

For Barista Release 9.12
Revised June 05, 2010



How to design your first Barista application

Barista Introduction Tutorial

Table of Contents

1. Introduction.....	2
2. Our Projekt.....	4
2.1 Start Barista.....	4
2.2 Create a Barista Application	5
2.3 Build a Simple Form	7
2.4 Define Element Types.....	7
2.5 Define a table.....	14
2.6 Using the Tables Form.....	14
2.7 Create the Table	17
2.8 Using the Form Manager	18
2.9 Build the Form.....	18
2.10 Run the Form.....	18
3. Customize the Form.....	20
3.1 Load the Form Manager.....	20
3.2 Changing a Control's Location or Tab Order	21
3.3 Adding an Optional Definition to a Column	23
3.4 Restoring Form's Original Layout	23
3.5 Callpoints	24
3.6 Additional Options	24
3.7 Bring up the Callpoint Editor	24
3.8 Callpoint Editor.....	25
3.9 Add After Column Input (AINP) Callpoint Code for TUT_USER_EMAIL	26
3.10 Calling a custom programm via an Option Button	34
3.11 Setting up the Spam section.....	36
4. Options Entry Form	38
4.1 What We Are Going To Build.....	38
5. Appendix	55
5.1 Further Reading	55
5.2 Definition of Screen Elements.....	56
5.3 Contacts	57

1. Introduction

Barista is a fully-featured rapid application development framework. It offers not only fast creation of application screens based on any given database layout, but ships all of the infrastructure necessary to make a professional business application complete: powerful menu system, role based security, reporting, inquiries with custom filtering and retrieval functionality, and much more.

Before we start with our first simple Barista project, we will give you some overview over the basics.

Concept of Database Driven Development in Barista

Barista does not just generate screens from a data file. It tries to abstract one step further by allowing the programmer to define element types for the basic data field types used in the entire project. This avoids redundancy and saves work, because we only need to define basic behavior once for our entire project, instead of for every single form or grid where a certain type may appear.

Example: A customer number is used in many modern business applications – at least if it's about earning money. So, you're likely to end up with a variety of places in a project where users may enter a customer number, e.g. customer maintenance, order entry, invoice entry, cash receipts, etc.

Thanks to the concept of element types, Barista allows the programmer to define once how the customer id is formatted or entered, no matter where it will occur. I/O masking, basic validation rules, and lookup values are just some of the things used to build the definition of an element type.

In tables, these element types are used like bricks to build a table definition representing data in a database table on the screen:

Forms

Forms are the representation of the table alias definitions on the screen. Variations of forms offered by Barista are maintenance grids, options entry screens, or inquiry grids. They all benefit from definitions of the element types to make any screen consistently behave according to the one-time definition of the business rules of the application.

Table Alias Definitions

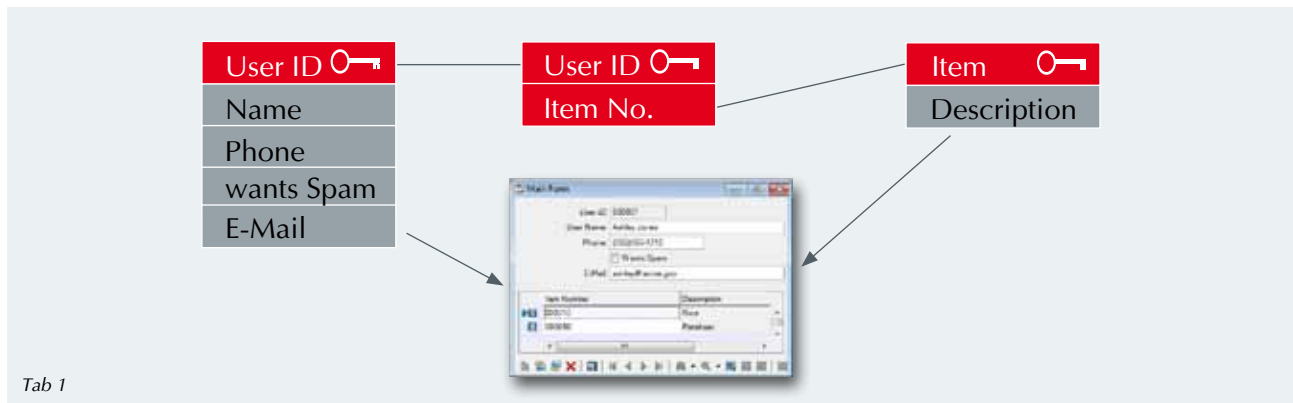
Table Alias Definitions link data, usually residing in a database, with element types. Each column in the database table is associated with an element type through an alias definition. A special case is the options entry alias: element types can be used to build a form that is not displaying data from a database table, but is rather used to collect input data from the user meant to trigger or proceed a workflow.

Element Types

Element types define data elements as basic bricks of a Barista project. They avoid redundancy by allowing the programmer to define everything that is characteristic to a field in one place. This definition is used wherever a data field based on that element type appears throughout your project.

2. Our Project

In your first Barista project, you will create a very simple database with this design:



First Steps

2.1 Start Barista

In this chapter, you will build a simple data driven form using Barista. We'll build on this form in the follow-up chapters. For complete Barista documentation, see the "Getting Started" document.

Select Barista from the BASIS Menu...



Figure 1

...and click the Login button:



Figure 2

The first time you run Barista, it will run through some quick configuration steps. Just click OK through the initial messages and then click „Synchronize“ when it gets to the „Auto Synchronize Barista“ form. This process will just take a minute.

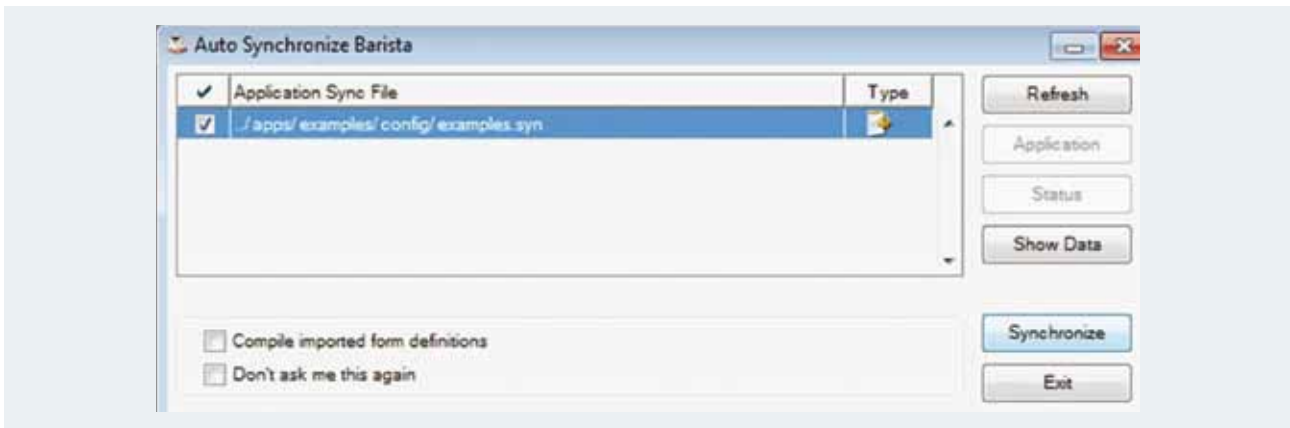


Figure 3

2.2 Create a Barista Application

Barista applications are the “glue” that hold logical parts together. One single Barista application can host a variety of different application products, even from different software vendors. A Barista application is identified by an eight digit company ID issued by BASIS, and a three-letter product ID that is used to differentiate between different application products or modules under the same company ID.

For our example, we will use the company ID “00-000000” and the product ID “TUT” for Tutorial. Always remember to use the product ID (here “TUT”) to identify all element types and tables that belong to your Barista application.

So, let’s create the application by selecting from the Barista Development menu the item “Create Application”. Then fill in the first form of the wizard as follows and press “Next >.”

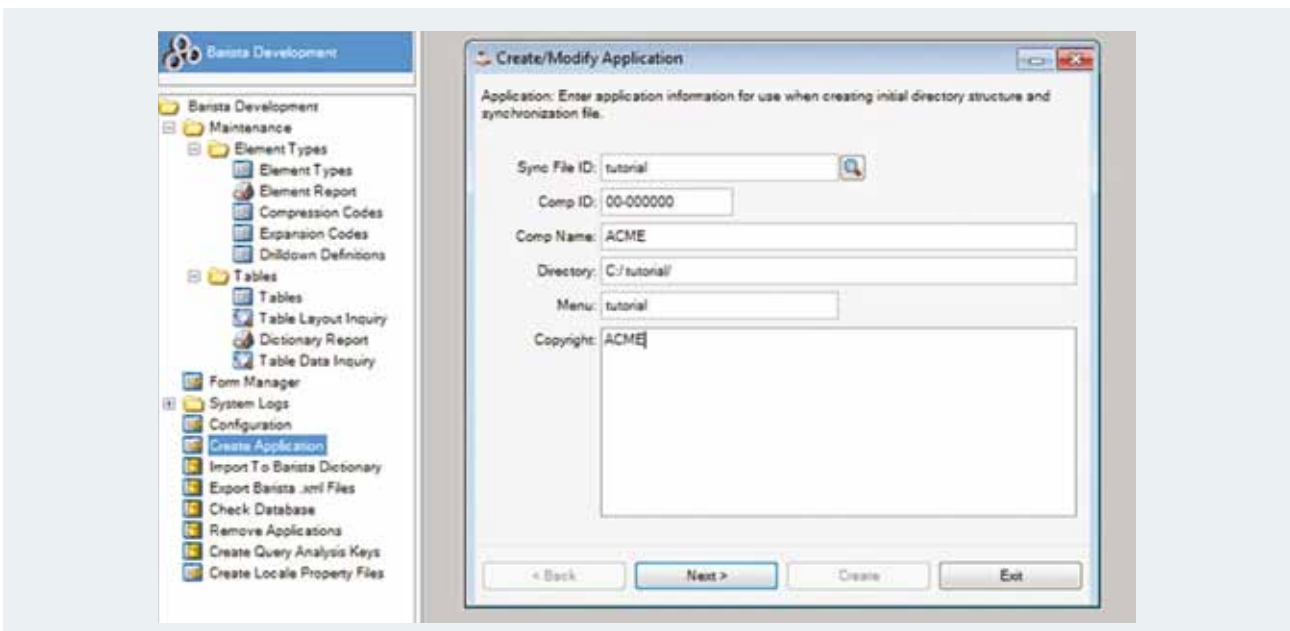


Figure 4

In the next frame, you can choose the languages that are appropriate for your project. Barista offers up to seven working languages from the start from which to choose. When your application is completed, you can translate application text using Barista’s built-in localization tools. For our small project, “ENU” for “English” will suffice:

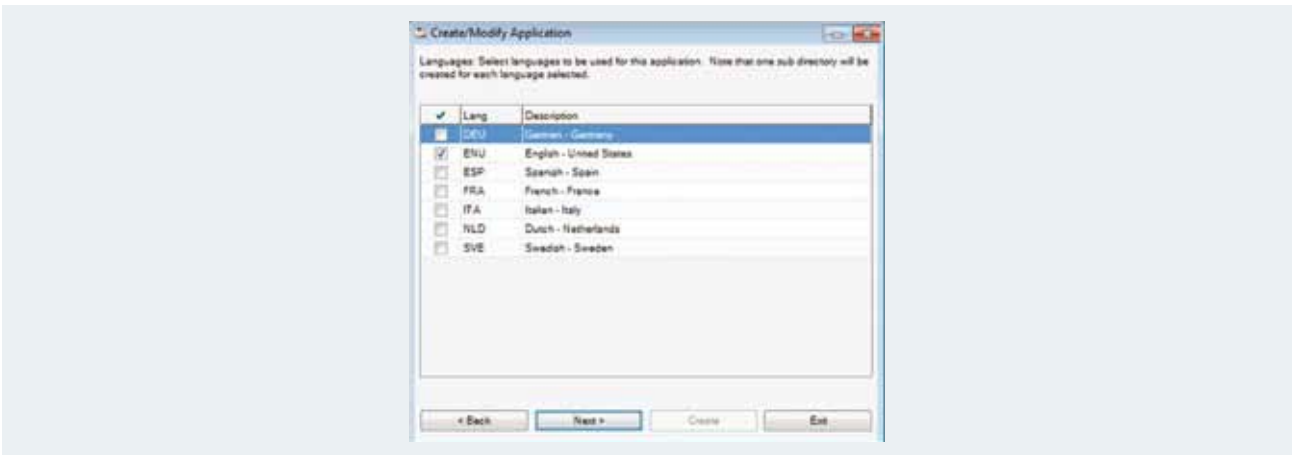


Figure 5

In the next frame, we change the product ID from the default to "TUT".

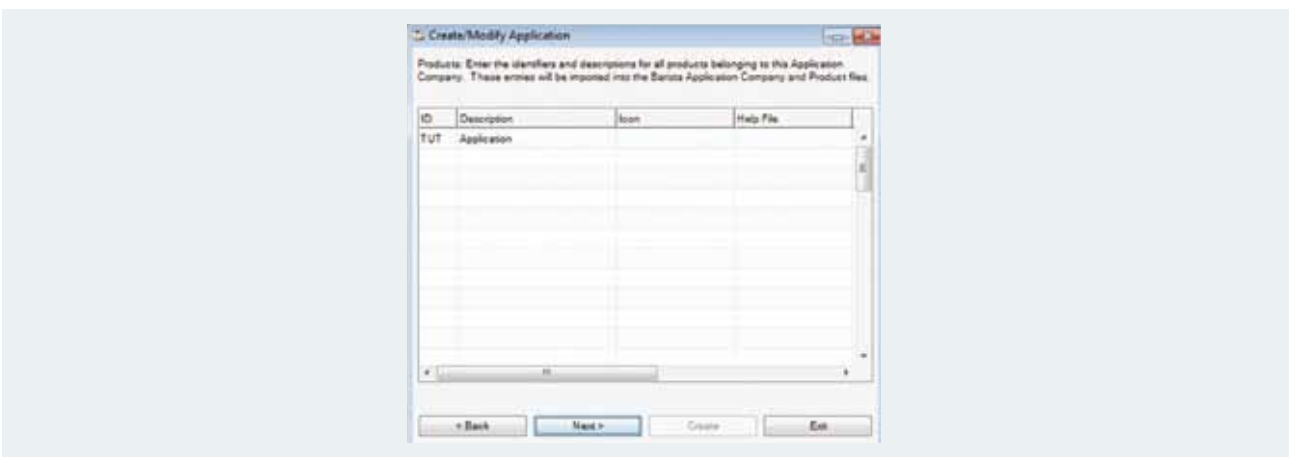


Figure 6

Just confirm the next few frames by clicking the "Next >" button. At the end click "Create App", confirm we want to auto-synchronize the application, and push the "Synchronize" button in the "Auto Synchronize Barista" form:

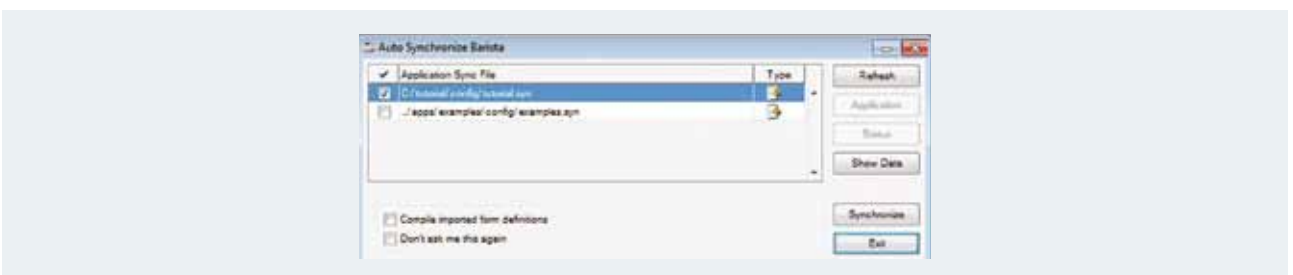


Figure 7

2.3 Build a Simple Form

Our goal is to build this simple file maintenance form for our “Users” table (see Table 1):



Figure 8

To get to that form, we’ll work through these three steps:

- (1) Define the **element types needed for the Users table.**
- (2) Define the **table.**
- (3) Build the **form.**

2.4 Define Element Types

Select the „Barista Development“ Menu option and then select the „Element Types“ node in the menu tree to bring up the “Element Types” maintenance form.

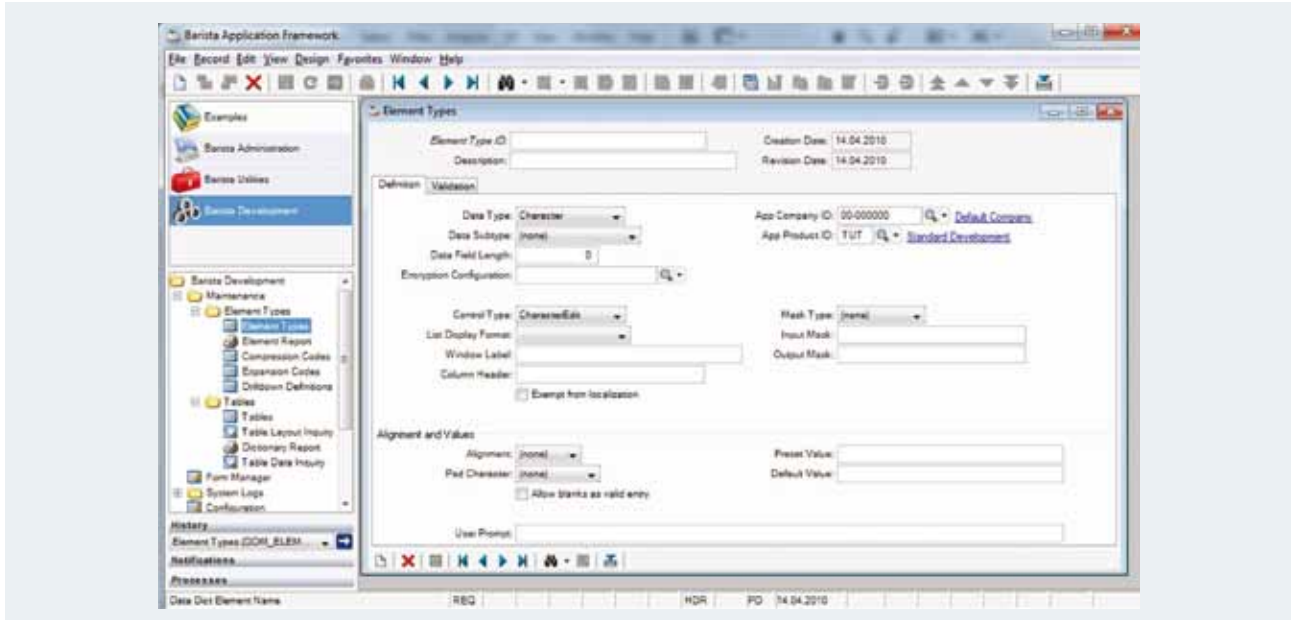


Figure 9

A complete rundown of the fields is available in Getting Started (see p. 55), but we’ll briefly cover some of the more important fields here. The ones we don’t cover here can be jumped over for this project.

In the header:

Element Type ID: A unique 16 character identifier used throughout your Barista application to reference this element.

