enables you to create applications using object-oriented development, conventional development, or a hybrid of these two methods, providing you with a flexible path.

What Are Classes and Objects?

Most applications need many objects of the same type. For example, if your business has 500 customers, you need one object to represent each customer. No one would want to design a customer object 500 times. Clearly, you need a template that defines all customer objects, so that you can design the template once, and use it often, each time you create a new customer object to represent a new customer.

The template of an object is called its *class*. Each object is an instance of a class. The class defines what type of object it is. When you create a class, it becomes a new data type, one which you can use to define an object, in the same way that you can use a built-in data type like integer or alphanumeric to define a simple variable like a customer code.

You define a class by describing its properties. Classes have two kinds of properties:

- ❑ Adding the funds to the target account (UPDATE Savings FROM TargetAccts).
- ❑ **Data**, in the form of the class variables. Because these variables exist only as members of the class, they are called *member variables*. In some object-oriented development environments, these are also known as object attributes or instance variables.
- ❑ **Processes**, implemented as functions. Because these functions exist only as members of the class, they are called *member functions*. In some object-oriented development environments, these are also known as methods.

If you want to create a new class that is a special case of an existing class, you could derive it from that existing class. For example, in a human resources application, a class called Manager could be considered a special case of a more general class called Employee. All managers are employees, and possess all employee attributes, plus some additional attributes unique to managers. The Manager class is derived from the Employee class, so Manager is a subclass of Employee, and Employee is the superclass of Manager.

A subclass inherits all of its superclass properties, that is, it inherits all of the superclass member variables and member functions. When you define a subclass you can choose to override some of the inherited member functions, meaning that you can recode them to suit the ways in which the subclass differs from the superclass. You can also add new member functions and member variables that are unique to the subclass.

Classes and objects are described in greater detail in the *App Studio Maintain Data Language Reference* manual.

Chapter 4

Creating an Update Application With Update Assist

Update Assist provides a simple way to create web-based data source file browsers and data maintenance applications in just a few minutes without having to write code.

In this chapter:

Starting Update Assist

You can create an application with Update Assist that adds records, updates records, or deletes records against any data source for which you have read/write access. Data navigation and input validation are automatic. This means you get an update application with no need to design forms, or to write navigation, validation, or update procedures.

To start using Update Assist, do the following:

1. In a Domain folder, create a new HTML page, using the HTML/Document wizard.
2. In the Requests & Data Sources panel, from the New drop-down list, select *Embedded Request* and then select *New Maintain Data with Update Assist* or select *External Request* and then select *Maintain Data: New with Update Assist*.
3. Select the Master File from the Open File dialog box.

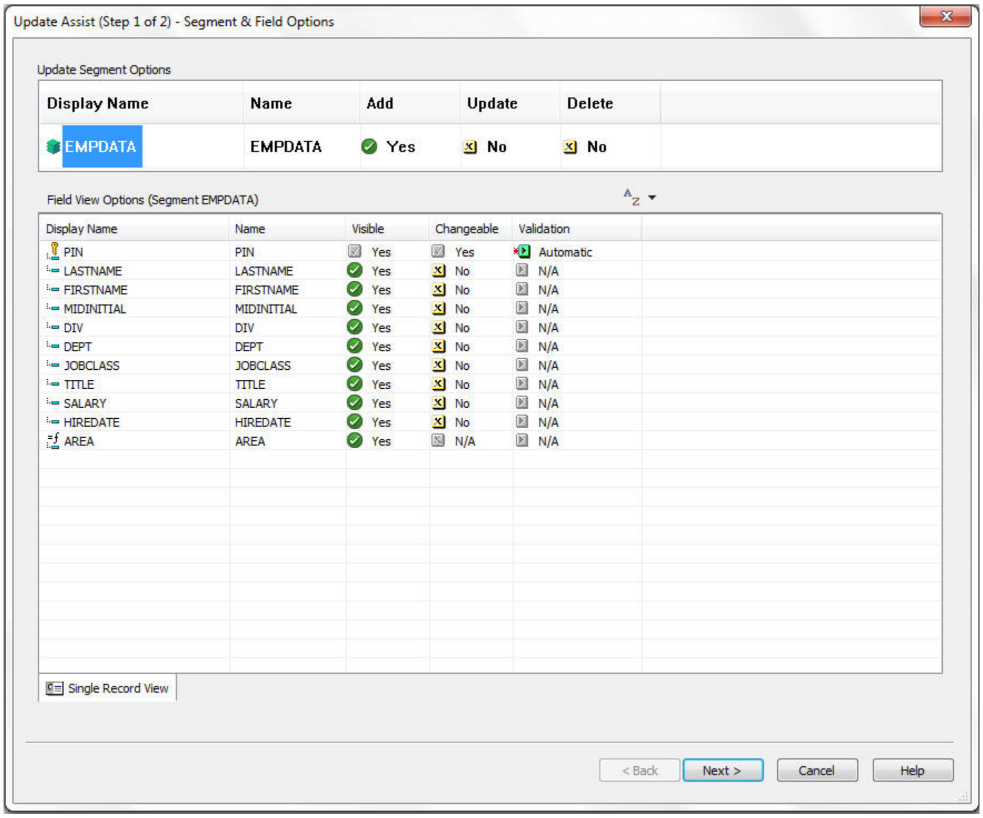
Note:

- ❑ If you are using a single segment Master File, Update Assist consists of two steps.
- ❑ If you are using a multi-segment Master File, Update Assist consists of three steps. The additional dialog box asks you to first select the lowest segment in the hierarchy that you want to update.

The Update Assist dialog boxes correspond to the steps for generating Update Assist applications.

Update Assist: Segment & Field Options

In the Segment & Field Options window of Update Assist, you select the segment and fields you want to update.


Update Assist (Step 1 of 2) - Segment & Field Options
Update Segment Options

Display Name	Name	Add	Update	Delete
EMPDATA	EMPDATA	✓ Yes	☒ No	☒ No

Field View Options (Segment EMPDATA)

Display Name	Name	Visible	Changeable	Validation
PIN	PIN	Yes	Yes	Automatic
LASTNAME	LASTNAME	✓ Yes	☒ No	N/A
FIRSTNAME	FIRSTNAME	✓ Yes	☒ No	N/A
MIDINITIAL	MIDINITIAL	✓ Yes	☒ No	N/A
DIV	DIV	✓ Yes	☒ No	N/A
DEPT	DEPT	✓ Yes	☒ No	N/A
JOBCLASS	JOBCLASS	✓ Yes	☒ No	N/A
TITLE	TITLE	✓ Yes	☒ No	N/A
SALARY	SALARY	✓ Yes	☒ No	N/A
HIREDATE	HIREDATE	✓ Yes	☒ No	N/A
AREA	AREA	✓ Yes	N/A	N/A

This window contains the following fields and options:

Update Segment Options

Contains the segment you selected. In order to enable changes to any of the fields in a segment, select the *Add*, *Update*, or *Delete* option.

Field View Options

Contains the fields in the segment that you selected in the Update Segment Options section. Once you have enabled changes to the segment as a whole, you can set change options for each individual field in the segment.

Display Name

Is how the field name is displayed on the form.

Name

Is the name of the data source field.

Visible

Determines whether the field is visible to the user.

Changeable

Determines whether the user can change the field. This option is available only if *Add* or *Update* was selected in the Update Segment Options section.

Note: A key field cannot be changed.

Tip: You can select multiple fields and then click once to change the *Visible* or *Changeable* settings.

Validation

Applies a validation technique which verifies the value a user enters in the field. This option is available only if Changeable is set to *Yes*.

The options for Validation are:

Automatic. The default validation option, validates the user entry against the field format defined in the Master File. This automatically supports validation for Alphanumeric, Numeric (including Floating Point and Integer), and Date formats. The validation is performed using client-side JavaScript and does not require the server to validate the data.

Range. Allows you to define a numeric range between which data is valid for the field. See *How to Use a Range to Validate a Field* on page 79. This option is best used for numeric fields.

Static List. Allows you to supply a list of valid values from which the user selects at run time. When Static List is selected, the Field Validation - List dialog box opens. To retrieve a list of all values currently in the database, click the *Get Values* button. You can edit this supplied list. For details, see *How to Use a Static List to Validate a Field* on page 80.

Dynamic List. Allows you to supply a list of valid values for the field that are retrieved from a specified data source at run time. When Dynamic List is selected, you are prompted for the Master File and field from which to retrieve values. For details, see *How to Use a Dynamic List to Validate a Field* on page 81.

None. Does not perform a validation.

Required Field. Specifies that the user must supply a value for the field.

Procedure: How to Rename a Segment or a Field

You can easily rename a segment or field as it is displayed to the user (this is called the Display Name).

1. Right-click the segment or field.
2. Click *Rename*.
3. Type the new name and press *Enter*.

Procedure: How to Resort Fields in the Segment & Field Options Window

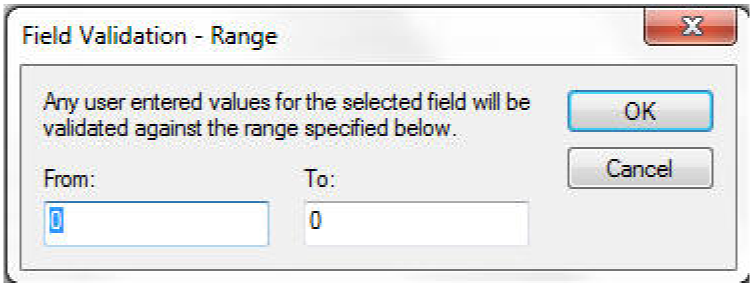
You can change the order of the fields as they appear in the window.

1. In the upper-right corner of the Field View Options pane, click the arrow to the right of the alphabetical sort button .
2. To sort fields by:
   - ❑ Display name, select *Display name* from the drop-down menu.
   - ❑ Original name, select *Name* from the drop-down menu.
   - ❑ The order in which they appear in the Master File, select *Original order* from the drop-down menu.

Note: This does not affect the order of the fields in your application. In the application, the fields are sorted according to their order in the Master File.

Procedure: How to Use a Range to Validate a Field

When you choose the *Range* option to validate a field, Update Assist opens the Field Validation - Range dialog box. You use this option with numeric fields to specify a range of values for any information the user enters. The Field Validation - Range dialog box is shown in the following image.



Any user entered values for the selected field will be validated against the range specified below.

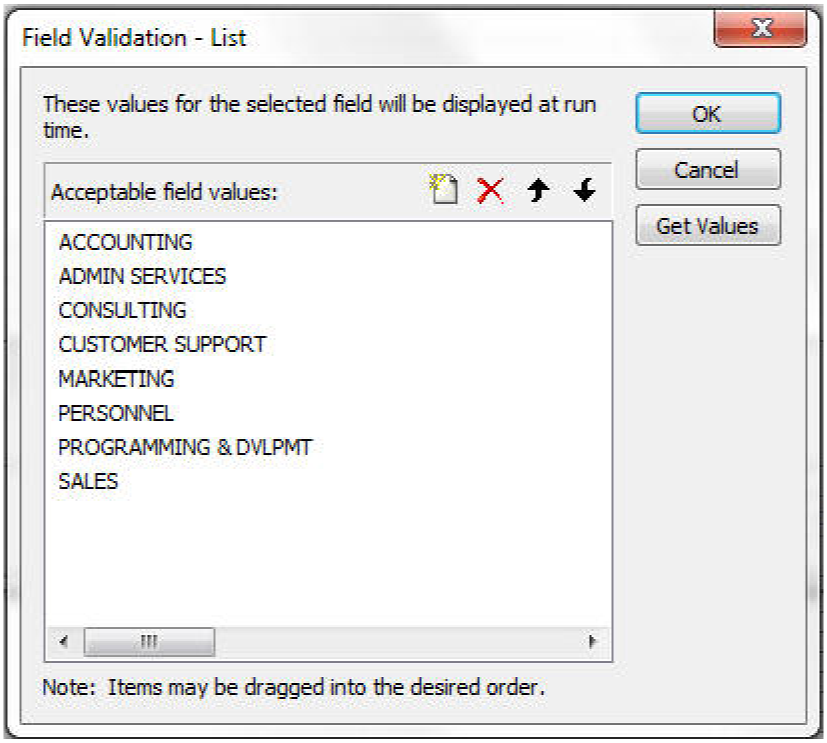
From: 0

To: 0

1. Enter a *From* value to indicate the beginning of the acceptable range of values.
2. Enter a *To* value to indicate the end of the acceptable range of values.
3. Click *OK*.

Procedure: How to Use a Static List to Validate a Field

When you choose the *Static List* option to validate a field, the Field Validation - List dialog box opens. Use this option to specify a static list of values that the user can select from a pull-down list. The Field Validation - List dialog box is shown in the following image.



These values for the selected field will be displayed at run time.

ACCOUNTING
ADMIN SERVICES
CONSULTING
CUSTOMER SUPPORT
MARKETING
PERSONNEL
PROGRAMMING & DVLPMT
SALES

Note: Items may be dragged into the desired order.

1. To enter new acceptable field values, click the *Add new item* button, type the text for the value, and press Enter.
2. To edit an existing value, select it, make any changes, and press Enter.
3. To delete an existing value, select it, and click the *Delete selected items* button.
4. To change the order of the values, use the *move item up in the list* and the *move item down in the list* buttons.
5. When you are done, click *OK*.

Note: When populating a Static list, make sure you scan the data source for all possible values and enter them into the list. If you leave a value off the list that is in a current record and that record is selected for update, the value for the bound column will change to the first item on the Static List.

Tip: If a field is not required and you want to give your user the option to leave it blank, put an empty entry in as the first item in your Static list.