In the “Definition” tab:

- Data Type: The basic data type that determines how the data will be stored in a file.
- Data Subtype: Can be used to impose a semantic meaning on a field, for example as a YYYYMMDD date string.
- Data Field Length: The length of the field on disk.
- Control Type: Specifies what type of user interface control will be generated to handle data entry.
- Window Label: Specifies the default label for a field of this element type on maintenance forms.
- Input Mask: Data entry mask. (See the Mask Reference section in Getting Started.)
- Alignment/Pad Character: Specifies how the user’s input will be justified and padded before being saved to the file.

In the “Validation” tab:

- Check Box Data: Values corresponding to “Checked” and “Unchecked”.
- Validate Data Table: Data against which input values are validated.
- Data Column: Name of a column in the validation table to be used as a descriptive hyperlink.
- Min/Max Length: Imposes minimum and maximum input length constraints.

We’ll define 5 element types and then add them as data elements (columns) to a table.

Our first element, and the key to the file, is the user ID. We will define it as a **Sequence Counter**. This is useful to automatically generate the next highest ID for a new record.

Add Element Type: TUT_USER_ID

Fill in the following fields of the “Element Types” form with the following data. All other fields can be left as their default values.

Location	Field	Value
Header	Element Type ID	TUT_USER_ID
	Description	User ID
Definition Tab	Data Subtype	Sequence Counter
	Input Mask	000000
	Output Mask	000000
	Alignment	Right (R)
	Pad Character	Zero (30)
	App Product ID	TUT

Tab 2

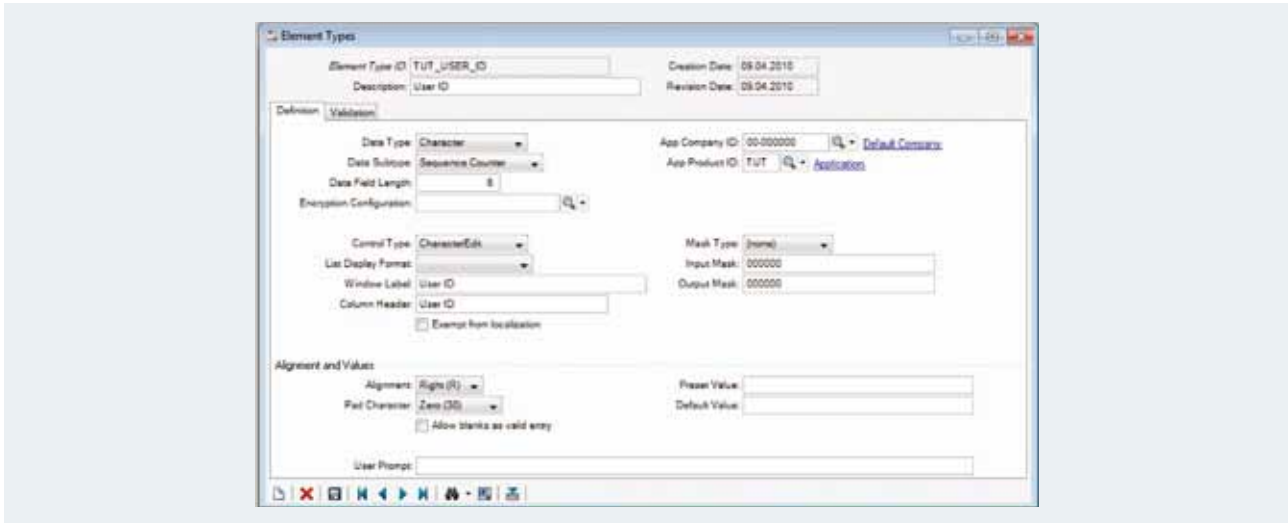



Figure 10

Click  or press CTRL+S to save this element type.

Add Element Type: TUT_USER_NAME

Click  or press CTRL+N to begin entering a new element type.

Location	Field	Value
Header	Element Type ID	TUT_USER_NAME
	Description	User Name
Definition Tab	Data Field Length	30
	App Product ID	TUT

Tab 3

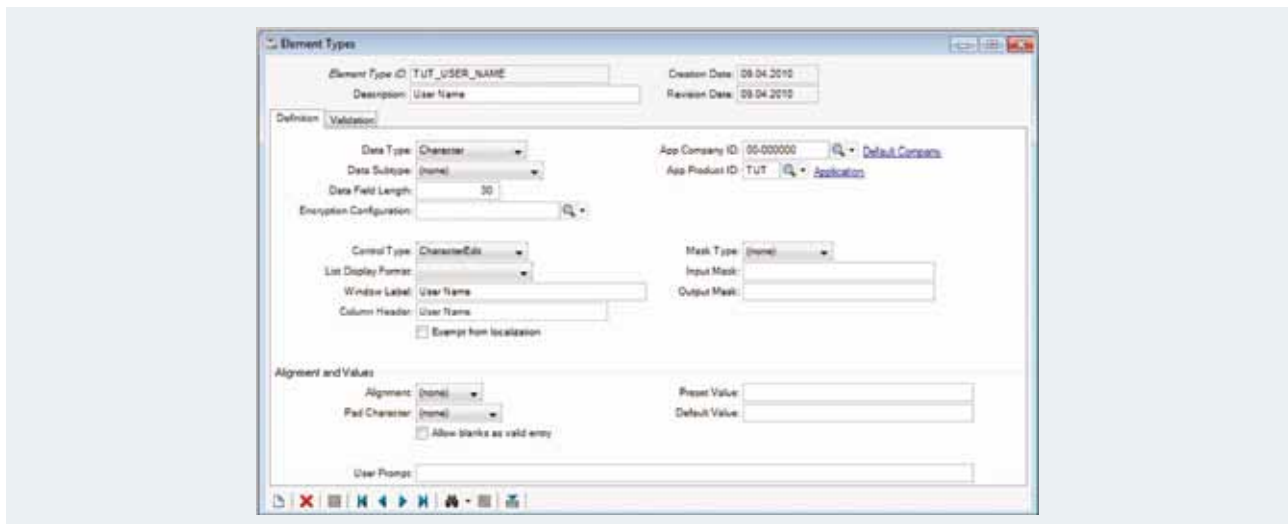



Figure 11

Click  or press CTRL+S to save this element type.

Add element type: TUT_USER_PHONE

Click  or press CTRL+N to begin entering a new element type.

Tab 4

Location	Field	Value
Header	Element Type ID	TUT_USER_PHONE
	Description	Phone
Definition Tab	Data Field Length	10
	App Product ID	TUT
	Input Mask	(000)000-0000
	Output Mask	(000)000-0000
Validation Tab	Minimum Length	10
	Maximum Length	10

- This example uses the North American phone number layout standard to demonstrate masking and input length validation.

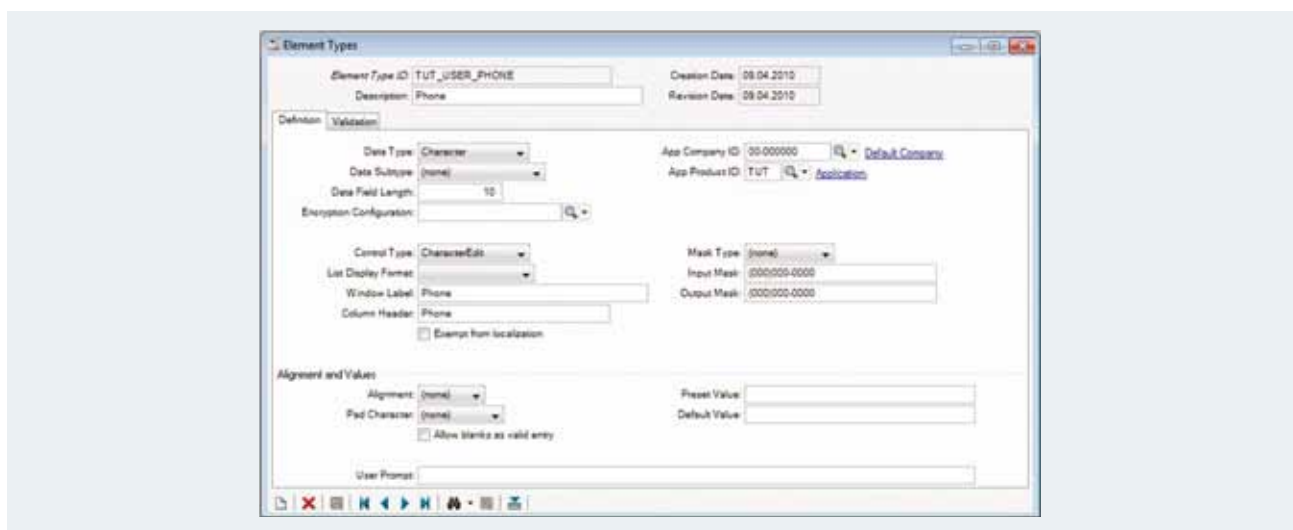


Figure 12

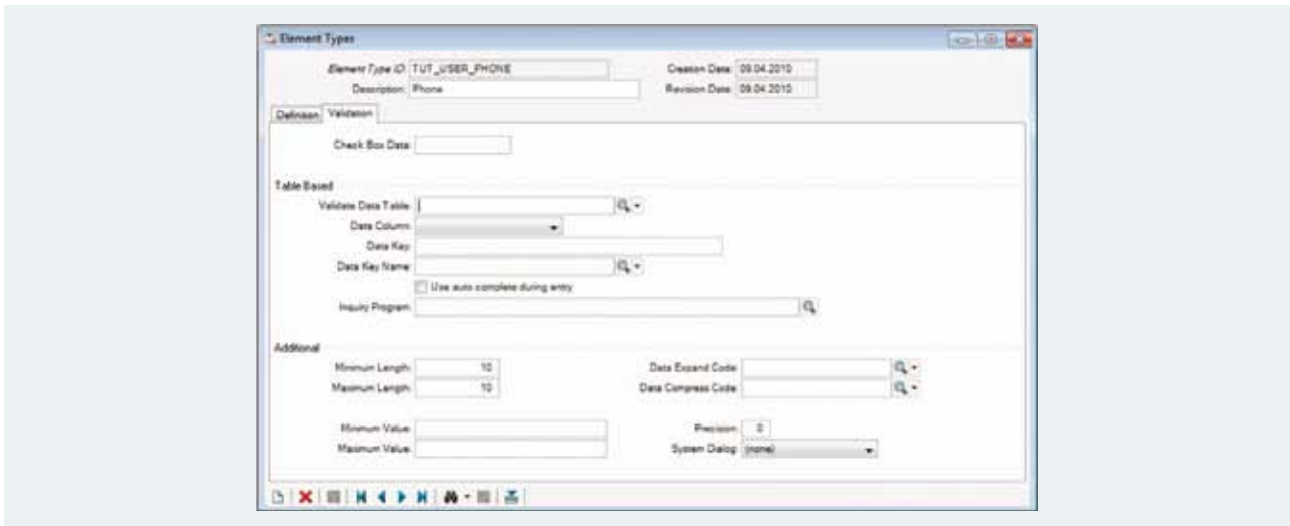



Figure 13

Click  or press CTRL+S to save this element type.

Next, we'll add a checkbox in which we will mark whether a person has allowed us to send news via email.

Add Element Type: TUT_WANTS_SPAM

Click  or press CTRL+N to begin entering a new element type.

Location	Field	Value
Header	Element Type ID	TUT_WANTS_SPAM
	Description	Wants Spam
Definition Tab	Data Field Length	1
	Control Type	CheckBox
	Column Header	Spam
	App Product ID	TUT

Tab 5

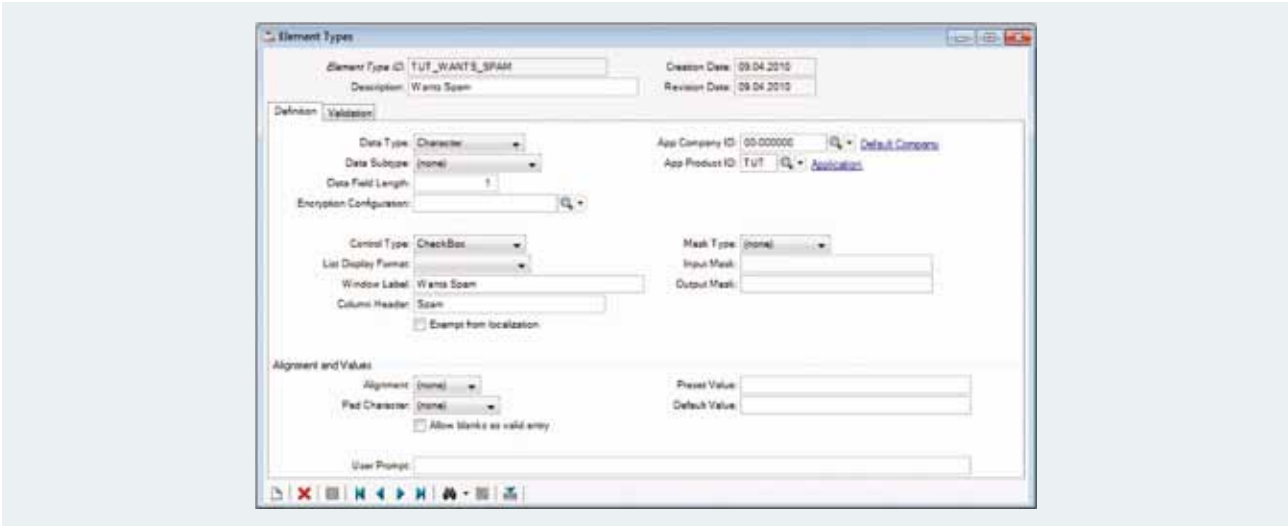



Figure 14

Behind the scenes, Barista has set the **Check Box Data** on the Validation tab to “Y;N”, which means the value “Y” should be set if the checkbox is checked, otherwise “N”. For 1 digit numeric fields, Barista will default the **Check Box Data** to “1;0”. Any checked/unchecked values may be used.

Click  or press CTRL+S to save this element type.


Add Element Type: TUT_USER_EMAIL


Click  or press CTRL+N to begin entering a new element type.

Location	Field	Value
Header	Element Type ID	TUT_USER_EMAIL
	Description	Email
Definition Tab	Data Field Length	30
	App Product ID	TUT

Tab 6

Figure 15

Click  or press CTRL+S to save this element type.

Now we've entered our 5 elements. Click  or press CTRL+Q to query the records for the current form. Enter a search string of TUT to only see records that start with "TUT".

Element Type	Desc	Create Date	Rev Date	DT type	ST type	Dat. En.	CT type	Win Label	Comp ID	Prod.	Input Mask	Output Mask	Pad	FacC
TUT_USER_EMAIL	E-Mail	09.04.2010	09.04.2010	Character (C)		30	CharacterEdit (E)	E-Mail	00-000000	TUT				
TUT_USER_ID	User ID	09.04.2010	14.04.2010	Character (C)	Sequence Counter (S)	8	CharacterEdit (E)	User ID	00-000000	TUT	000000	000000	Right Zero	
TUT_USER_NAME	User Name	09.04.2010	09.04.2010	Character (C)		30	CharacterEdit (E)	User Name	00-000000	TUT				
TUT_USER_PHONE	Phone	09.04.2010	09.04.2010	Character (C)		10	CharacterEdit (E)	Phone	00-000000	TUT	000000-0000	000000000000		
TUT_WANTS_SPMAT	Wants SPMAT	09.04.2010	09.04.2010	Character (C)		1	Character (C)	Wants SPMAT	00-000000	TUT				

Figure 16

Close the Element Types form. The next step is to define a table containing these element types as data elements (columns).

2.5 Define a Table

Select the “Barista Development” Menu option and click the “Tables” node in the menu tree to bring up the Tables entry form.

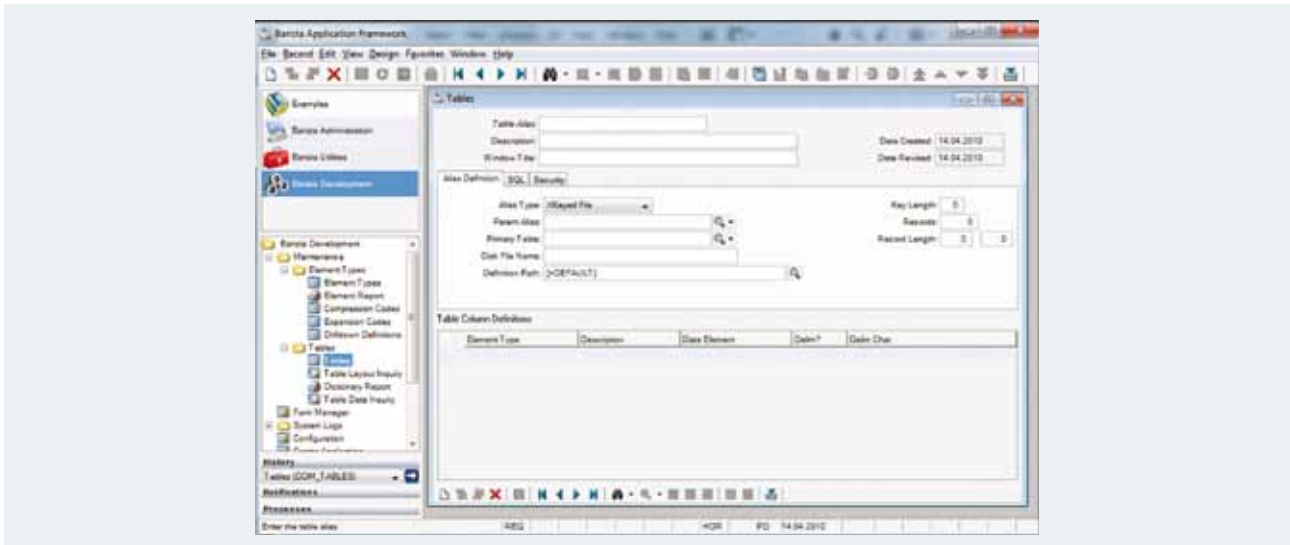


Figure 17

2.6 Using the Tables Form

A complete rundown of the fields in the Tables form is available in Getting Started, but we’ll briefly cover some of the more important fields here.

In the header:

Table Alias: A unique 16 character identifier used throughout your Barista application to reference the table.

Window Title: This defines what appears in the title bar of the form which corresponds to the table.

In the Alias Definition tab:

Alias Type: Specifies the type of file. In some cases, the alias will define a reference to a set of data, as opposed to a physical disk file. The default alias type is the BBJ XKEYED file type. For a file that will contain data that varies significantly in size (for example a file that contains BLOBs), the VKEYED data type would be a better choice.

Disk File Name: Defaults to the table alias name, converted to lowercase with an extension of „.dat“.

Definition Path: The data directory. It defaults to [+DEFAULT], which resolves to “../apps/default/data/”.

Here, Barista expects you to type or choose the global variable (STBL) which points to the directory where the data of your Barista application are stored. Usually, this is the “Sync File ID” established during application creation, but in capital letters and between square brackets. Enter or choose [TUTORIAL].

Click on the "Security" tab and ensure that the company and product IDs are 00-000000/TUT.

The detail grid at the bottom of the screen contains the list of data elements or columns in the table.