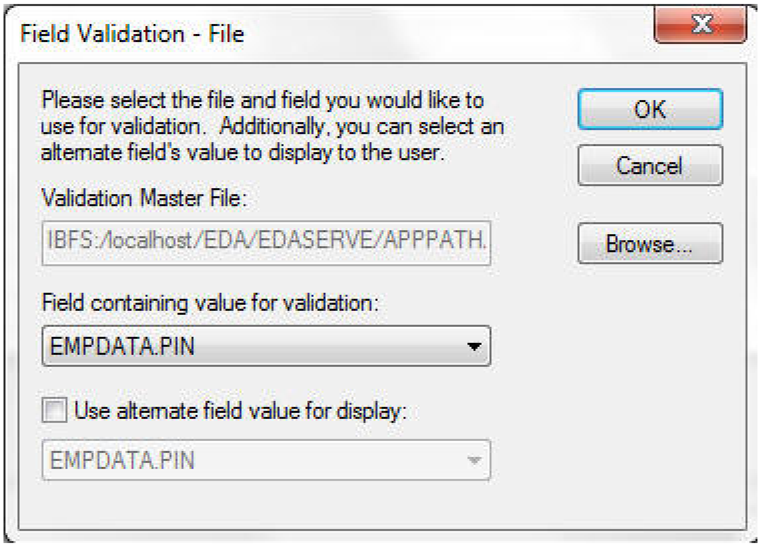
Procedure: How to Use a Dynamic List to Validate a Field

When you choose the *Dynamic List* option to validate a field, you specify a field in the data source that contains the possible values. At run time, a list of values is retrieved from that data source and the user can then select one of these values from a drop-down list.

The real power of Dynamic Lists is that you can add items to the lists in your Update Assist applications without having to make changes to the form code of the application. Static lists require you to edit the forms in your Maintain Data application using the HTML canvas. For example, if you choose to use a flat file as the source of items in your lists, you can simply add items to the flat file or export a new flat file from your data source to change the list. You do not need to change a line of application code.

1. In the Open dialog box, select a Master File for the data source containing the values for the field and click *OK*. You can use any data source type supported by App Studio.

2. The Field Validation - File dialog box opens, as shown in the following image. Select the name of the field in the data source that contains the values you want to validate against and click *OK*. (If you want to select a different data source, click *Browse*.)


Field Validation - File

Please select the file and field you would like to use for validation. Additionally, you can select an alternate field's value to display to the user.

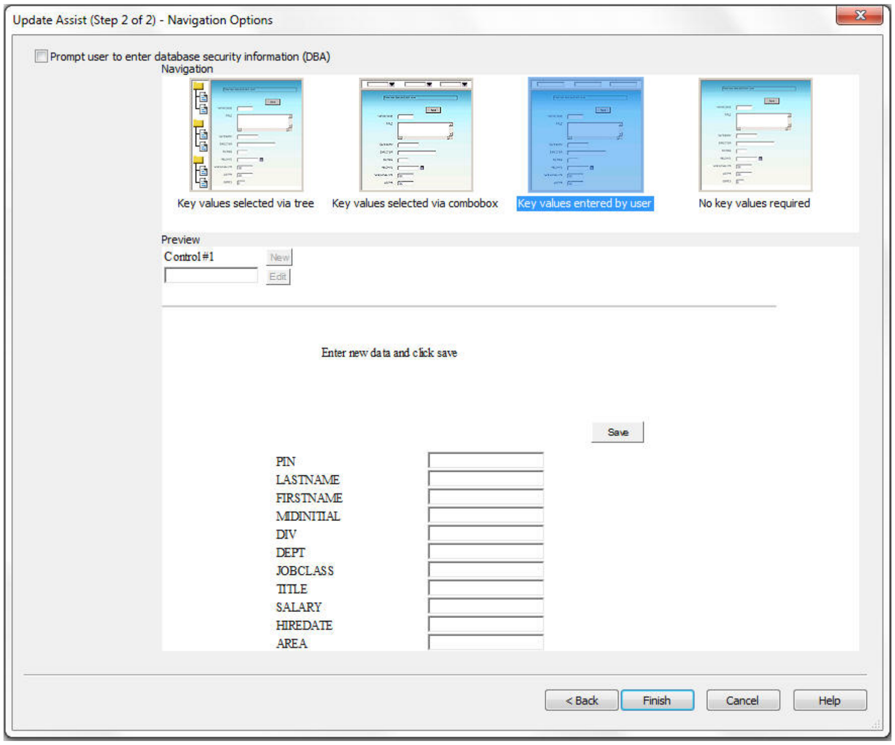
Validation Master File: IBFS:/localhost/EDA/EDASERVE/APPPATH.

Field containing value for validation: EMPDATA.PIN

☐ Use alternate field value for display: EMPDATA.PIN

Update Assist: Navigation Options

The Navigation Options window of Update Assist is where you determine what the user interface for your Update Assist application will look like. The following image shows the Navigation Options window.


Update Assist (Step 2 of 2) - Navigation Options

☐ Prompt user to enter database security information (DBA)

Navigation

Key values selected via tree | Key values selected via combobox | Key values entered by user | No key values required

Preview

Control #1

Enter new data and click save

PIN
LASTNAME
FIRSTNAME
MIDINITIAL
DIV
DEPT
JOBCLASS
TITLE
SALARY
HIREDATE
AREA

This window contains the following fields and options:

Prompt user to enter database security information (DBA)

Generates a page prompting the user to enter a password to access the data in the data source. Use this option if data source security is enabled.

The application will store the password in a cookie, so the user will only be prompted for it once.

Key values selected via tree

Generates a form in which the user selects records using a hierarchical tree control.

Key values selected via combobox

Generates a form in which the user selects records using a combobox.

Key values entered by user

Generates a form in which the user selects records by entering key values.

Note: This requires that the user knows the actual values for the key values.

No key values required

Generates a Maintain Data procedure to be called by another procedure (usually an App Studio report) with the appropriate values to fill out the screen. To see how to create the App Studio report that calls this type of Update Assist application, see *Calling an Update Assist Procedure From an App Studio Report* on page 88.

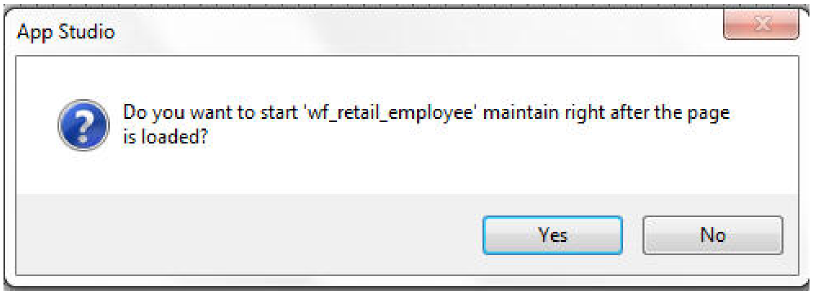
Preview

Displays the selected navigation, theme, and fields.

About Your Update Assist Application

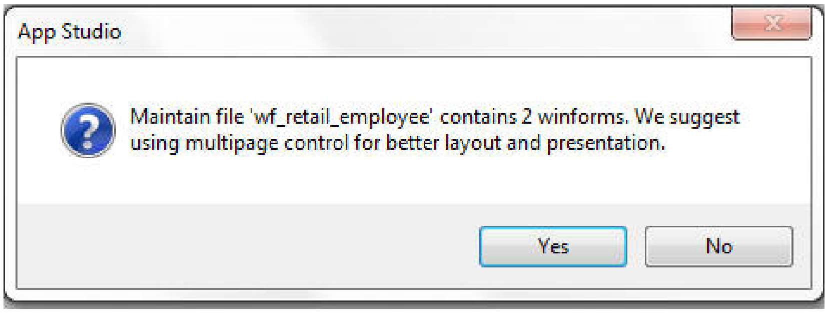
This section describes how to run your application and how Update Assist applications work.

Once you click *Finish* in the final Update Assist window, the dialog box shown in the following image opens. Click *Yes* to start Maintain Data right after the page is loaded.



Do you want to start 'wf_retail_employee' maintain right after the page is loaded?

If the Maintain Data file contains Winforms, the dialog box shown in the following image opens. Click *Yes* to create a multi-page control.



Maintain file 'wf_retail_employee' contains 2 winforms. We suggest using multipage control for better layout and presentation.

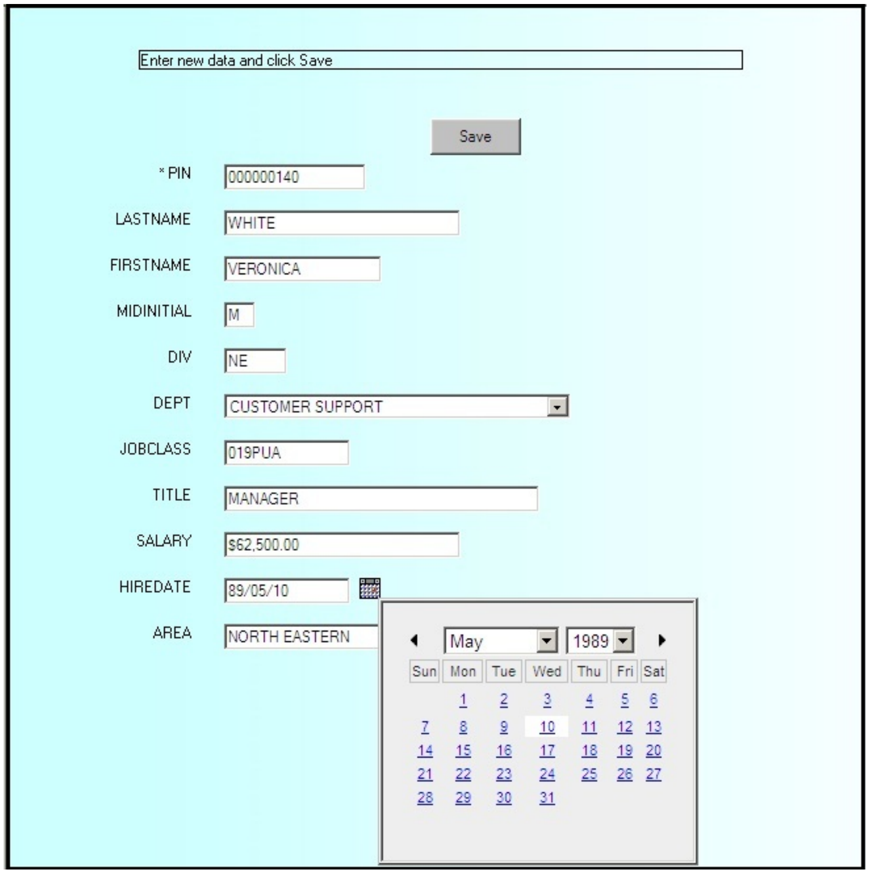
Maintain Data generates the files needed for your application, and runs the application, if specified.

Reference: Working With Empty or New Data Sources

- ❏ **In tree navigation.** If you select the Add option for any data segments that do not contain data, the Tree will display a period (.) for the null segment. You can right-click on the period (.) to enter new data for that segment.
- ❏ **In combo box navigation.** For any data segments that do not contain data, the combo boxes display a [New] command. This option enables you to enter new records.

Calendar Control for Date-Formatted Fields

A calendar icon appears next to changeable date-formatted fields. When a user clicks the calendar icon, a calendar appears, as shown in the following image. Any date selected on this calendar is entered into the date field. Users can also type dates into the date field manually.


Enter new data and click Save

* PIN	000000140
LASTNAME	WHITE
FIRSTNAME	VERONICA
MIDINITIAL	M
DIV	NE
DEPT	CUSTOMER SUPPORT
JOBCLASS	019PUA
TITLE	MANAGER
SALARY	$62,500.00
HIREDATE	89/05/10
AREA	NORTH EASTERN

May 1989

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Date-Stamping Fields

Many DBMSs allow you to create a *time stamp* field. This automatically fills the field with the current date and/or time and saves the user having to do it. There are many reasons at an application level for doing this. The most common is to give reporting applications some way to track when a record was first created or when each change was entered.

Note: If you are using an external DBMS that directly supports Date and Time Stamp field types, you will not need to use this technique. Instead, make sure the field that contains the time stamp is set to Changeable = No to prevent Update Assist from touching that field.

Procedure: How to Date-Stamp a Field in an Update Assist Application

To date-stamp a field in an Update Assist application, so that when a user clicks *New*, the application can set the initial value of the field to the current date in the stack before it is displayed in the form:

1. Open the *SegmentName*.mnt file in the HTML canvas.
2. Add the following line of code to the top of the Maintain Data procedure, just above Case Top:

   ```
   MODULE IMPORT(MNTUWS);
   ```

   This imports the library of functions included with App Studio Maintain Data.
3. Scroll down to the newrecord case and add the following code right below the first Stack Clear statement:

   ```
   COMPUTE TheDate/MDY = Today();
   COMPUTE stack.datefield = TheDate;
   ```

   where *stack* and *datefield* are the stack name and field name to which you want to assign the current date.

Note: If you have multiple fields that need to be set to the current date, you only need to set the variable TheDate once and can reuse it as many times as you need.

Example: Date-Stamping a Field in the MOVIES Data Source

If you wanted the Release Date field from the MOVIES data source to contain the current date, your code would look like this:

```
COMPUTE TheDate/MDY = Today();
COMPUTE Movinfo_stack.RELDATE = TheDate;
```

Auto-numbering Fields in Update Assist Applications

Some DBMSs allow you to create an *auto-number* field. This automatically fills the field with a sequence number that is the last record index plus one. This saves the user having to make up an arbitrary key for the record.

Procedure: How to Auto-Number a Field in an Update Assist Application

To auto-number a field in an Update Assist application, so that when a user clicks *New*, the application can set the initial value of the field to the next sequence number in the stack before it is displayed in the form:

1. Open the *SegmentName*.mnt file in the HTML canvas.
2. Scroll down to the newrecord case and add the following code right below the first Stack Clear statement:

   ```
   Stack clear SegmentNameStk;
   For all next MasterFileName.SegmentName.autonum into SegmentNameStk;
   NextVal/I5 = SegmentNameStk(SegmentNameStk.FOCCOUNT).val + 1;
   Stack clear SegmentNameStk;
   ```

Note: If you are using an external DBMS that directly supports Date Stamp field types, you will not need to use this technique.

Continuing to Display Current Values After a New Action

By default, Update Assist clears all text boxes and controls in the form on a New action. You can have the values stay in the text boxes by editing the *SegmentName*.MNT file.

For example, users of some types of applications may be entering many similar records, one after another, and would like to display a record, then essentially have the New action display a copy of the record which they can edit before clicking *Save*.

Procedure: How to Continue Displaying Current Values After a New Action

1. Open the *SegmentName*.MNT file and go to the newrecord case.
2. Comment out the line that clears the stack, using a double dollar sign ($$).

Calling an Update Assist Procedure From an App Studio Report

One way to use an Update Assist procedure is to call it from an App Studio report. You can set up the App Studio report so that a user can click on a row in the report and open the Update Assist procedure with the data from the row of the report.