Add Table: TUT_MAIN_FORM

Location	Field	Value
Header	Table Alias	TUT_MAIN_FORM
	Description	Main Form
Column Definitions	Element Type	Data Element
	TUT_USER_ID	TUT_USER_ID
	TUT_USER_NAME	TUT_USER_NAME
	TUT_USER_PHONE	TUT_USER_PHONE
	TUT_WANTS_SPAM	TUT_WANTS_SPAM
	TUT_USER_EMAIL	TUT_USER_EMAIL

Tab 7

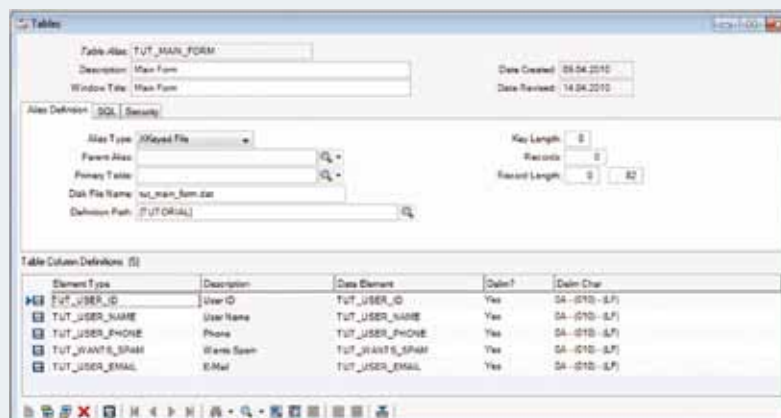


Figure 18

To switch between the detail grid and header, press **F7**. The status bar will display HDR or DTL accordingly.

Enter the table columns as shown. When finished, press **F7** to switch to the header.

In HDR status, click  or press **CTRL+S** to save the table definition.

A table must have at least one key, the primary key. We must define a primary key for our new table.

Define Keys for TUT_MAIN_FORM

Click  or press CTRL+O, then select **Key Definitions** from the dropdown **Options menu**.

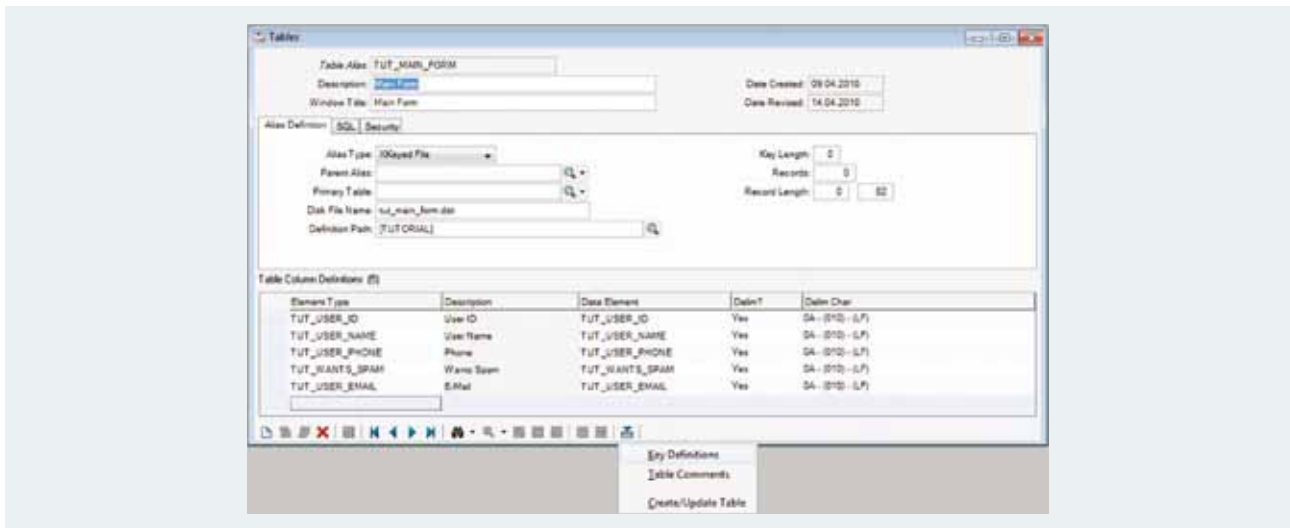


Figure 19

Set the Key Name and Key Description to **PRIMARY** and **Primary**, respectively, and add **TUT_USER_ID** to the Grid. Key number 00 is considered the unique primary key by default and therefore does not have to be flagged extra as unique. The key name and description are free to name as you wish. We've use "Primary" just for convenience.

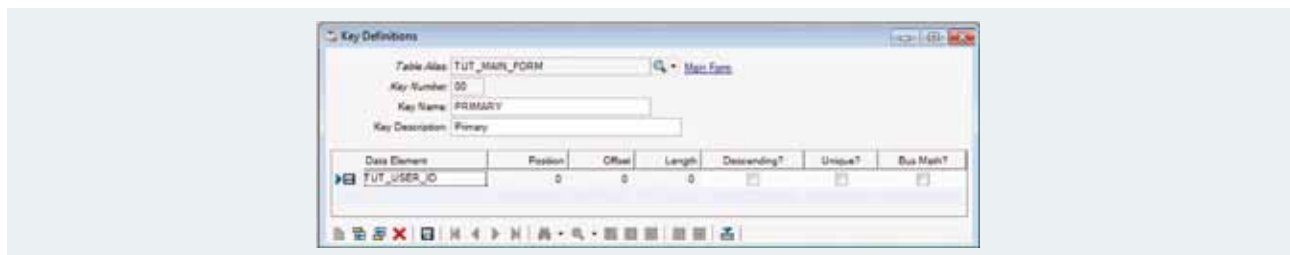



Figure 20

Press F7, if necessary to switch to the header and click  or press CTRL+S to save the key definition. Close the Key Definitions dialog by clicking the close box.

2.7 Create the Table

Click  or press CTRL+O, then select **Create/Update Table** from the dropdown Options menu.



Figure 21



Figure 22

Click the **Update** button.

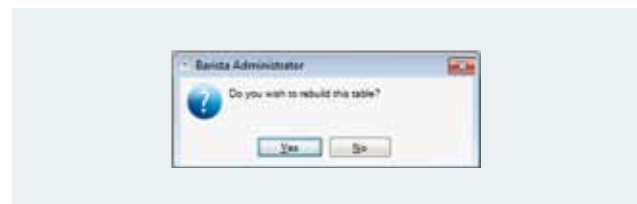


Figure 23

Click the **Yes** button. After a few seconds, the following completion dialog should appear.

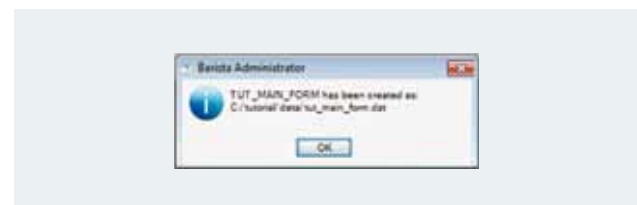



Figure 24

Click **OK**.

Close the table maintenance form. The final step is to build a **Form** based on this table.

Build a Form

2.8 Using the Form Manager

Load the Form Manager by clicking  or pressing F8 or choosing “Form Manager” from the “Barista Development” menu.

Click on „Tutorials“ in the tree on the left to restrict the display to just the tutorial form.

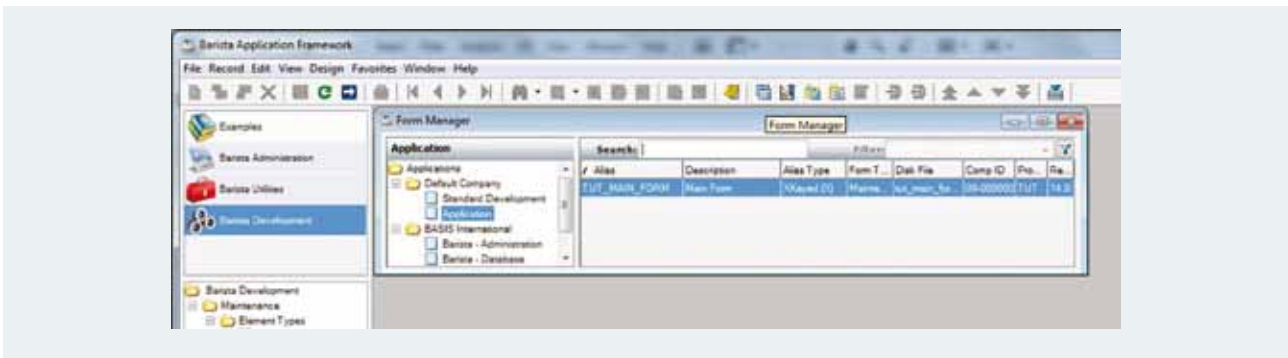



Figure 25

2.9 Build the Form

Select **TUT_MAIN_FORM** and click  or press CTRL+B or choose “Build Object” from the “Design” menu. This will automatically build the form in the default design provided by Barista.

2.10 Run the Form


Click  or press F5 or choose “Run Process” from the “Record” menu to run your form.



Figure 26

Experiment with using the form to enter and view data in your table.

In addition to entering data into the fields of the form, the user of a Barista developed application will use the supplied toolbar to navigate, save, create, and search the records of our table. (All buttons have been enabled here for illustration).

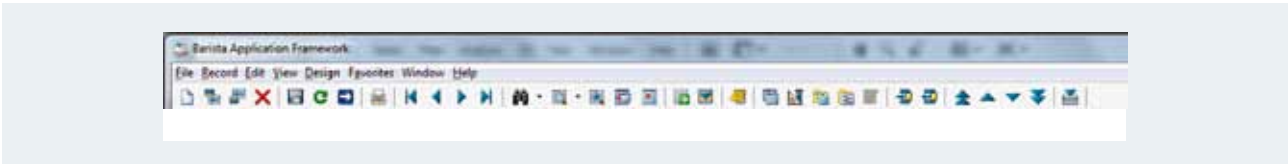











Figure 27

The following tool buttons are used most often:

	New Record (CTRL+N)
	Delete Record (CTRL+D)
	Save Record (CTRL+S)
	Refresh Data (ALT+F5)
	First Record (ALT+PGUP)
	Previous Record (PGUP)
	Next Record (PGDN)
	Last Record (ALT+PGDN)
	Record Query (CTRL+Q)

Tab 8

Once you have entered some records, click  or press CTRL+Q:

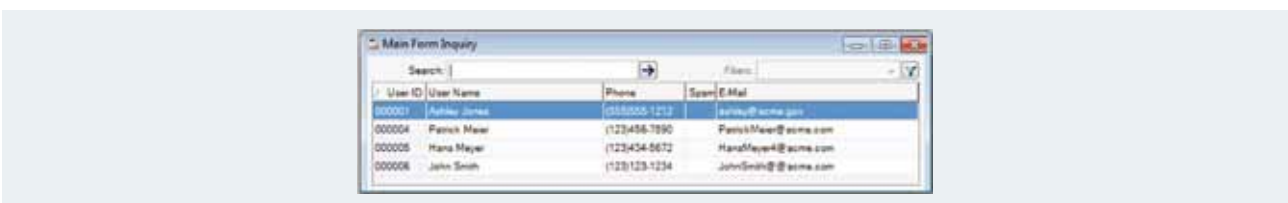



Figure 28

That's it! You've built and tested your first Barista form. The remaining chapters all build on this form.

3. Customize the Form

3.1 Load the Form Manager

Load the Form Manager by clicking  or pressing F8 or choosing “Form Manager” from the “Barista Development” menu.

Double-click TUT_MAIN_FORM to load it in the **Form Designer**.

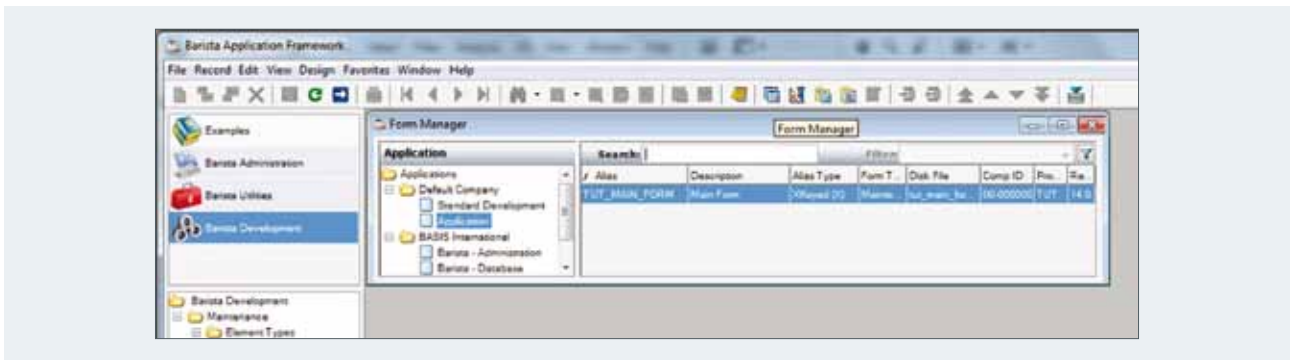


Figure 29

On the left of the Form Designer is a list which contains the columns and the form itself

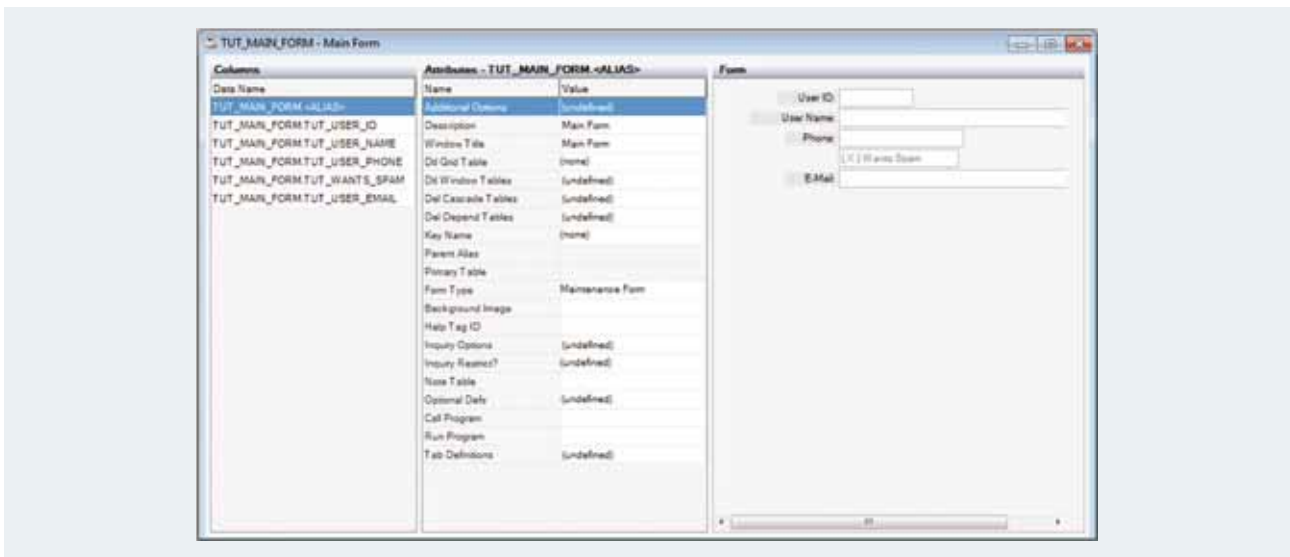



Figure 30

(the “.<ALIAS>” entry). Selecting items here displays the corresponding attributes in the center area. Double click the values of the attributes to edit them. On the right of the Form Designer is a display of what the form will generally look like. You can select or rearrange items on the form from here.

Many aspects of a field’s appearance and behavior can be modified via the Attributes column. There are two ways to pick the field you want to modify: select the field in the list on the left or

click the field in the form on the right. For modifying components of the form overall, click **TUT_MAIN_FORM.<ALIAS>** in the columns list.

The available attributes change depending on what element you've selected. For example, select

TUT_USER_ID, change the Control Label from „User ID“ to „User Number“, then click  or press ALT+F5 to refresh the form without saving.

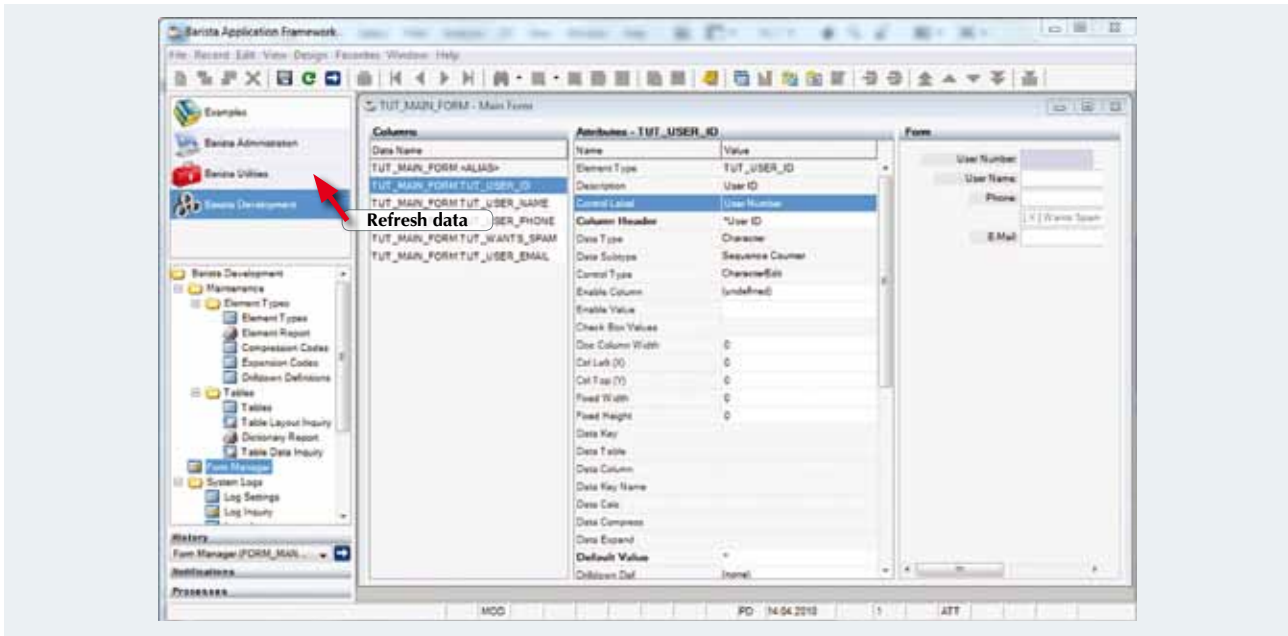



Figure 31

3.2 Changing a Control's Location or Tab Order

To change the location of a control, you can a) use the up/down arrows  on the toolbar, or b) drag and drop columns within the columns list, or c) drag and drop the controls to the desired position on the graphical form panel.

Barista uses a relative positioning system, so moving the WANTS_SPAM field will cause everything below it to move relative to the new position of the WANTS_SPAM field.