Procedure: How to Call an Update Assist Maintain Data Procedure From an App Studio Report

1. Create an Update Assist procedure.
2. In the Update Assist - Navigation Options window, select *No key values required* for your user interface.
3. In the HTML canvas, create a report using the same data source you used for the Update Assist application. The report must contain the key fields in the segment you want to update (if you do not want to view them in the results, you can make them invisible).

4. Select the column you want to make clickable on the report, and on the Appearance tab, click *Drill Down* in the Links group.

   The Drill Down dialog box opens.

5. Click the Add new item icon to create a new drill down.
6. From the Drill Down Type drop-down menu, select *JavaScript*.
7. In the Source field, type *parent.IbComposer_triggerExecution*.
8. From the Target Frame drop-down menu, select *_parent*.
9. Double-click in the Parameters field to display the Parameters dialog box.
10. Click the Add new item icon to create a parameter.

    a. From the Parameter Type drop-down menu, select *Constant Value* and enter *taskn*.

       where:

       `taskn`

       Is the number of the task that launches the Maintain procedure.

    b. Click the Add new item icon, and from the Parameter Type drop-down menu, select *Constant Value*, and type *1*.

    c. Click the Add new item icon, and from the Parameter Type drop-down menu, select *Constant Value*, and type *segname.segnamestk.keyfield*.

       where:

       `segname`

       Is the name of the segment that contains the key field.

       `segnamestk`

       Is the name of the segment with stk added to the end.

       `keyfield`

       Is the name of the key field selected for drill down.

    d. Click the Add new item icon, and from the Parameter Type drop-down menu, select *Field*.

    e. Select the name of the key field from the drop-down list, as entered in step c.

    f. Click *Ok* to create the drill down.

11. Close your procedure and save it.

When you run your report, you will see that all of the items in the selected column of the report are underlined and clickable. Clicking any item on the report opens the Update Assist form with the information for that item already filled in.

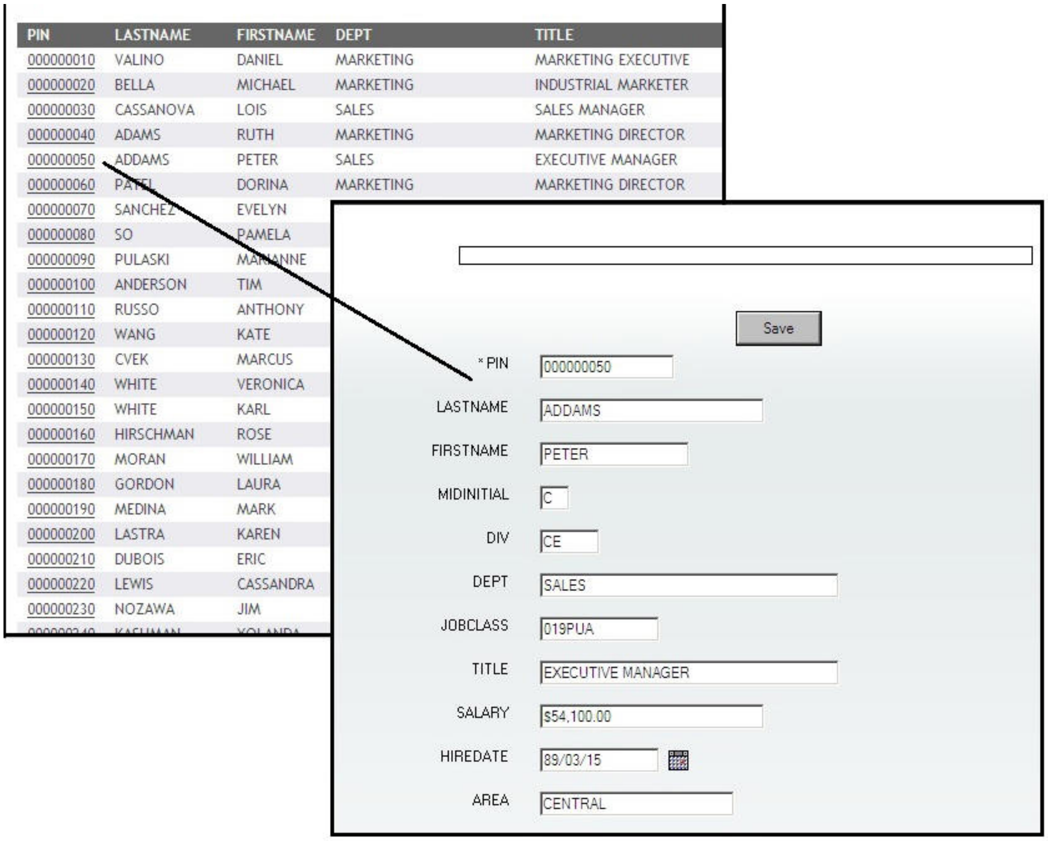
Calling an Update Assist Application From an App Studio Report Example

This example describes how to create a report in the HTML canvas and then create a link to a simple Update Assist application that will update information in the report.

This example is broken down into two steps:

1. Create an Update Assist application that updates an employee from the empdata data source.
2. Create a report in the HTML canvas that contains a simple list of the employees in the empdata data source.

When you are done, you will have an App Studio report that displays a list of customers in the employee data source. Clicking on the PIN adjacent to the last name of someone in this report will bring up a form where you can change information about the employee, or delete the employee from the data source. The result is shown in the following image.



PIN	LASTNAME	FIRSTNAME	DEPT	TITLE
000000010	VALINO	DANIEL	MARKETING	MARKETING EXECUTIVE
000000020	BELLA	MICHAEL	MARKETING	INDUSTRIAL MARKETER
000000030	CASSANOVA	LOIS	SALES	SALES MANAGER
000000040	ADAMS	RUTH	MARKETING	MARKETING DIRECTOR
000000050	ADDAMS	PETER	SALES	EXECUTIVE MANAGER
000000060	PATEL	DORINA	MARKETING	MARKETING DIRECTOR
000000070	SANCHEZ	EVELYN		
000000080	SO	PAMELA		
000000090	PULASKI	MARIANNE		
000000100	ANDERSON	TIM		
000000110	RUSSO	ANTHONY		
000000120	WANG	KATE		
000000130	CVEK	MARCUS		
000000140	WHITE	VERONICA		
000000150	WHITE	KARL		
000000160	HIRSCHMAN	ROSE		
000000170	MORAN	WILLIAM		
000000180	GORDON	LAURA		
000000190	MEDINA	MARK		
000000200	LASTRA	KAREN		
000000210	DUBOIS	ERIC		
000000220	LEWIS	CASSANDRA		
000000230	NOZAWA	JIM		