Using arrow keys

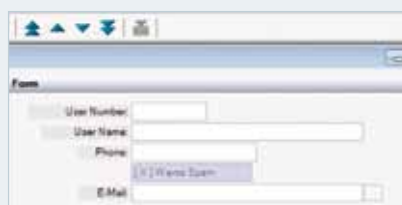


Figure 32.1

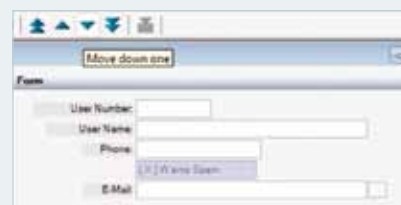


Figure 32.2

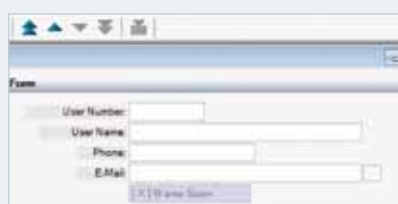


Figure 32.3

Drag and drop in the graphical form panel

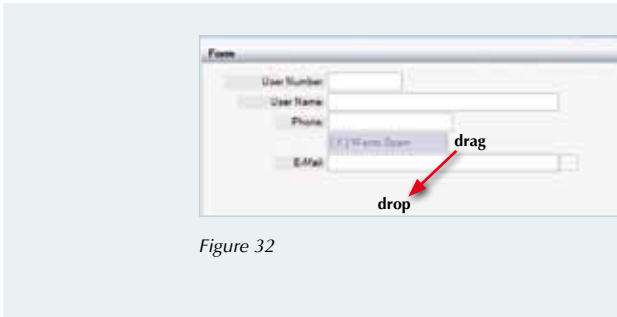


Figure 32



Figure 33

Dragging one control directly onto another control will cause a prompt for confirmation whether this move should be reflected in the tab (key) order sequence.



Figure 34

Answering **Yes** inserts the control being moved before the control on which it is placed in the tab order.

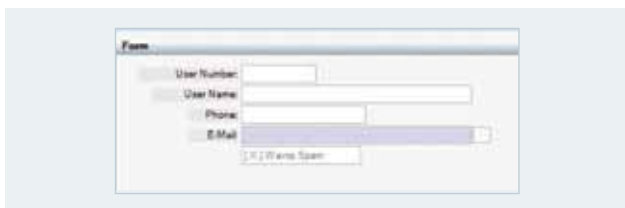


Figure 35

Answering **No** to the prompt causes the control to be moved on top of the other control without changing the tab order in the columns list.

Repositioning columns can also be accomplished by dragging and dropping columns within the "Data Name" area on the left of the Form Designer. Doing so, will be reflected in the tab order automatically.

This re-ordering within the Form Designer has no effect on the data in the underlying file.

3.3 Adding an Optional Definition to a Column

You can set additional options for a column.

Example: Click on the Email column and scroll down its attributes until you see “Optional Defs”. Double-click on the “(undefined)” (or possibly “(modified)”) value to bring up the list of optional definitions.

Scroll as needed and select the “Control contains external link” option. After rebuilding the form, Barista will analyze values entered into the field and try to run the appropriate associated program to handle the field content.



Figure 36

Save changes by clicking  or pressing CTRL+S.

3.4 Restoring Form's Original Layout

To discard some or all changes, select “Clear Formatting Attributes” from the “Design” menu.

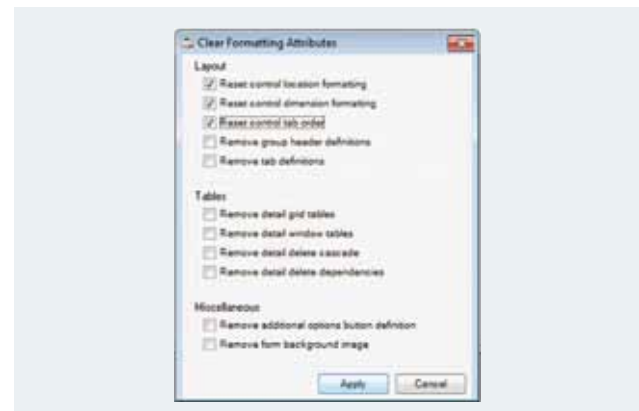



Figure 37

Run the Form

To run the form, click  or press F5 or choose “Run Process” from the “Record” menu.

Extending the Basic Form with Callpoints and Additional Options

In this chapter, you will extend the form you created by adding your own custom code and options buttons, using Barista's "Callpoint" and "Additional Options" features. You will have to handle some BBJ source code, but we'll supply you with code examples to paste. There is no BBJ knowledge needed yet.




Figure 38

3.5 Callpoints

With Barista you can essentially create and customize much of your application without using any program code. However, there are times when you may need to add business-specific logic to the application. Whether it is performing advanced validation, executing auxiliary back-end programs, or starting a foreign application, Callpoints provide you appropriate places to develop and run custom code.


3.6 Additional Options

Additional options can be added to the  button, as individual buttons on the form's toolbar, or both. Developer-defined callpoint code determines what a given option item will do.

See Getting Started for more detailed background information about Callpoints and Additional Options.

3.7 Bring up the Callpoint Editor

To open the Callpoint Editor, open the Form Manager, select TUT_MAIN_FORM, and

click  or press F2

or right-click on the highlighted selection and select "Callpoint Editor".

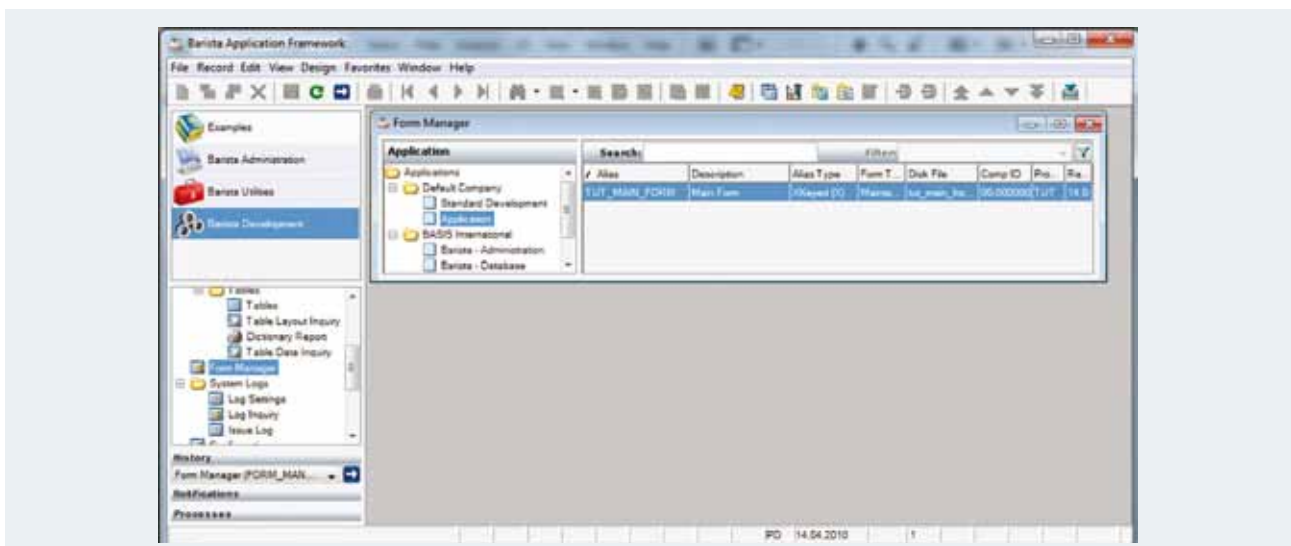


Figure 39

3.8 Callpoint Editor

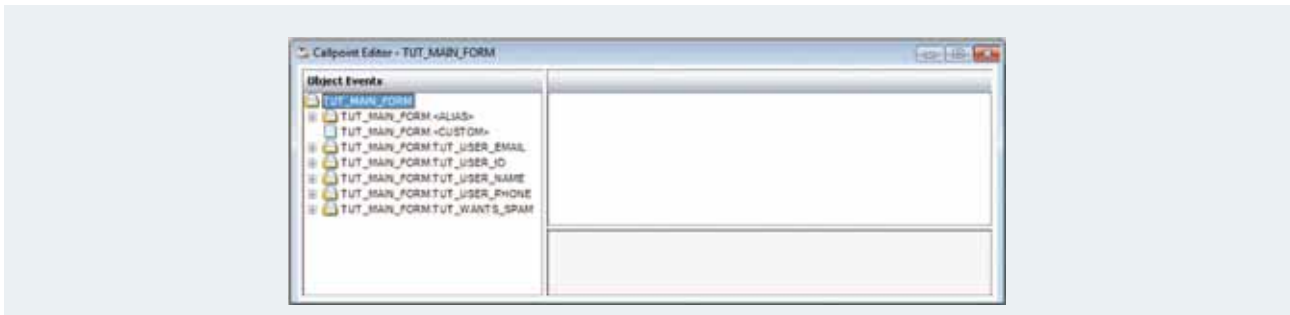





Figure 40

The callpoint editor consists of three panels. On the left is a tree which contains the form, its columns and their associated callpoints. At a glance one can tell if there is code in a given callpoint based on the icon ( = empty,  = non-empty). The upper right panel displays the code for the selected callpoint. The lower right panel displays the compilation status and other status info.

When a callpoint is selected, right clicking the upper-right panel (or clicking ) displays a list of code templates provided by Barista, in order to serve as a quick reference and to save you some typing. We will give you two examples how these prefabricated code snippets can make your life easier.

First, we will use the **After Column Validation (AVAL)** callpoint to generate a default email address derived from the contents of the user name column (field).

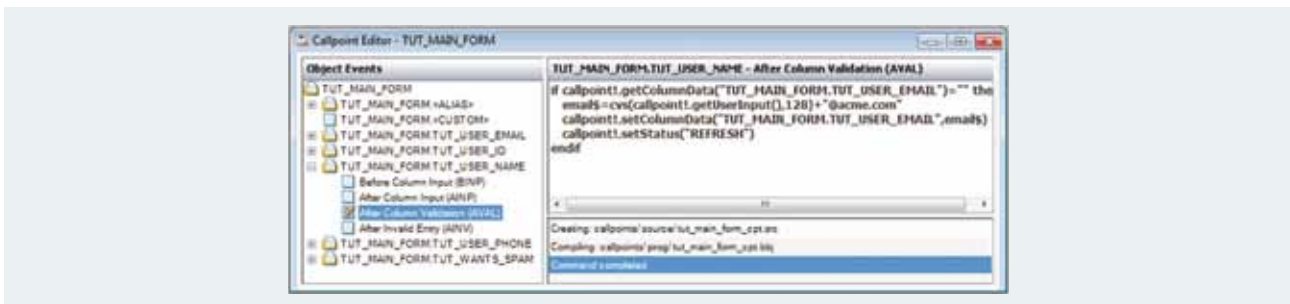



Figure 41

- Bring up the Callpoint Editor for TUT_MAIN_FORM.
- Select “After Column Validation (AVAL)” under TUT_MAIN_FORM.TUT_USER_NAME.
- Enter (or paste) the following BBj code (note we don’t change an existing email value):

```
if callpoint!.getColumnData („TUT_MAIN_FORM.TUT_USER_EMAIL“)=“” then
  email$=cvs(callpoint!.getUserInput(),128)+“@acme.com”
  callpoint!.setColumnData („TUT_MAIN_FORM.TUT_USER_EMAIL“,email$)
  callpoint!.setStatus („REFRESH“)
endif
```

- Click  or press F5 to run the form. If there is a problem compiling the callpoint code, then you will be notified and asked if you wish to continue. If this should happen, click “No” and recheck your code.

Note that running a form while in the Form Designer or Callpoint Editor automatically saves and rebuilds the form, as opposed to running a form from the Form Manager overview, which does not rebuild the form first.

- Add a new record and note the email is automatically populated as you tab out of the “User Name” field:



Figure 42

3.9 Add After Column Input (AINP) Callpoint Code for TUT_USER_EMAIL

Barista provides several basic validation mechanisms, such as enforcing a minimum length, or performing a foreign key lookup. There are times, however, when you need to perform more complicated validation. The After Column Input (AINP) callpoint is called after the user has attempted to leave the field, but before validation. In our second example for using a callpoint, we will check for an obviously invalid email address.

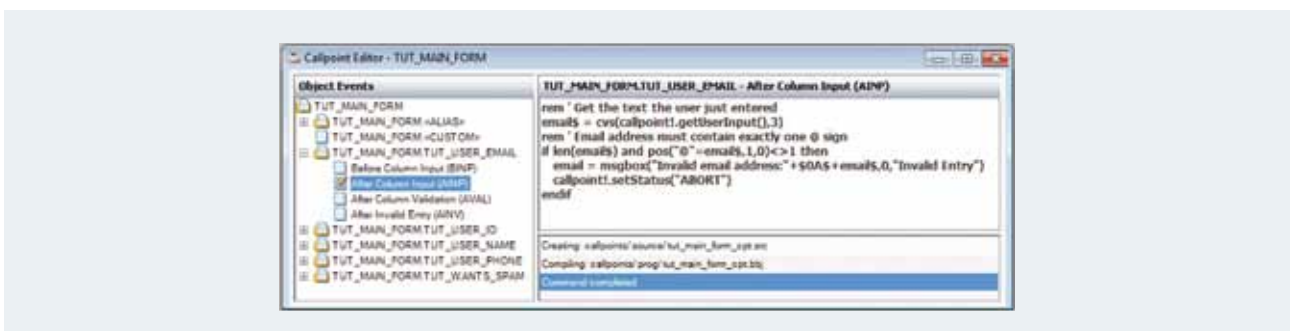


Figure 43

- Select After Column Input (AINP) under TUT_MAIN_FORM.TUT_USER_EMAIL.
- Add the following code:

```
rem ' Get the text the user just entered
email$ = cvs(callpoint!.getUserInput(),3)
rem ' Email address must contain exactly one @ sign
if len(email$) and pos(,"@"=email$,1,0)<>1 then
    email = msgbox(„Invalid email address:“+$0A$+email$,0,„Invalid Entry“)
    callpoint!.setStatus(„ABORT“)
endif
```


- Click  or press F5 to run the form.
- Attempt to enter an email address with no or more than one @ sign:



Figure 44

Options Entry Form

There are times you would like the user to enter data, that are not meant to be written in a database table, but for triggering service or batch processes. In this chapter, we will demonstrate how you can use Option Entry forms to write a time-stamped email address and comment to a text file and to call a text editor or other custom program. This same basic approach can be used to send email, write reports, etc., from your Barista application.

We'll start by defining a table alias that will be used to generate an Options Entry form.

Open Tables Maintenance and define the following alias:

Location	Field	Value
Header	Table Alias	TUT_DATA_ENTRY
	Description	Comment Form
	Alias Type	Options Entry
Security	Product ID	TUT
Column Definitions	Element Type	Data Element
	EXM_TEXT_30	EMAIL
	EXM_TEXT_30	COMMENT

Tab 8

(If you didn't install the Barista Examples application, you'll need to first define EXM_TEXT_30 as a 30-character text field in Element Types.)

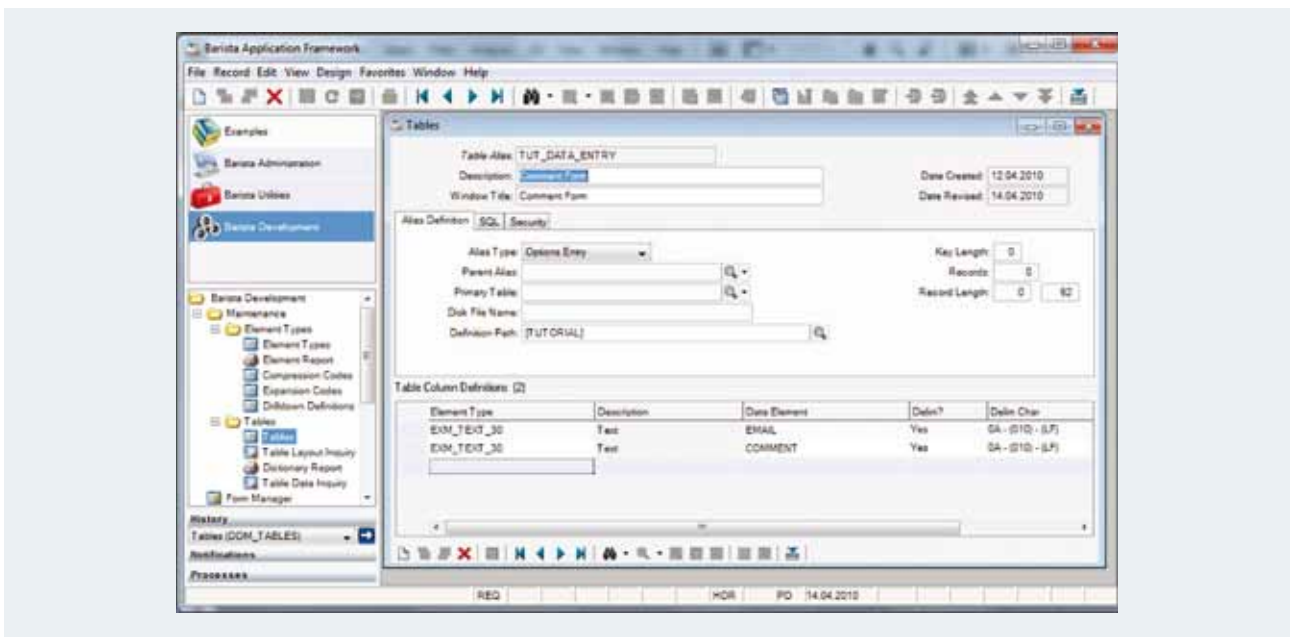



Figure 45

Click  or press CTRL+S to save this record and then close the Tables form.

Run or switch back to the Form Manager and click  or press ALT+F5 to refresh the list of aliases, if necessary. Double-click on your newly defined alias to load it into the Form Designer.

- Mark TUT_DATA_ENTRY<alias>, double-click on the line “Optional Defs”, and check the option “Hide saved selections options”.

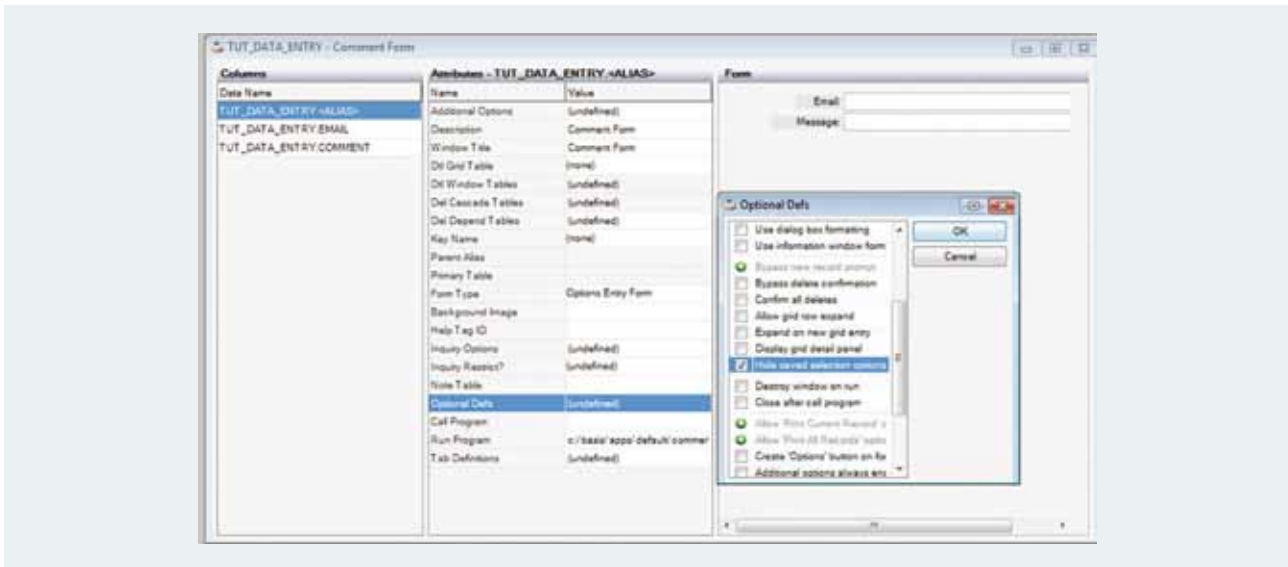




Figure 46

- Set the Control Label for the EMAIL field to „Email“
- Set the Control Label for the COMMENT field to „Message“
- Set the **Run Program** attribute for the alias to `../apps/default/comment.src` and save the following program to this file under the BASIS home directory. On Windows, the full name would typically be `C:\Program Files\basis\apps\default\comment.src`.

```

email$ = option!.getOptionData(„EMAIL“)
comment$ = option!.getOptionData(„COMMENT“)
timestamp$ = new java.util.Date().toString()
message$ = timestamp$+$09$+email$+$09$+comment$
filename$ = „../apps/default/comment.txt“
string filename$,err=*next
comment = unt
open (comment,MODE=„O_APPEND“)filename$
print (comment)message$
close (comment)
release

```

Click  or press F5 to run the form, type any values into the entry fields, then click  or press F5 again to pass control to the overlay program.

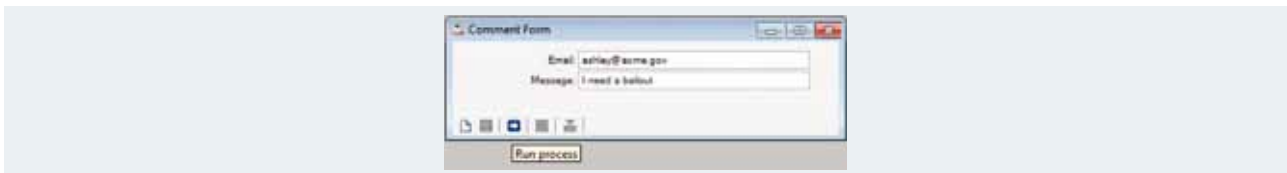


Figure 47

The log file `comment.txt` should contain an entry that looks something like this:

```
C:\Program Files\basis\apps\default>type comment.txt
Mon Jan 12 21:00:21 PST 2009      suggestions@whitehouse.gov
I need a bailout!
```

Adding an Option Button to Display the TUT_DATA_ENTRY Form

Now we'll add a button to TUT_MAIN_FORM to display our new form.


Load **TUT_MAIN_FORM** in the Forms Designer.

Double-click the **Additional Options** attribute for the alias and enter:

Description	Code	Location
Comment	COMM*	Menu And Form**

Tab 9

* Any four character code will do.

** We can choose to add the option as a button on the form, as an item in the options menu () or both.

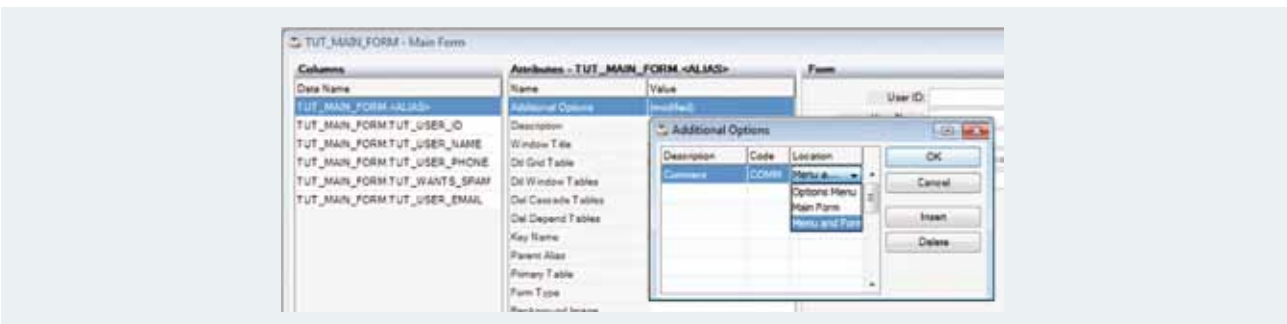


Figure 48


Click  or press F5 to run the form. It has a new button, but the button



Figure 49

doesn't do anything yet.

Next, we'll add code to invoke the TUT_DATA_ENTRY option entry form when the user clicks the **Comment** button.

Add Callpoint Code for the Option Item

Still in the Form Designer, right-click on the form or press F2 to bring up the Callpoint Editor.

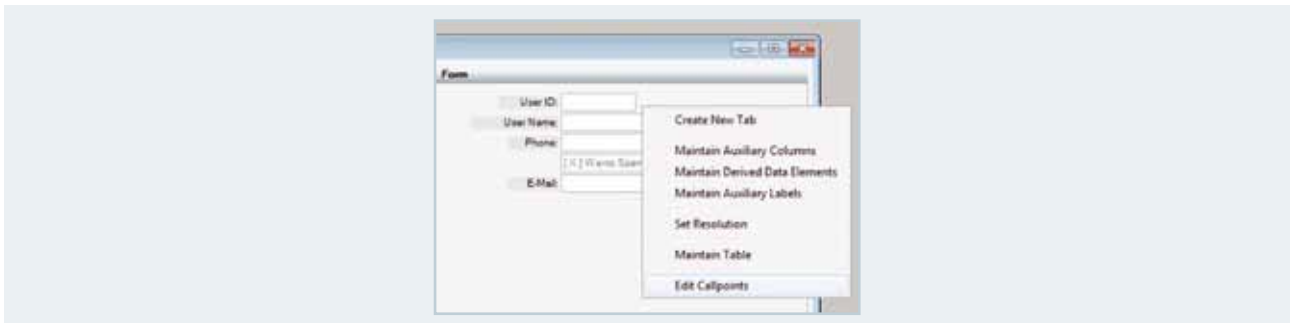


Figure 50

Under TUT_MAIN_FORM.<ALIAS>, select **After Option Select (AOPT)**, then select **Comment (COMM)** and paste in the following callpoint code:

```
user_id$ = stbl(„+USER_ID“)
email$ = callpoint!.getColumnData(„TUT_MAIN_FORM.TUT_USER_EMAIL“)
dim dflt_data$[1,1]
dflt_data$[1,0] = „EMAIL“
dflt_data$[1,1] = email$
call stbl(„+DIR_SYP“)+„bam_run_prog.bbj“, „TUT_DATA_ENTRY“,user_id$, „“, „“, table_
chans$[all], „“, dflt_data$[all]
```

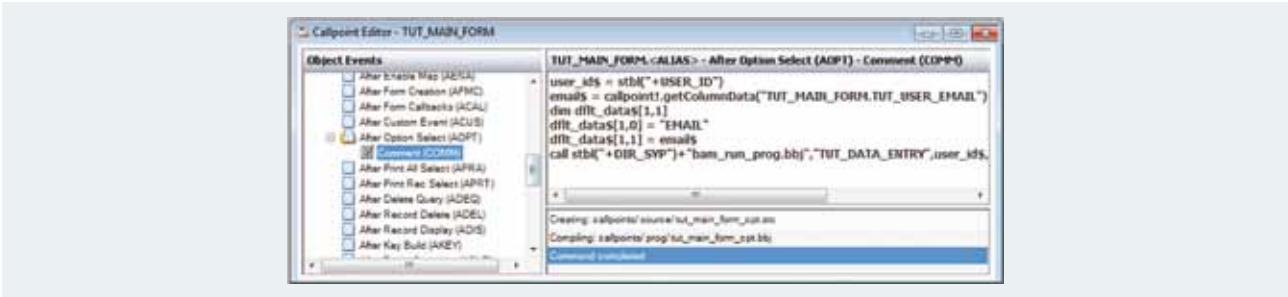


Figure 51



Click  or type CTRL+B to compile the callpoints and check for errors. Click  or press F5 to run the form. Now, when you display a record and click the Comment button, the overlay program should come up with the email field pre-filled:



Figure 52



Figure 53

By default, options buttons are only enabled when a record is being displayed and after having saved any changes. This can be changed via the **Additional options always enabled** setting under **Optional Defs** for the alias.

In our next example, you will add an option to call up a text editor via the Additional Options attribute in the Form Designer for the TUT_MAIN_FORM alias.

Tab 10

Description	Code	Location
Editor	EDIT	Menu and Form

Add callpoint code for Editor (EDIT) under After Option Select (AOPT) in the Callpoint Editor:

```
client$ = bbjapi().getThinClient().getClientOSName()
editor$ = „gedit“; rem , Adjust as necessary for UNIX
if client$ = „Mac OS X“ then editor$ = „open -a TextEdit“
if pos(„Windows“=client$)=1 then editor$ = „notepad.exe“
bbjapi().getThinClient().clientExec(editor$)
```

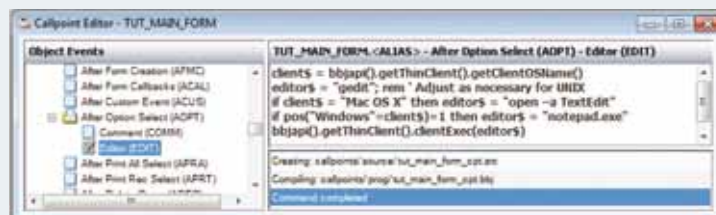


Figure 54


Click  or press F5 to run the form, jump to a random data record and try out the new **Editor** option menu item.



Figure 55

3.10 Calling a custom program via an Option Button

In our third example, you will be shown how to call a simple custom BBJ program via an options button.

Save the following program as `../apps/default/sample.src`:

```

sysgui = unt
open (sysgui) "X0"
sysgui! = bbjapi().getSysGui()
window! = sysgui!.addWindow(100,100,200,200,"Sample")
window!.setCallback(window!.ON_CLOSE,"eoj")
button! = window!.addButton(101,50,50,100,30,"Click",$$)
button!.setCallback(button!.ON_BUTTON_PUSH,"click")
process_events
click:
i=msgbox(„Click“)
return
eoj:
release
    
```

Add another option via the **Additional Options** attribute in the Form Designer for `TUT_MAIN_FORM`.

Description	Code	Location
Sample	TEST	Menu and Form

Tab 11

Add the following for **Sample (TEST)** under **After Option Select (AOPT)** in the Callpoint Editor:

```
i=scall(„bbj ../apps/default/sample.src“)
```

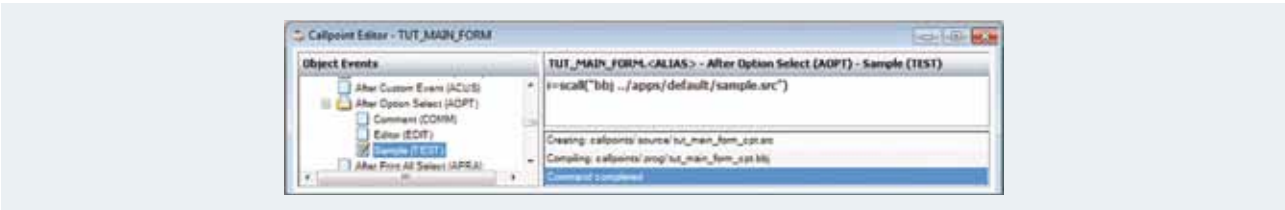


Figure 56


Click  or press F5 to run the form and try out the Sample options menu item:



Figure 57



Figure 58

As you have seen, Barista automatically takes care of the basics of building standard data entry forms from element types and tables you have defined. Callpoints take over from there to enable the developer to customize Barista applications as necessary.

3.11 Setting up the Spam section

Here, we will check whether our user wants to receive emails. Our default value is “yes”. If the user resolved not to receive any emails from us, we will delete the email address from the user’s record, too.

Select the TUT_WANTS_SPAM field, double-click the **Default Value** attribute and change it to **Y**.

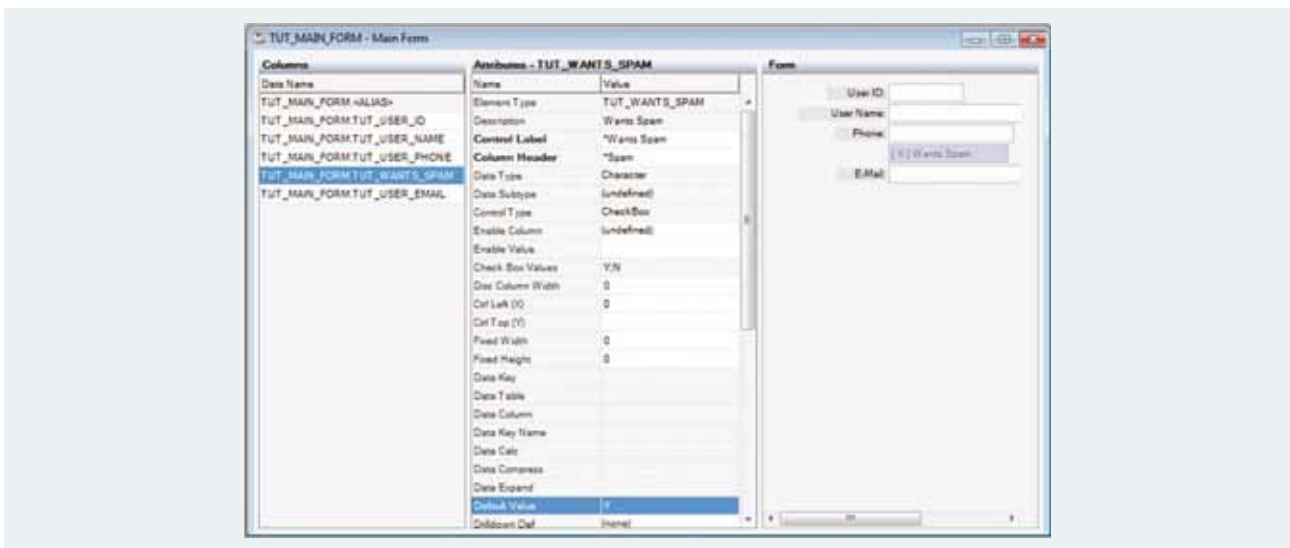


Figure 74 .1

Select the TUT_USER_EMAIL field, then:

- Double-click the **Enable Column** and select TUT_MAIN_FORM.TUT_WANTS_SPAM.
 - Double-click the **Enable Value** and enter **Y**.
- This is telling Barista that the **Email** column (field) should only be enabled when the **Wants Spam** column contains “Y”.

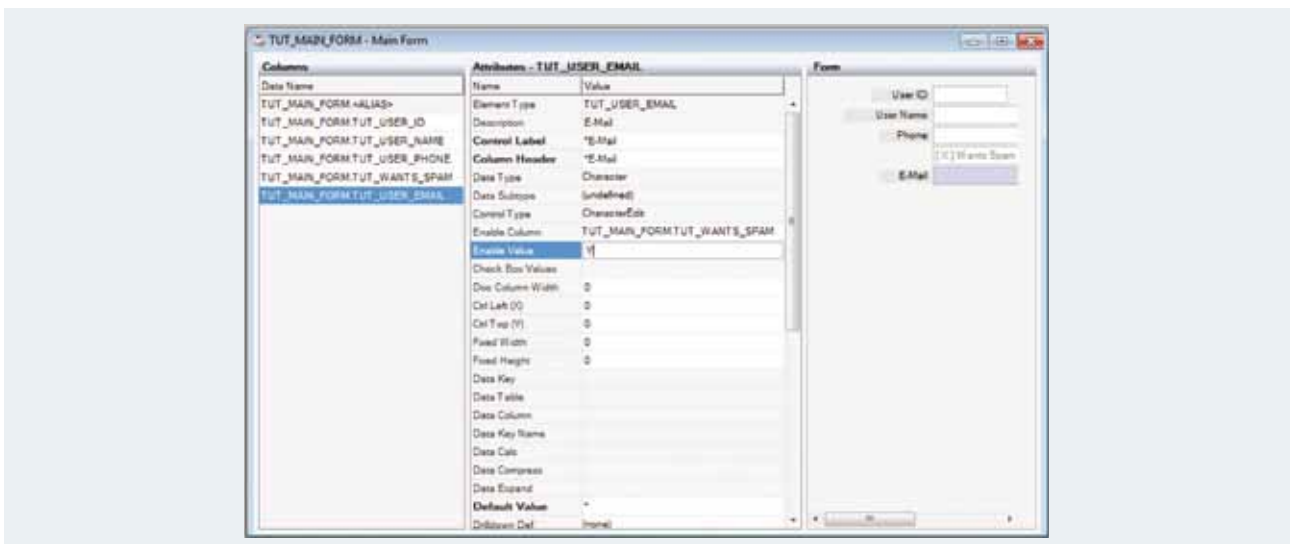


Figure 75

Click  or press F5 to test these changes.

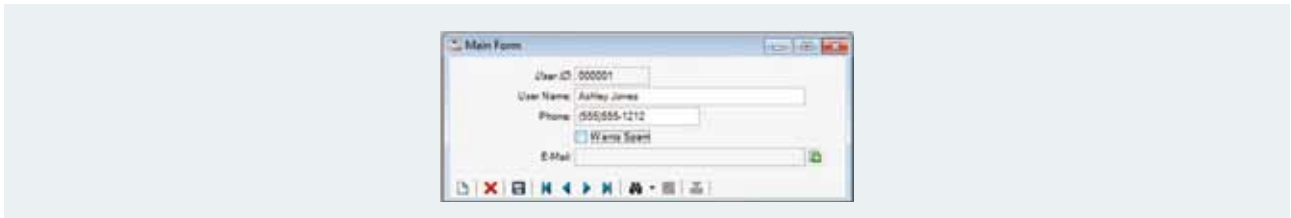


Figure 76

Unchecking the „Wants Spam“ checkbox now disables and clears the email field. You might want to keep the value, even if the checkbox is cleared. We'll do this via the **Optional Defs** attributes.

Keeping a Conditionally Disabled Field's Value

- Select the TUT_USER_EMAIL field.
- Double click next to Optional Defs.
- Check the box next to “Do not delete on conditional disable”.
- Click OK to save this change.