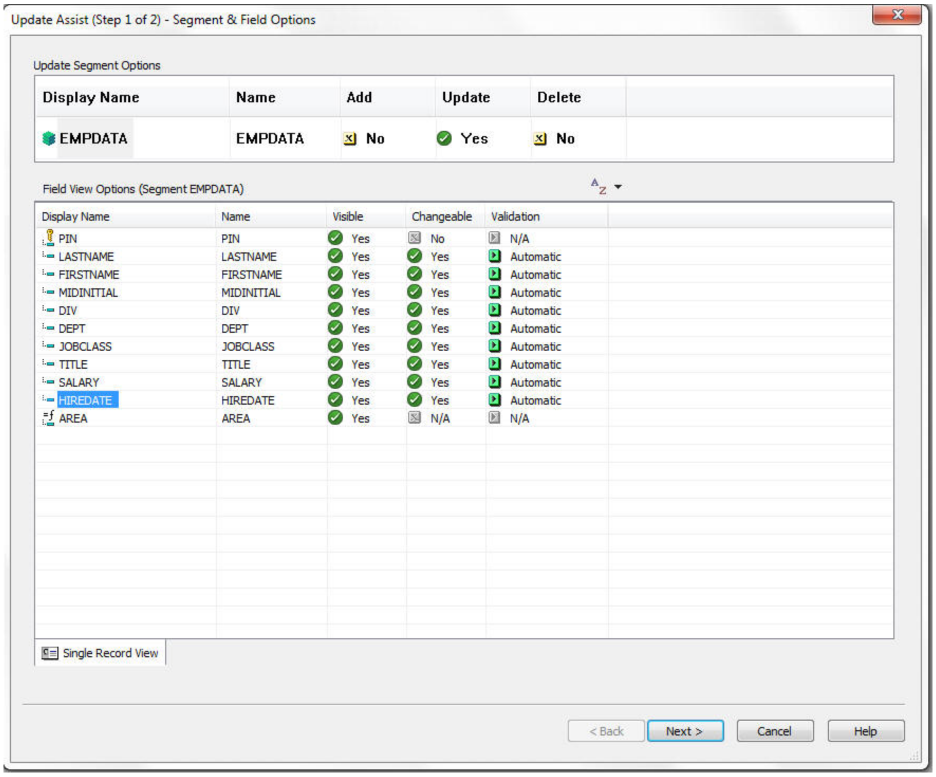
* PIN	000000050
LASTNAME	ADDAMS
FIRSTNAME	PETER
MIDINITIAL	C
DIV	CE
DEPT	SALES
JOBCLASS	019PUA
TITLE	EXECUTIVE MANAGER
SALARY	$54,100.00
HIREDATE	89/03/15
AREA	CENTRAL

Example: **Creating an Update Assist Application For the Empdata Data Source**

The following is an example of creating an Update Assist application for the empdata data source.

1. In a Domain folder, create a new HTML page, using the HTML/Document wizard.
2. Attach the ibisamp app folder to the Domain folder, using the Application Paths property in the File/Folder Properties panel.
3. In the Requests & Data Sources panel, from the New drop-down list, select *Embedded Request* and then select *New Maintain Data with Update Assist*.

4. Select the *empdata* Master File.
5. In the Update Segment Options section, set *Update* to *Yes*.
6. In the Field View Options section, set *Changeable* to *Yes* for all fields except PIN, which is a key field, as shown in the following image.



Update Assist (Step 1 of 2) - Segment & Field Options

Update Segment Options

Display Name	Name	Add	Update	Delete
EMPDATA	EMPDATA	No	✓ Yes	No

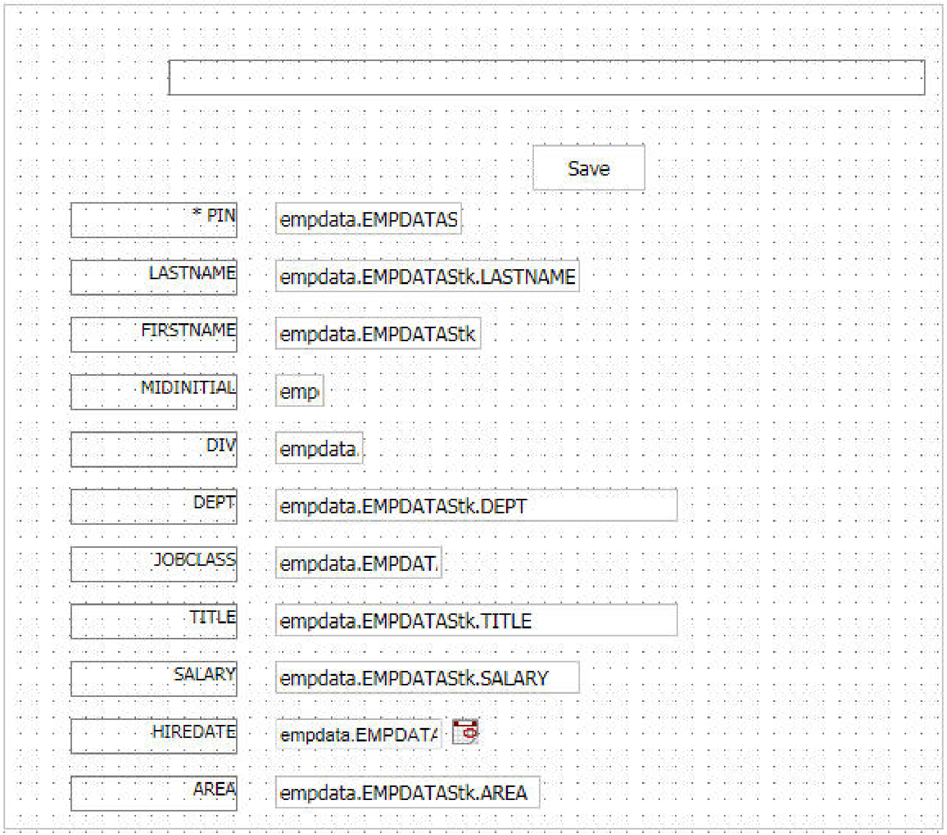
Field View Options (Segment EMPDATA)

Display Name	Name	Visible	Changeable	Validation
PIN	PIN	✓ Yes	No	N/A
LASTNAME	LASTNAME	✓ Yes	✓ Yes	Automatic
FIRSTNAME	FIRSTNAME	✓ Yes	✓ Yes	Automatic
MIDINITIAL	MIDINITIAL	✓ Yes	✓ Yes	Automatic
DIV	DIV	✓ Yes	✓ Yes	Automatic
DEPT	DEPT	✓ Yes	✓ Yes	Automatic
JOBCLASS	JOBCLASS	✓ Yes	✓ Yes	Automatic
TITLE	TITLE	✓ Yes	✓ Yes	Automatic
SALARY	SALARY	✓ Yes	✓ Yes	Automatic
HIREDATE	HIREDATE	✓ Yes	✓ Yes	Automatic
AREA	AREA	✓ Yes	N/A	N/A

7. Click *Next*.

   The Update Assist - Navigation Options window opens.
8. Select *No key values required* and click *Finish*.
9. Click *Yes* to start Maintain Data right after the page is loaded.
10. Click *Yes* to create a multi-page control.

App Studio Maintain Data displays the first screen of the application you created, as shown in the following image.



11.Open the Tasks & Animations panel.