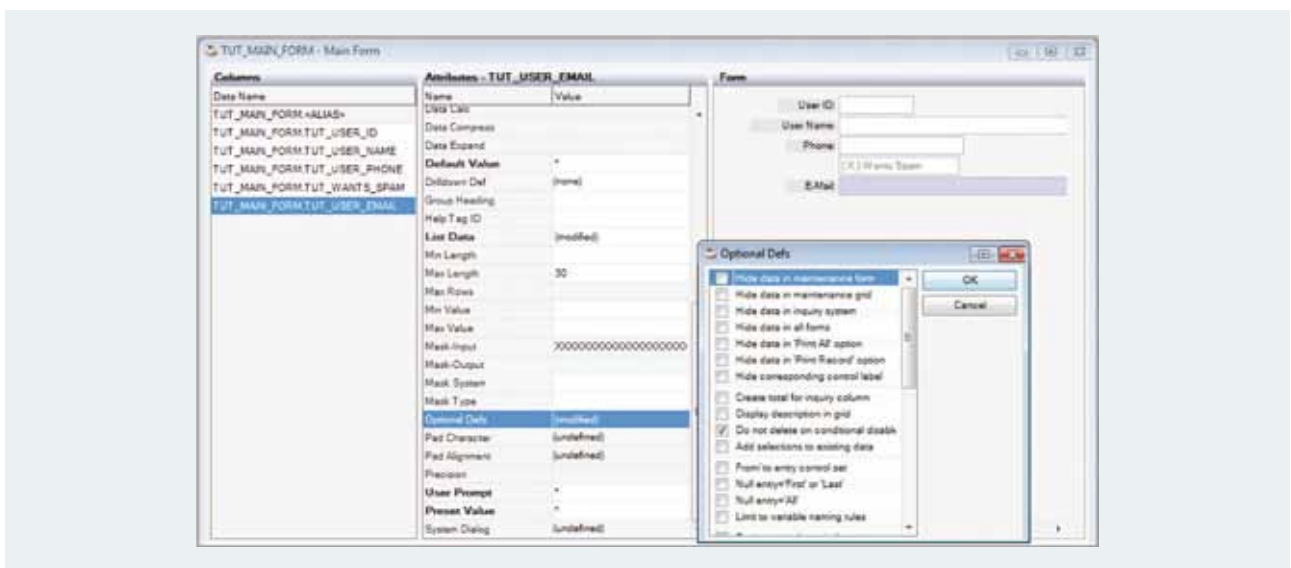


Figure 77

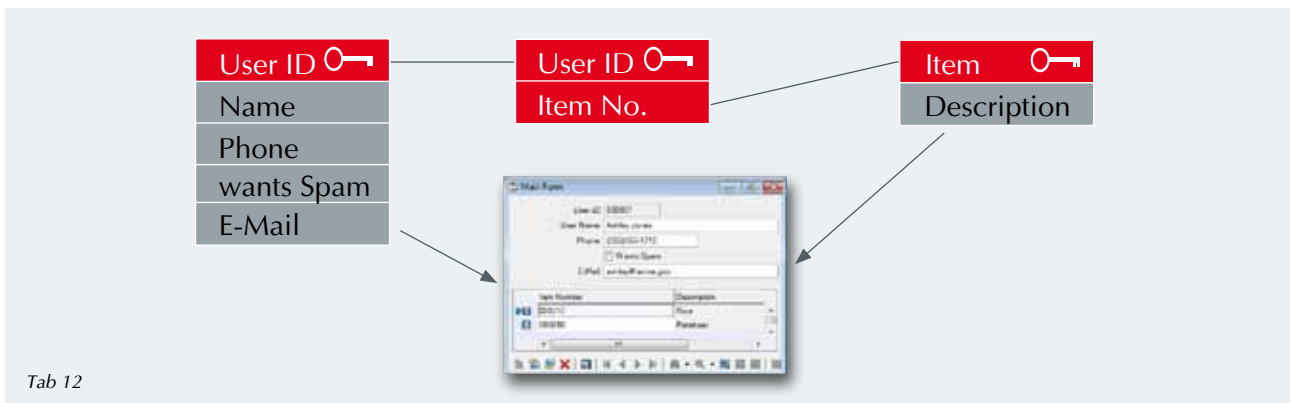
Run the form again to test the difference.

4 Options Entry Form

Build a Header/Detail Form

4.1 What We Are Going To Build

Let's get back to our initial project.



The forms we have created will be extended to allow us to keep track of different items that we have currently rented to users.

To achieve this, we will add two more small tables – a simple Item Master table to record our inventory and a User/Item detail table.

Furthermore, we will utilize foreign key constraints to exploit more application functionality. We'll link together the information from the two tables in a form that tells us what items a specific user has currently rented. We will derive this form from the Main Form we've designed before.

The resulting form will look like this:

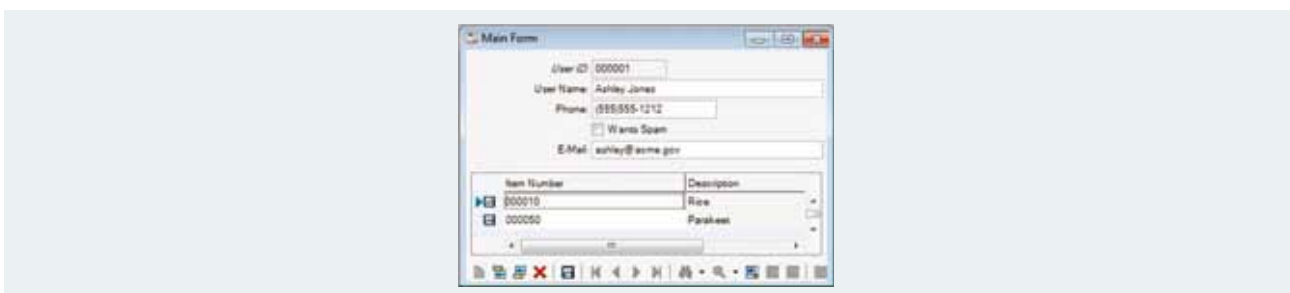


Figure 59

Again, we'll start by defining the necessary element types.

Define Element Types

Open the Element Types form and define the following element types.

Add Element Type: TUT_ITEM_NUM

Location	Field	Value
Header	Element Type ID	TUT_ITEM_NUM
	Description	Item Number
Definition Tab	Data Field Length	6
	Input Mask	000000
	Output Mask	000000
	Alignment	Right
	Pad Character	Zero
	App Product ID	TUT

Tab 13

Figure 60

Click  or press CTRL+S to save the record.

Add Element Type: TUT_ITEM_DESC

Location	Field	Value
Header	Element Type ID	TUT_ITEM_DESC
	Description	Item Description
Definition Tab	Data Field Length	50
	Column Header	Description
	App Product ID	TUT

Tab 14

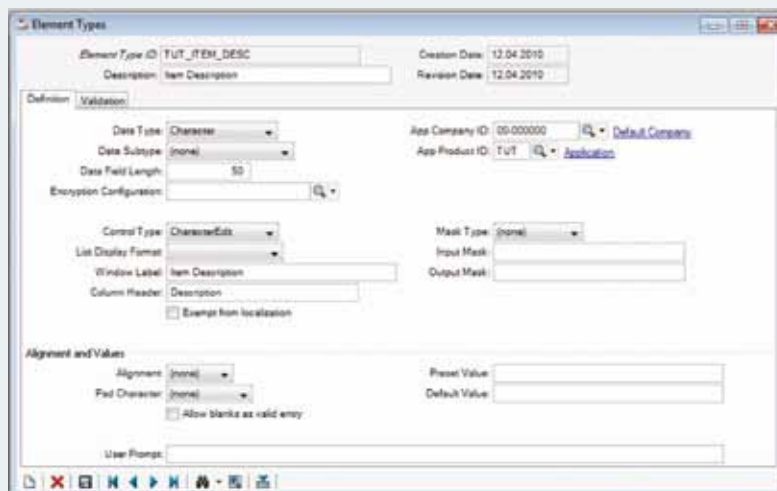



Figure 61

Click  or press CTRL+S to save the record.

Add Element Type: TUT_RENTAL_ID

This is the same as TUT_USER_ID from TUT_MAIN_FORM with the addition of table-based validation that will restrict input to values that already exist in TUT_MAIN_FORM.TUT_USER_ID. In other words, in our form TUT_RENTAL_ID serves as a foreign key in TUT_RENT_FORM. (Tip: There's a "Save As" item in the "Record" menu, but be sure to reset the Data Subtype to "(none)")

Location	Field	Value	Notes
Header	Data Element Name	TUT_RENTAL_ID	
	Description	Rental User ID	
Definition Tab	Data Field Length	6	
	Input Mask	000000	
	Output Mask	000000	
	Alignment	Right	
	Pad Character	Zero	
	App Product ID	TUT	
Validation Tab	Validate Data Table	TUT_MAIN_FORM	This enforces the rule that TUT_RENTAL_ID must exist as a key to the TUT_MAIN_FORM table.
	Data Column	TUT_USER_NAME	TUT_USER_NAME is the field of TUT_MAIN_FORM that will be displayed next to fields based on TUT_RENTAL_ID.

Tab 15

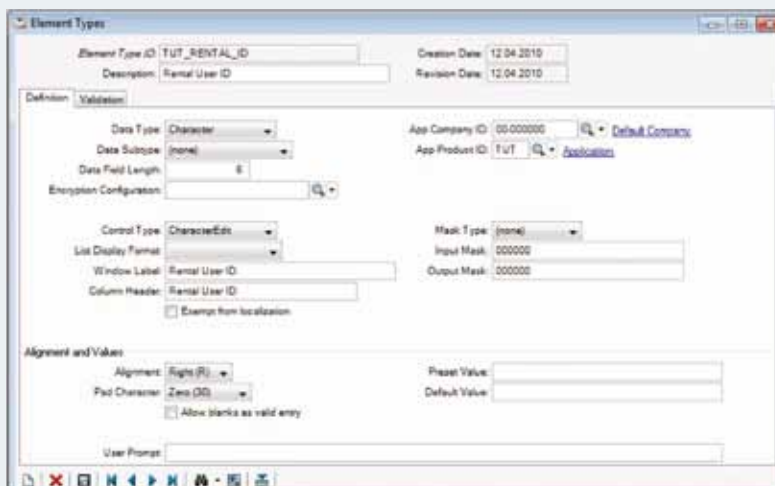


Figure 62

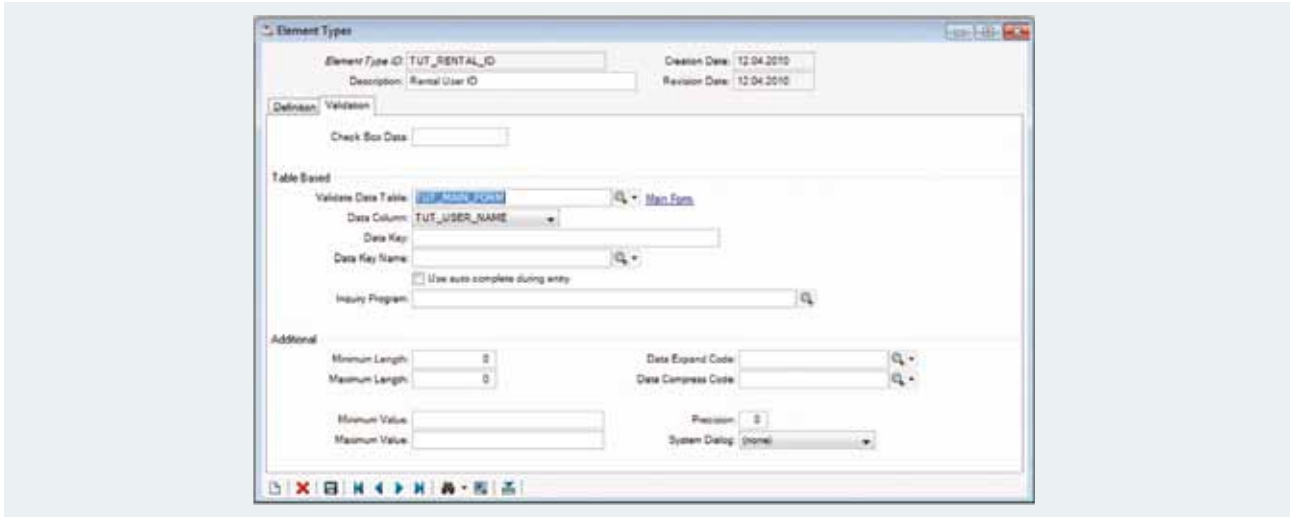




Figure 63

Click  or press CTRL+S to save the record.
 Click  or press CTRL+S to save the record.

Close the Element Types form. The next step is to define the tables that will use these element types.

Define Tables

Open the Tables form and define the following tables.

Add Item Inventory Table: TUT_ITEM_INV

Location	Field	Value
Header	Table Alias	TUT_ITEM_INV
	Description	Item Inventory
Security	App Product ID	TUT
Column Definitions	Element Type	Data Element
	TUT_ITEM_NUM	TUT_ITEM_NUM
	TUT_ITEM_DESC	TUT_ITEM_DESC

Tab 16

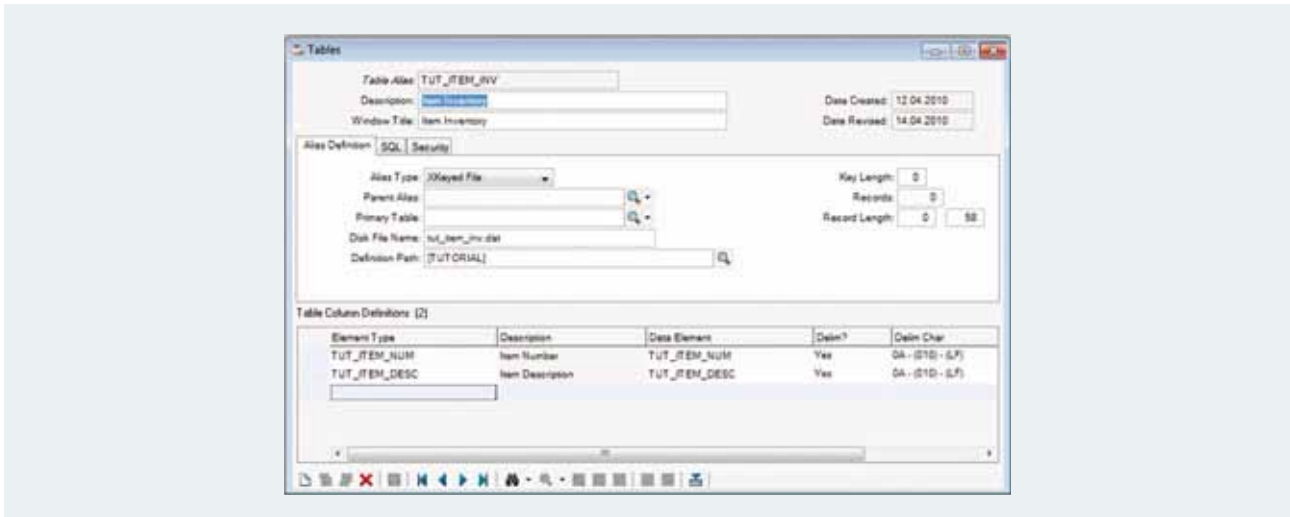




Figure 64

Press F7 to return to the header, then click  or press CTRL+S to save the table definition. Define the primary key to be TUT_ITEM_NUM using Key Definitions in the Options menu.

Click  or press CTRL+S to save the key definition, then close the Key Definitions dialog.

Create the file using Create/Update Table in the Options menu.

Add Rented Items Table: TUT_RENT_FORM

This is the table which connects TUT_MAIN_FORM and TUT_ITEM_INV.

Location	Field	Value	Notes
Header	Table Alias	TUT_RENT_FORM	
	Description	Rented Items	
Security	App Product ID	TUT	
Detail	Element Type	Data Element	Notes
	TUT_RENTAL_ID	TUT_USER_ID	The Data Element name TUT_USER_ID enables Barista to link it to the TUT_MAIN_FORM key.
	TUT_ITEM_NUM	TUT_ITEM_NUM	

Tab 17

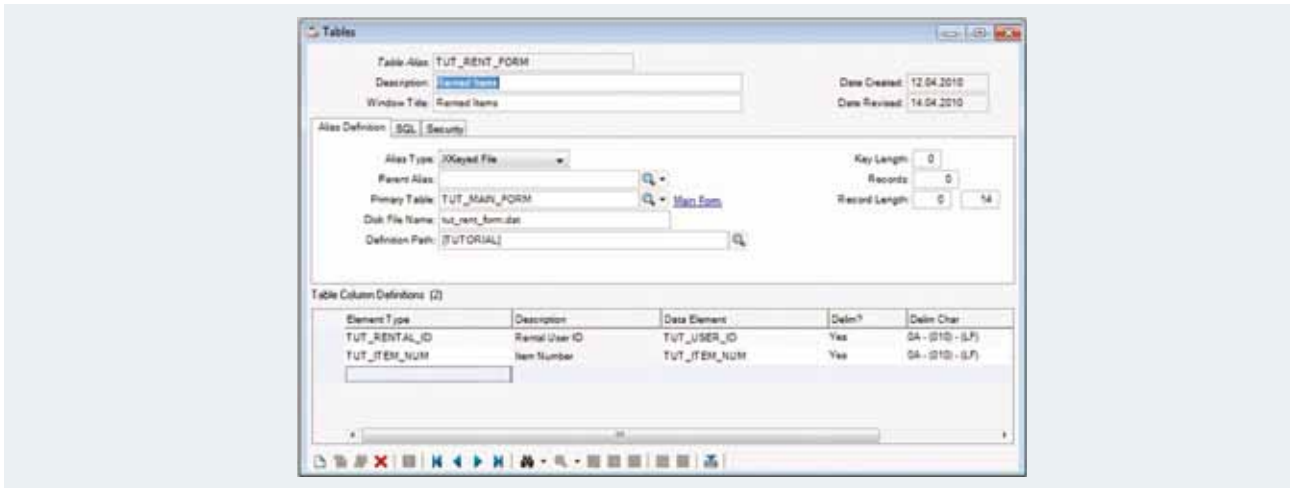


Figure 65

To establish the connection between the two tables, it is important to tell Barista over which actual Data Element you want your tables to link. You do this by using the same names in both tables in the “Data Element” column.

Press F7 to return to the header, then click  or press CTRL+S to save the table definition.

Set the primary key to TUT_USER_ID and TUT_ITEM_NUM using Key Definitions in the Options menu.

Create the file using Create/Update Table in the Options menu on the Tables form.

Close the Tables form.


Define Foreign Key Constraint for TUT_ITEM_NUM

Open the Element Types form and load the TUT_ITEM_NUM record. Add the following two fields:

Location	Field	Value
Validation Tab	Validate Data Table	TUT_ITEM_INV
	Data Column	TUT_ITEM_DESC

Tab 18

Figure 66

Click  or press CTRL+S to save these changes, then close the Element Types form. You can click **No** to the question that comes up, asking if you want to run the Table Rebuild facility. We'll be doing that as part of the next step.

Build the Forms for TUT_ITEM_INV and TUT_RENT_FORM

In this step, we tell Barista to actually generate the forms.

Open the Form Manager and double-click on TUT_ITEM_INV to open it in the Form Designer:

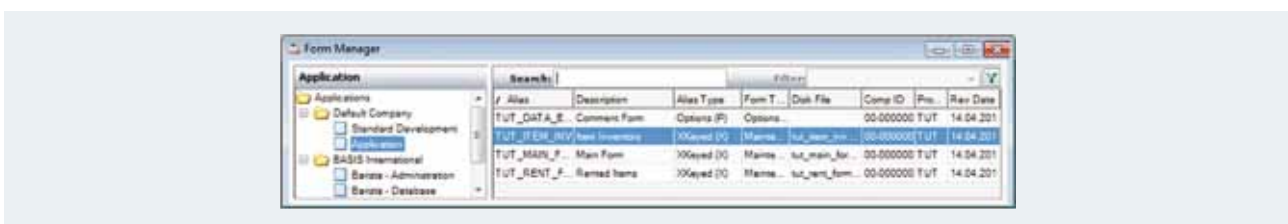


Figure 67

Change the Form Type from **Maintenance Form** to **Maintenance Grid**:

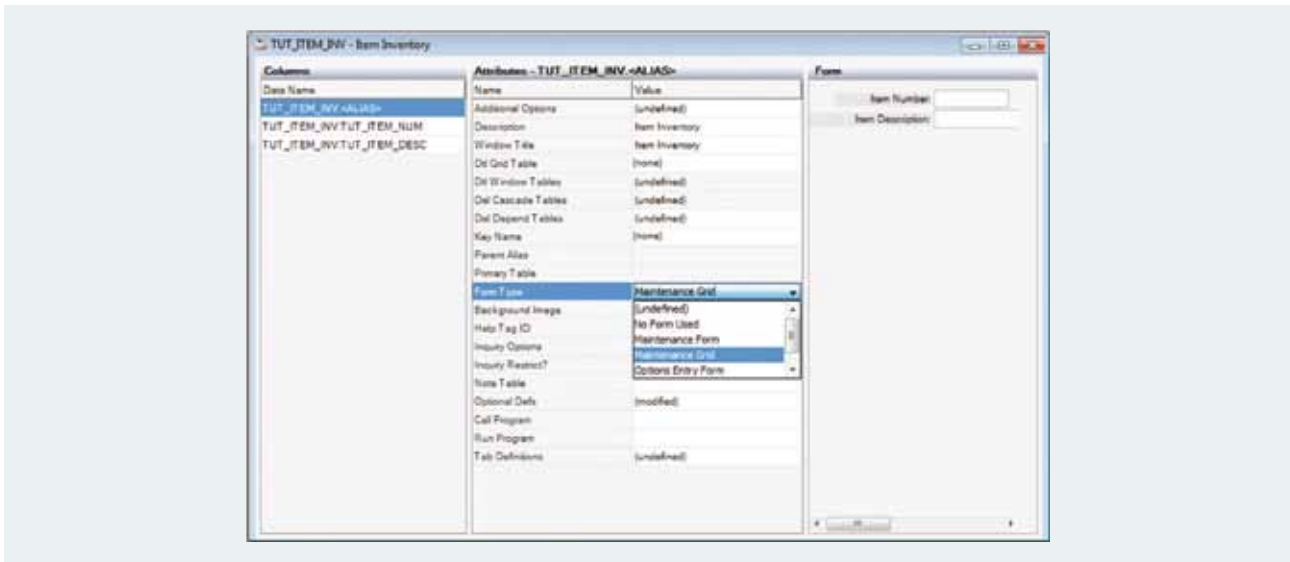


Figure 68

Close the Form Designer, saving that change:



Figure 69



Back in the Form Manager, select all of your forms and click  or press CTRL+B to build them:



Figure 70

You can now run any of those forms by selecting it and then clicking  or pressing F5. First, run TUT_ITEM_INV and create some inventory items:

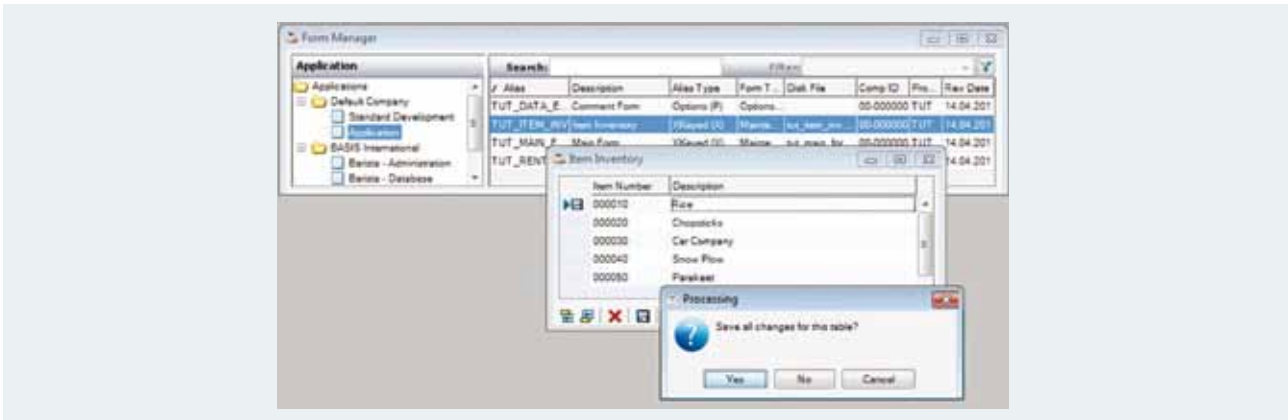


Figure 71

Next we will tie together all the tables to show at a glance what each user has rented. This will be done using a **detail grid**. The data in the detail grid is indexed by the primary key of the main (or header) form. So in this case, we will display the TUT_RENT_FORM as our detail grid.

The key for the TUT_RENT_FORM is the TUT_USER_ID of the TUT_MAIN_FORM. The TUT_ITEM_NUM of TUT_RENT_FORM uses a table validation and can be set to display the description from TUT_ITEM_INV as a read-only column in the grid.

Adding a Detail Grid

Before we can add a detail grid, the primary table needs to be assigned to the detail (sub-) table. This sets up the relationship between the two tables.

Open the Tables form and load the sub-table TUT_RENT_FORM. Enter TUT_MAIN_FORM as its **Primary Table** on the **Alias Definition** tab:

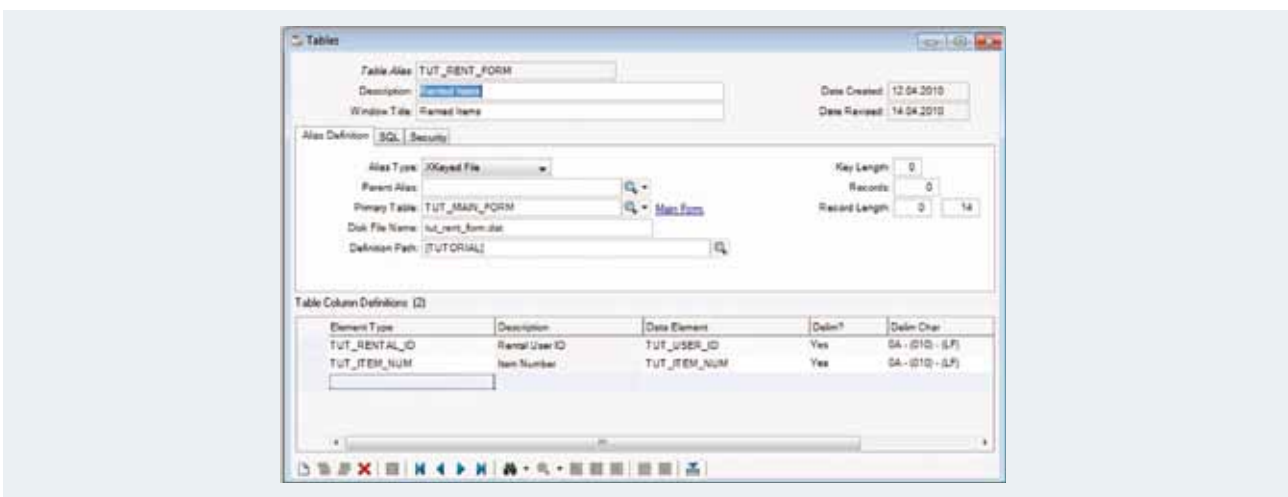


Figure 78

Save and exit the Tables form.

Open TUT_MAIN_FORM from the Form Manager.

Select TUT_MAIN_FORM.<ALIAS> under **Columns** and select TUT_RENT_FORM for **Dtl Grid Table**:

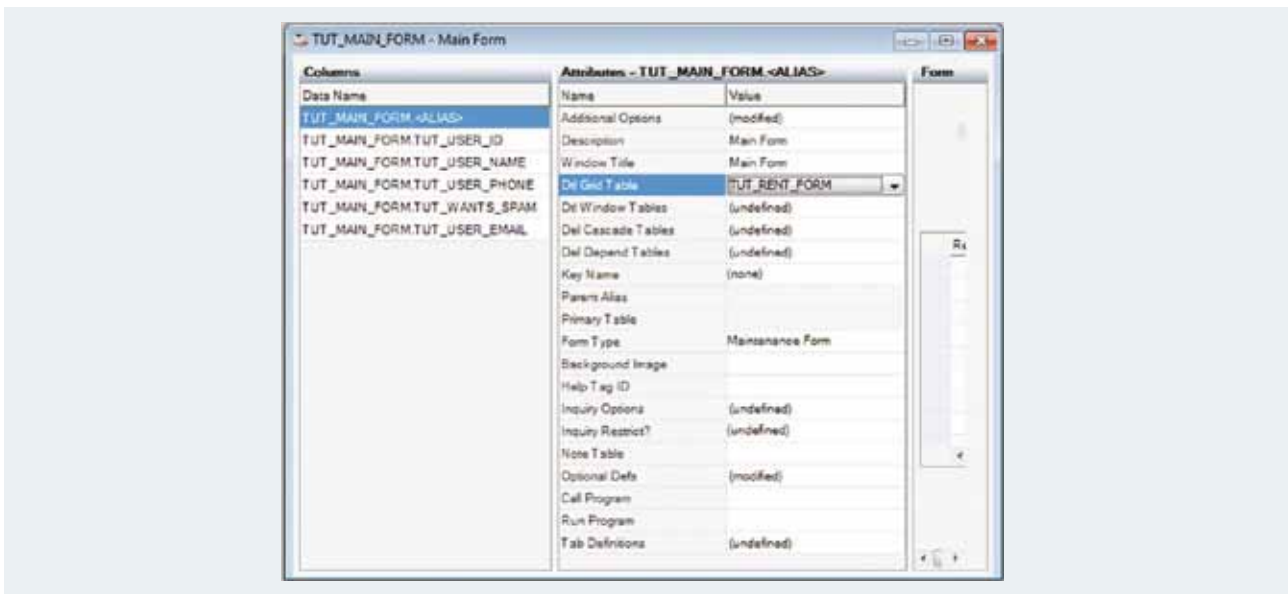



Figure 79

While we're here, let's also set to **cascade delete** the detail table records when the main record is deleted. Double-click in the value column next to the attribute named **Del Cascade Tables**:



Figure 80

Check the TUT_RENT_FORM table and click OK.

Click  or press ALT+F5 to update the displayed form.

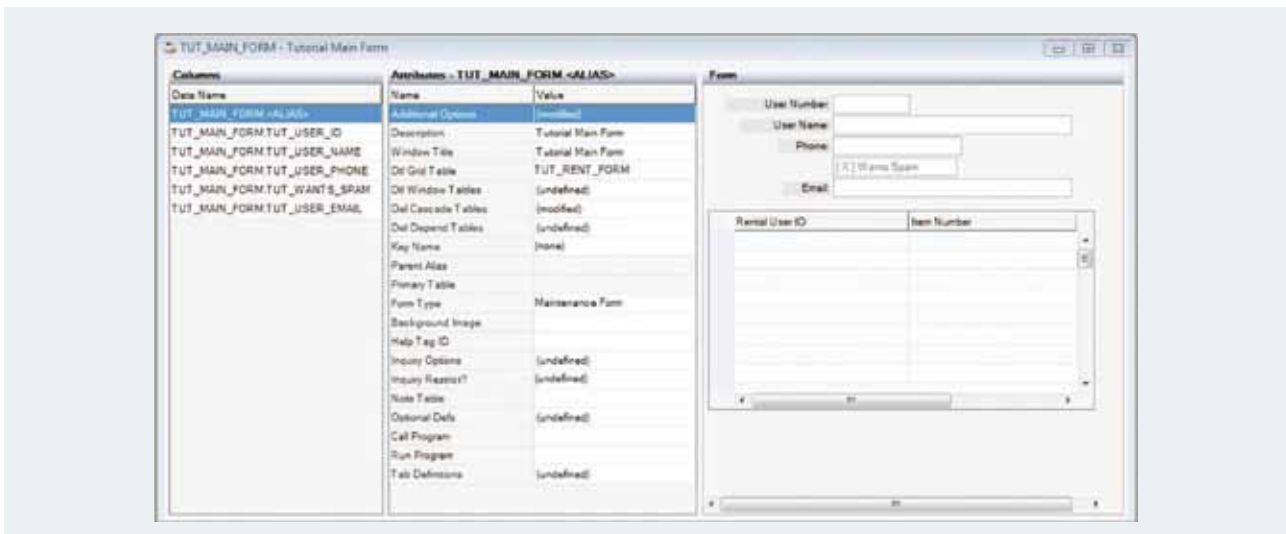


Figure 81

It's getting there, but we don't want to show the User ID for every line of the grid, and it would be nice to see the item description along with the item number.

Fixing up the Detail Grid

Close TUT_MAIN_FORM, saving your changes:



Figure 82

Double-click on TUT_RENT_FORM to open it in the Form Designer:



Figure 83

Select TUT_USER_ID. Double-click next to the attribute „Optional Defs“ or right-click on the field and select „Optional Definitions“ from the context menu. Check „Hide data in maintenance grid“ and then click OK.

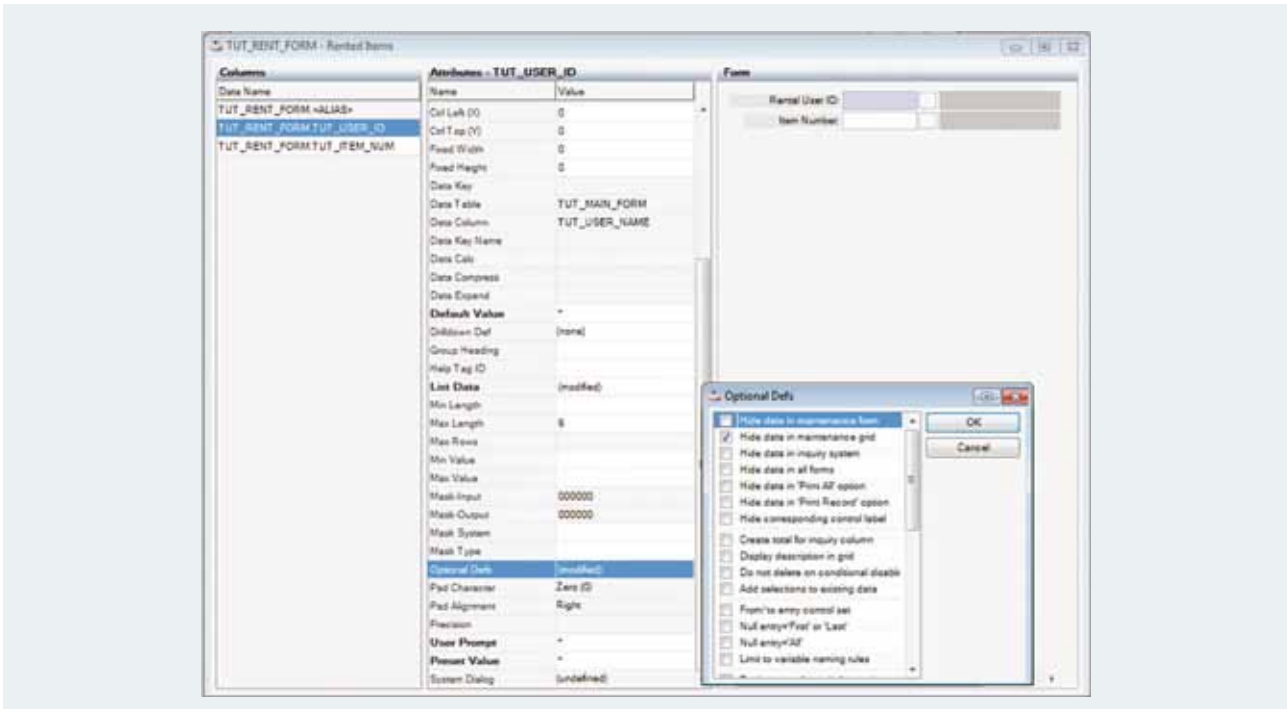


Figure 84

In a similar fashion, set the optional definition „Display description in grid“ for column TUT_ITEM_NUM to have it’s related data column (description) displayed in the detail grid.

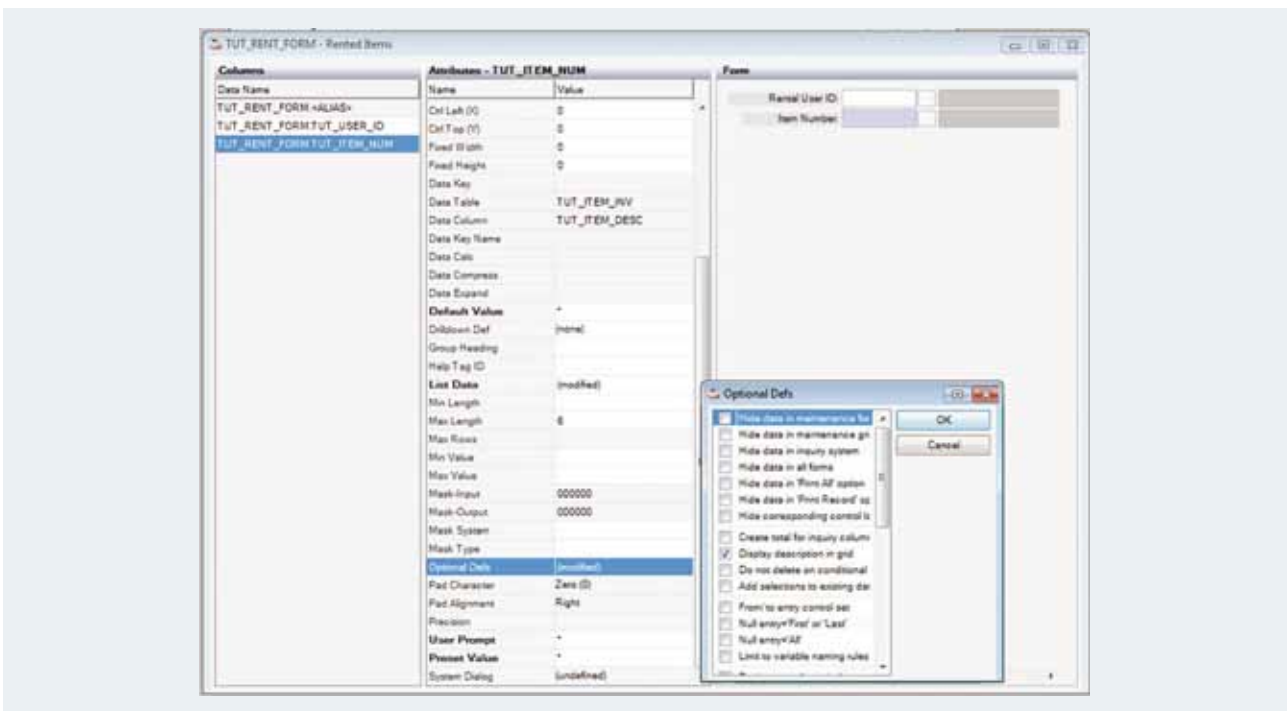



Figure 85

Close the Form Designer, saving those changes.



Figure 86

Back in the Form Manager, select all of your forms and click  or press CTRL+B to build them.