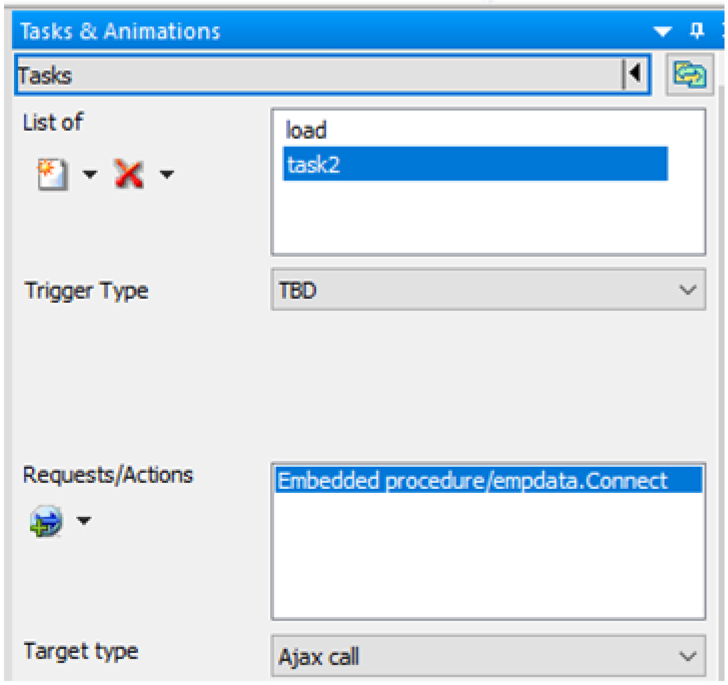
12.Click the New icon to create task2.

13.From the Trigger Type drop-down list, select *TBD (To Be Determined)*.

14.Click the down arrow, under Requests/Actions.

15.Select *Run Request*, then *empdata*, and then *empdata.Connect*.

The Tasks & Animations screen is shown in the following image.

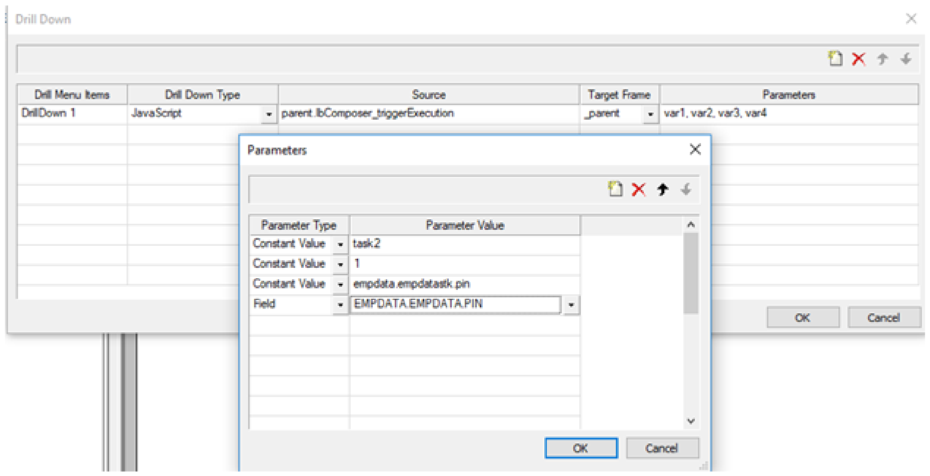


Example: Creating a Report With the Empdata Data Source

1. Create a report in the HTML canvas:
   a. From the Environments tree, right-click the folder that contains the Update Assist procedure, and select *New* from the pop-up window, then select *Report*.
   b. Select *empdata* and click *Finish*.
2. Place the *PIN*, *LASTNAME*, *FIRSTNAME*, *DEPT*, and *TITLE* fields in the report.
3. Select the *PIN* field on the report, and on the Appearance tab, click *Drill Down* in the Links group.

   The Drill Down dialog box opens.
4. Click the *Add new item* icon.
5. From the Drill Menu Items field, delete *DrillDown 1*.
6. Click the Add new item icon to create a new drill down.
7. From the Drill Down Type drop-down menu, select *JavaScript*.

8. In the Source field, type *parent.IbComposer_triggerExecution*.
9. From the Target Frame drop-down menu select *_parent*.
10. Double-click in the Parameters field to display the Parameters entry screen.
11. Click the Add new item icon to create a parameter:
    a. From the Parameter Type drop-down menu, select *Constant Value* and type *task2*.
    b. Click the Add new item icon, and from the Parameter Type drop-down menu, select *Costant Value*, and type *1*.
    c. Click the Add new item Icon, and from the Parameter Type drop-down menu, select *Constant Value*, and type *empdata.empdatastk.pin*.
    d. Click the Add new item icon, and from the Parameter Type drop-down menu, select *Field*.
    e. From the Parameter Value drop-down menu, select *EMPDATA.EMPDATA.PIN*.
    f. Click *Ok* to create the drill down.

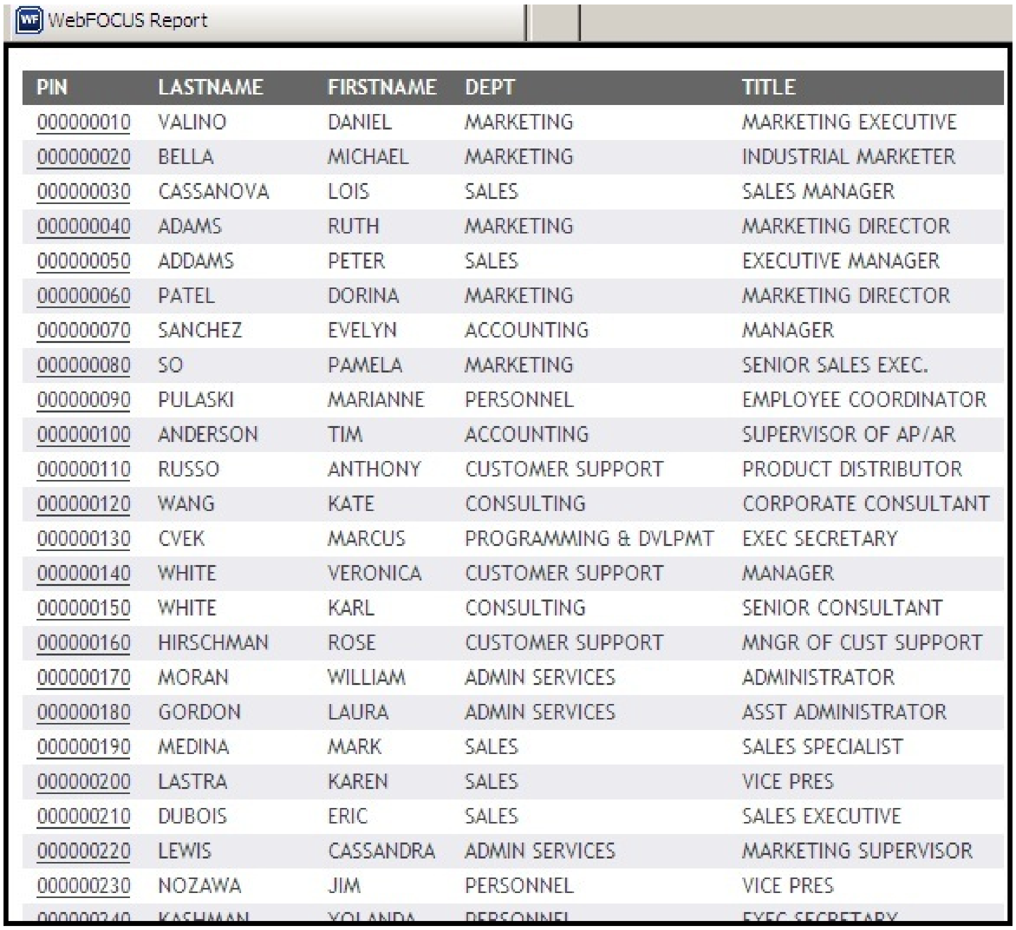


Drill Menu Items	Drill Down Type	Source	Target Frame	Parameters
DrillDown 1	JavaScript	parent.IbComposer_triggerExecution	_parent	var1, var2, var3, var4

Parameter Type	Parameter Value
Constant Value	task2
Constant Value	1
Constant Value	empdata.empdatastk.pin
Field	EMPDATA.EMPDATA.PIN

12. Click *OK*.
13. Close your procedure and save it.

When you run your report, you will see all of the PIN values in the report are underlined and clickable, as shown in the following image.



PIN	LASTNAME	FIRSTNAME	DEPT	TITLE
000000010	VALINO	DANIEL	MARKETING	MARKETING EXECUTIVE
000000020	BELLA	MICHAEL	MARKETING	INDUSTRIAL MARKETER
000000030	CASSANOVA	LOIS	SALES	SALES MANAGER
000000040	ADAMS	RUTH	MARKETING	MARKETING DIRECTOR
000000050	ADDAMS	PETER	SALES	EXECUTIVE MANAGER
000000060	PATEL	DORINA	MARKETING	MARKETING DIRECTOR
000000070	SANCHEZ	EVELYN	ACCOUNTING	MANAGER
000000080	SO	PAMELA	MARKETING	SENIOR SALES EXEC.
000000090	PULASKI	MARIANNE	PERSONNEL	EMPLOYEE COORDINATOR
000000100	ANDERSON	TIM	ACCOUNTING	SUPERVISOR OF AP/AR
000000110	RUSSO	ANTHONY	CUSTOMER SUPPORT	PRODUCT DISTRIBUTOR
000000120	WANG	KATE	CONSULTING	CORPORATE CONSULTANT
000000130	CVEK	MARCUS	PROGRAMMING & DVLPMT	EXEC SECRETARY
000000140	WHITE	VERONICA	CUSTOMER SUPPORT	MANAGER
000000150	WHITE	KARL	CONSULTING	SENIOR CONSULTANT
000000160	HIRSCHMAN	ROSE	CUSTOMER SUPPORT	MNGR OF CUST SUPPORT
000000170	MORAN	WILLIAM	ADMIN SERVICES	ADMINISTRATOR
000000180	GORDON	LAURA	ADMIN SERVICES	ASST ADMINISTRATOR
000000190	MEDINA	MARK	SALES	SALES SPECIALIST
000000200	LASTRA	KAREN	SALES	VICE PRES
000000210	DUBOIS	ERIC	SALES	SALES EXECUTIVE
000000220	LEWIS	CASSANDRA	ADMIN SERVICES	MARKETING SUPERVISOR
000000230	NOZAWA	JIM	PERSONNEL	VICE PRES
000000240	KASHMAN	YOLANDA	PERSONNEL	EXEC SECRETARY

Clicking a PIN on the report opens the Update Assist form with the information for that name already filled in.

Usage Notes

The following are known issues when using App Studio Update Assist:

- Update Assist will not prevent the use of special characters or wildcard designations such as $* when entering data. Entering such character combinations can cause unexpected results.

- When renaming Update Assist HTML files, the ampersand character (&) is not supported.
- Using the browser Refresh action while running an Update Assist application can cause unexpected results and is not recommended.
- Update Assist does not allow updates of cross-referenced segments. You must run the Update Assist on the individual Master Files and create separate update procedures.

Appendix A

App Studio Maintain Data Sample Data Sources

Sample data sources have been used in examples throughout these manuals in order to provide meaningful examples.

For information on the standard sample data sources, see the *Describing Data With WebFOCUS Language* manual.

This chapter contains the Master Files and structure diagrams of the Fannames, Users, and Contact data sources, which are used exclusively in the App Studio Maintain Data manuals.

You can find these sample files in the approot/maintain directory.

In this appendix:

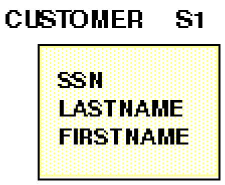
Fannames Data Source

The Fannames data source contains email, address, and telephone information for all fans in a fan club.

Fannames Master File

```
FILENAME=FANNAMES, SUFFIX=FOC
SEGNAME=CUSTOMER, SEGTYPE=S1
 FIELD=SSN,             ALIAS=SSN,        FORMAT=A11,         $
 FIELD=LASTNAME,        ALIAS=LASTNAME,   FORMAT=A10,         $
 FIELD=FIRSTNAME,       ALIAS=FIRSTNAME,  FORMAT=A8,          $
 FIELD=COMPANY,         ALIAS=COMPANY,    FORMAT=A12,         $
 FIELD=ADDRESS,         ALIAS=ADDRESS,    FORMAT=A20,         $
 FIELD=CITY,            ALIAS=CITY,       FORMAT=A10,         $
 FIELD=STATE            ALIAS=STATE,      FORMAT=A2,          $
 FIELD=ZIP,             ALIAS=ZIP,        FORMAT=A5,          $
 FIELD=PHONE,           ALIAS=PHONE,      FORMAT=A15,         $
 FIELD=EMAIL,           ALIAS=EMAIL,      FORMAT=A20,         $
 FIELD=TITLE,           ALIAS=TITLE,      FORMAT=A4,          $
 FIELD=USER,            ALIAS=USER,       FORMAT=A9,          $
```

Fannames Structure Diagram



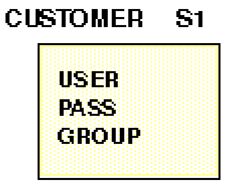
Users Data Source

The Users data source contains personal information about the types of users in a fan club.

Users Master File

```
FILENAME=FANNAMES, SUFFIX=FOC
SEGNAME=CUSTOMER, SEGTYPE=S1
 FIELD=USER,            ALIAS=USER,       FORMAT=A9,          $
 FIELD=PASS,            ALIAS=PASS,       FORMAT=A9,          $
 FIELD=GROUP,           ALIAS=GROUP,      FORMAT=A15,         $
```

Users Structure Diagram



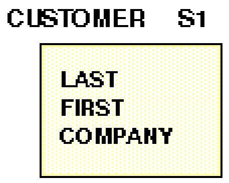
Contact Data Source

The Contact data source contains name, address, telephone, title, and position for all contacts.

Contact Master File

```
FILENAME=CONTACT,  SUFFIX=FOC
SEGNAME=CUSTOMER, SEGTYPE=S1
 FIELD=LAST,            ALIAS=LAST,       FORMAT=A10,          $
 FIELD=FIRST,           ALIAS=FIRST,      FORMAT=A8,           $
 FIELD=COMPANY,         ALIAS=COMPANY,    FORMAT=A12,          $
 FIELD=ADDRESS,         ALIAS=ADDRESS,    FORMAT=A20,          $
 FIELD=CITY,            ALIAS=CITY,       FORMAT=A10,          $
 FIELD=STATE,           ALIAS=STATE,      FORMAT=A2,           $
 FIELD=ZIP,             ALIAS=ZIP,        FORMAT=A5,           $
 FIELD=PHONE,           ALIAS=PHONE,      FORMAT=A15,          $
 FIELD=TITLE,           ALIAS=TITLE,      FORMAT=A6,           $
 FIELD=POSITION,        ALIAS=POSITION,   FORMAT=A15,          $
```

Contact Structure Diagram



Legal and Third-Party Notices

Index