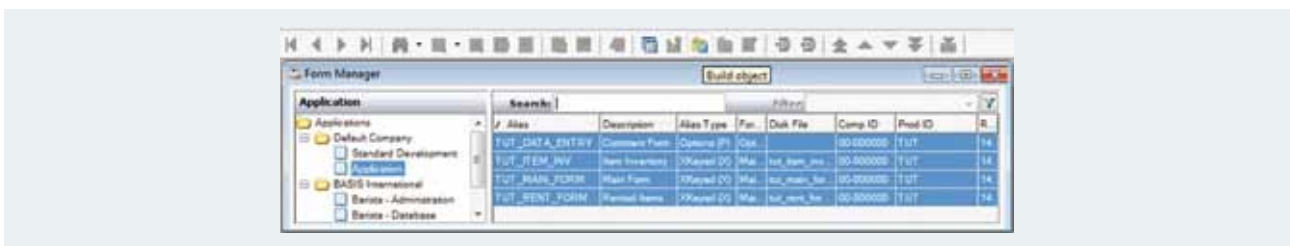



Figure 87

Now select the **TUT_MAIN_FORM** in the Form Manager, and run it by clicking , pressing F5, or right-clicking on it and selecting „Run Process“:

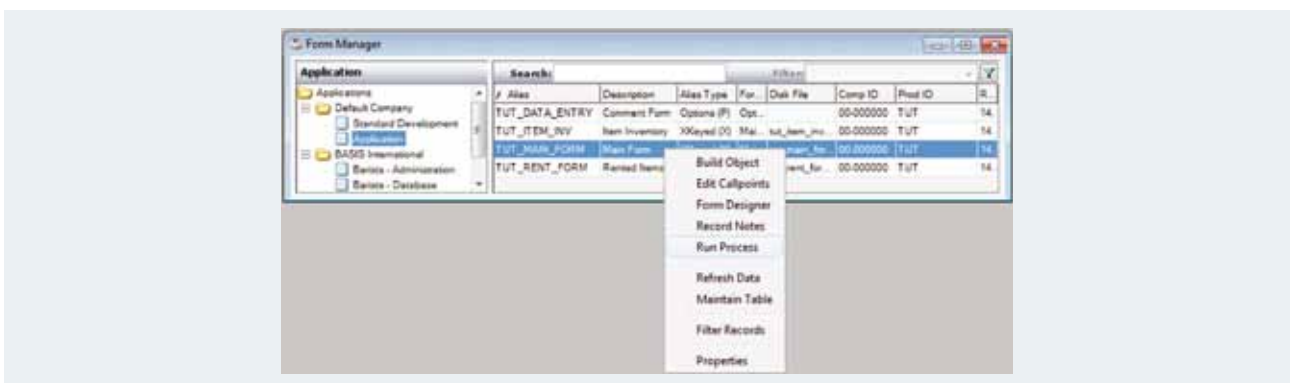


Figure 88

Barista remembers your last form size, so first resize the form to make the detail grid visible.

Now we can add rented item records:

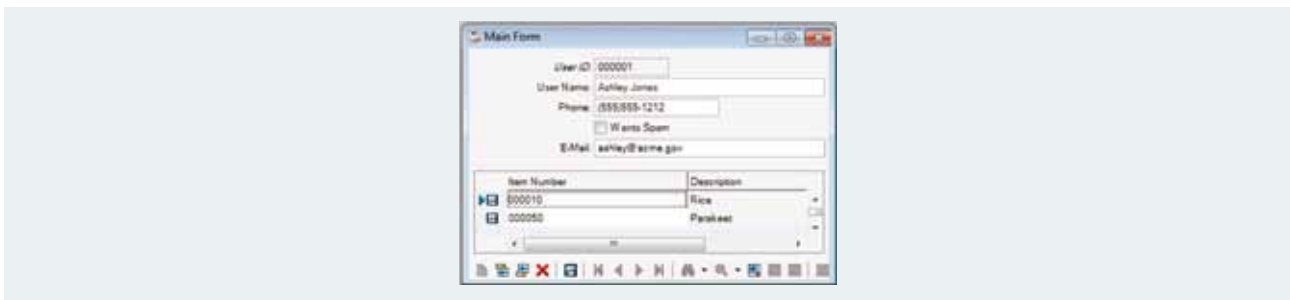


Figure 89

You can lookup item numbers while in the **Item Number** column by clicking  or pressing CTRL+F:



Figure 90

Add a Menu for the Tutorial Form

Our application is almost completed. By creating the tutorial application in the very first step, Barista created an empty menu definition, which we will now use to create a menu item to run the main form.

Choose “Menus” from the **Barista Administration** menu. In the dialog, select the tutorial menu and click the “Maintain” button.

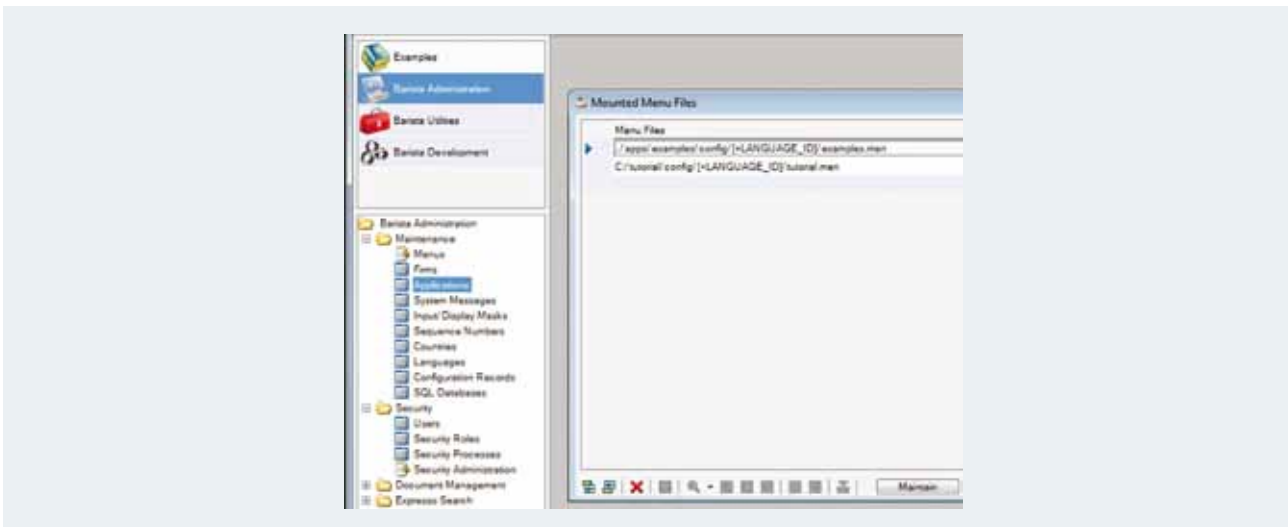


Figure 91

Right-click on the root node in the tree on the left and select “Add Application Menu”.

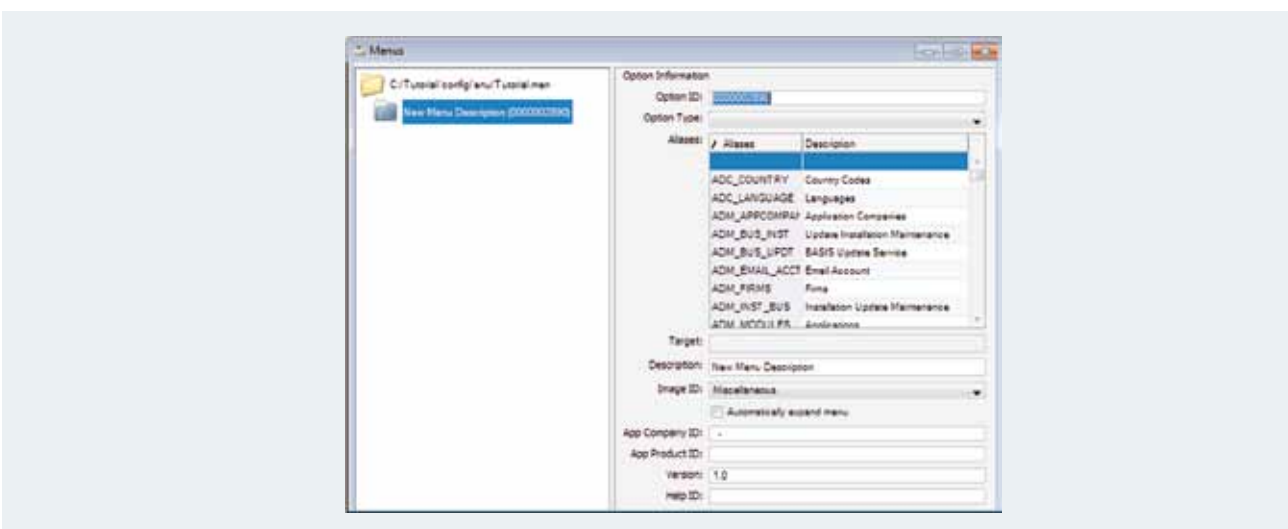


Figure 92

Now name your Menu “Tutorial” (under “Description”) and select an icon from the Image-ID pull-down. Enter the Application Company ID “00-000000” and the Application Product ID “TUT”. A right-click on our new “Tutorial” menu lets us select “Add Menu Item”.

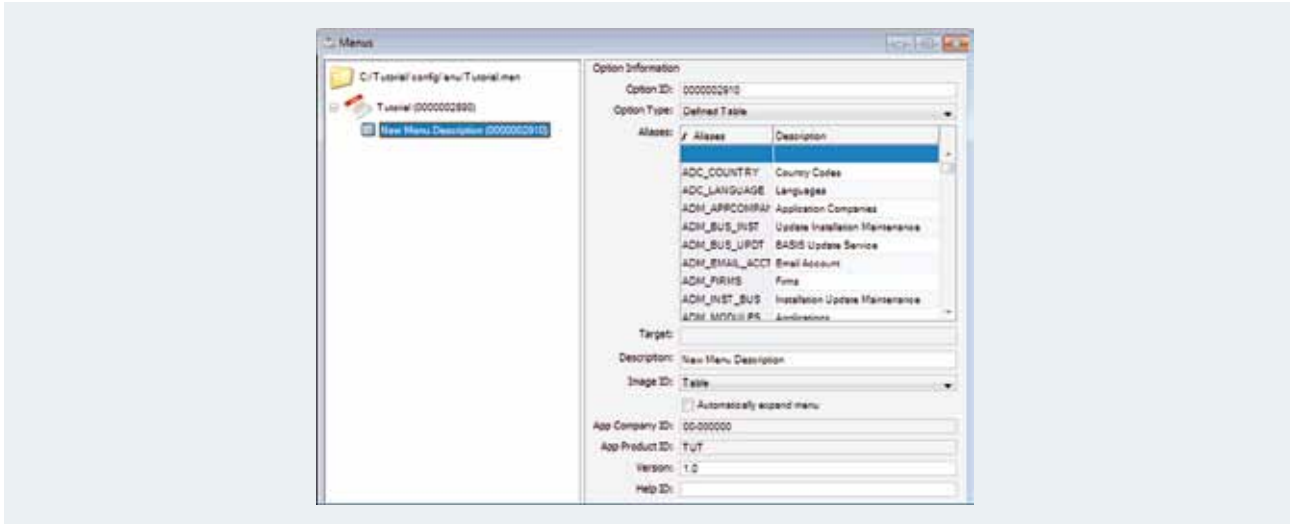


Figure 92

Select “Defined Table” as Option Type and chose the “TUT_MAIN_FORM” in the Aliases list.

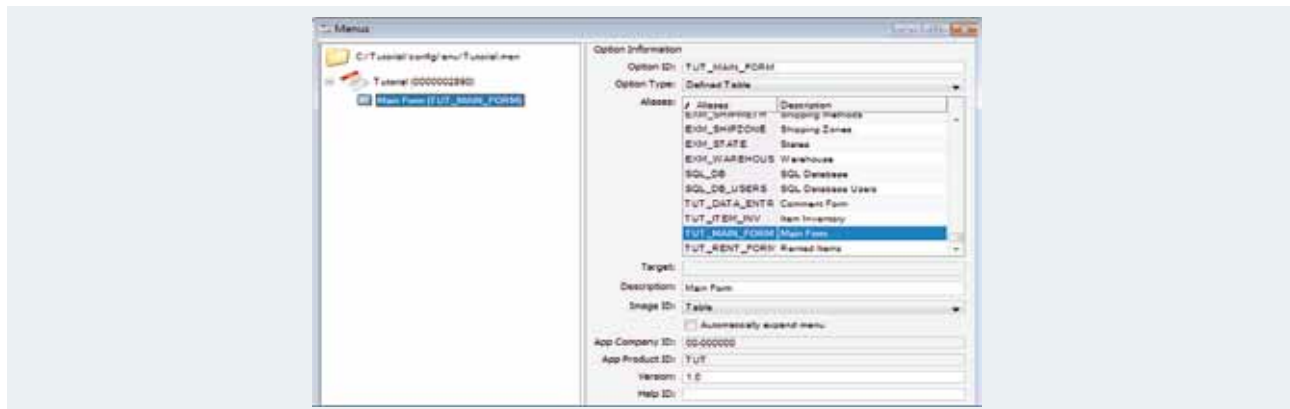



Figure 93

Click on the Save-Button  and close the form to return to the list of mounted menus. **Resize** the form if you don't see the **Refresh** button to the right of the **Maintain** button. Click on to redraw the menus and show our new item.

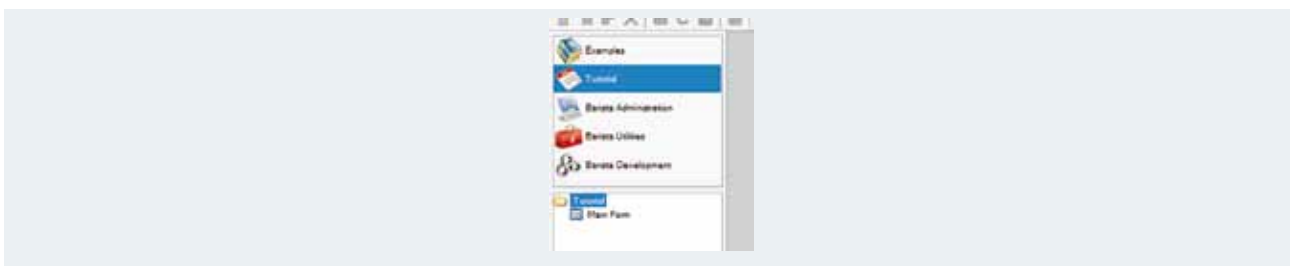


Figure 94

Run the form and have fun!

4. Appendix

4.1 Further Reading

Barista

This introductory tutorial has only scratched the surface of Barista capabilities. More Barista documentation, tutorials and presentations can be found at: <http://www.basis.com/products/devtools/barista/documentation>

There, we recommend that you look at the “**Getting Started**” document next. It offers a comprehensive overview of the vast opportunities of Barista for generating GUI applications from simply defining element types, tables and their relations.

BBj Programming Language and Development Environment

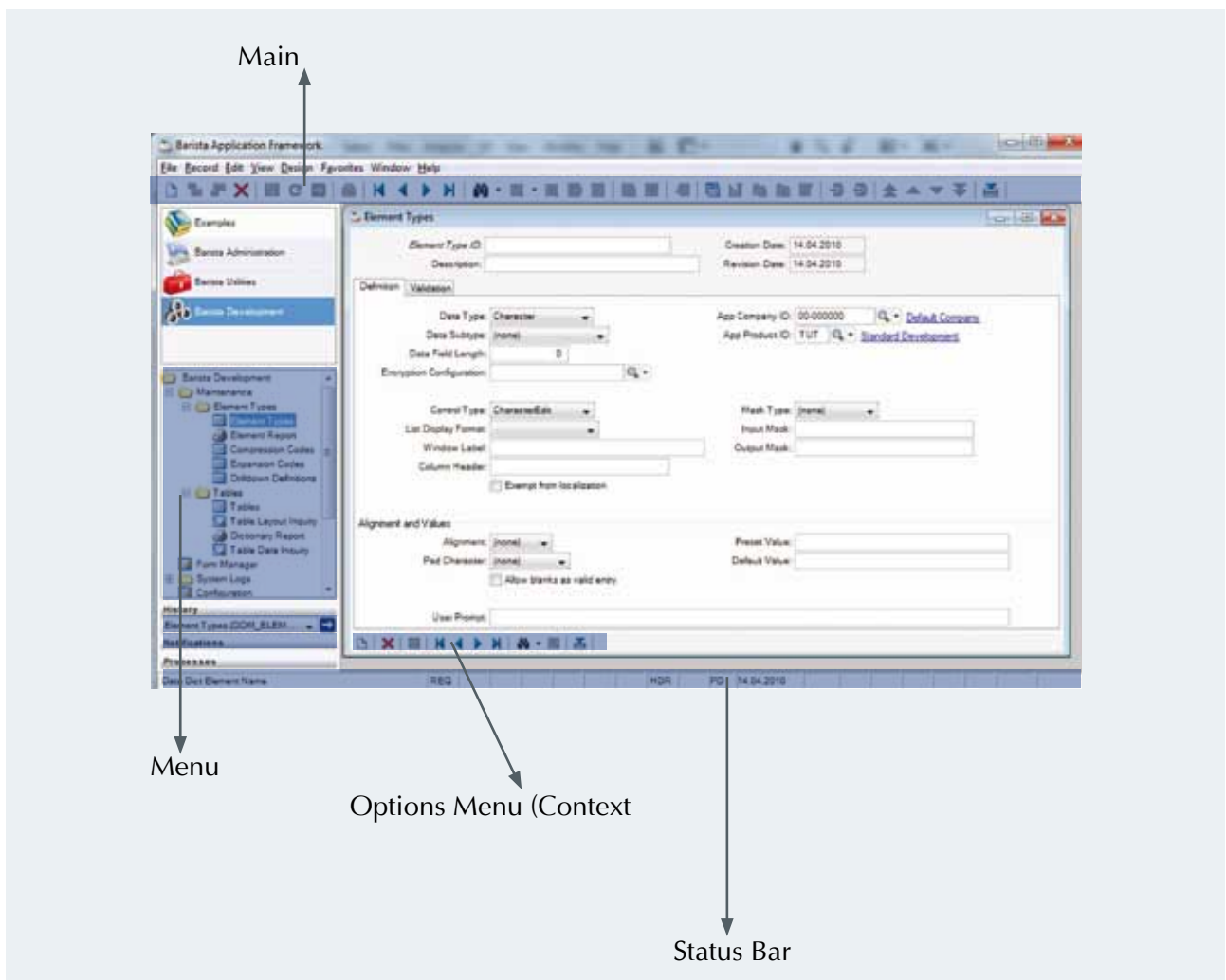
Barista is a Rapid Application Development (RAD) framework that belongs to the comprehensive BBj family of programming tools, optimized for designing scalable, platform independent, professional business applications. Here, you find an entry point for an abundance of information on our products:

<http://www.basis.com/products/index.html>

BASIS and BASIS Europe

For more information on BASIS International Ltd. and BASIS Europe Distribution GmbH, see our websites at www.basis.com and www.basis-europe.eu, respectively.

4.2 Definition of Screen Elements



4.3 Contacts

Europe:

BASIS Europe Distribution GmbH

Nell-Breuning-Allee 6
66115 Saarbrücken
Tel. +49 (681) 9 68 14-0
info@basis-europe.eu
www.basis-europe.eu

Company Headquarters:

BASIS International Ltd.

5901 Jefferson Street NE
Albuquerque NM 87109-3432
USA
Tel. +1.505.345.5232
Fax +1.505.345.5082
info@basis.com

For **Support**, **Sales**, and **Consulting** contacts in your country, please refer to
<http://www.basis.com/company/contactus.html>



BASIS Europe Distribution GmbH

Nell-Breuning-Allee 6
66115 Saarbrücken
Deutschland
Tel. +49 (0) 681 9 68 14-0
Fax +49 (0) 681 9 68 14-33
info@basis-europe.eu

www.basis-europe.eu