Guide to GUI Programming in BBj

The purpose of this guide is to introduce programmers to GUI programming in BBj®, the current generation of BBx®.

Guide to GUI Programming in BBj

Introduction

Back to top

Since its introduction in 1985, BBx® has evolved through the addition of new features while maintaining support for legacy features. Because of this evolutionary approach, much of BBj® is familiar to BBx programmers who have prior experience with PRO/5® for UNIX or Visual PRO/5® for Microsoft Windows. Programmers with Java experience and other GUI development languages will appreciate the newest object-oriented syntax and concepts.

Historical Background

Back to top

BBj is the most recent evolution of the Business BASIC programming language, originally developed for minicomputer systems in the early 1970s. Business BASIC has traditionally provided powerful and easy-to-use I/O verbs and functions, as well as sophisticated file structures to support complex application data designs.

In the 1980s, Business BASIC developers ported their software to PC-based systems. Starting in 1985, the BBx (Business BASIC Extended) programming language became one of the premier implementations in the world. It offered full portability of source code and data across many PC and minicomputer platforms.

With the release of Visual PRO/5 for the Microsoft Windows platform in the 1990s, BBx added a complete set of graphical commands and tools for developing GUI applications. At the same time, it continued to provide compatibility with legacy applications. Programs written in all previous versions of BBx continued to run in Visual PRO/5 with no modification or conversion of programs or data.

Towards the end of the 1990s, GUI programming gained widespread usage on platforms other than Microsoft Windows. Web-based applications were becoming commonplace, and GUI environments on Linux and Apple's Macintosh platform were becoming increasingly significant. To address these new requirements, BASIS ported the BBx language to Java and released this new evolution of the BBx language as BBj. With a few very limited exceptions, programs written in previous versions of the BBx language continue to run in BBj. More significantly, BBj programs – including GUI programs – run unchanged on all supported platforms, including Microsoft Windows, Linux, and Macintosh OS X.

The enhanced and re-engineered BBj Structured Query Language (SQL) engine continues to provide access to the BASIS DBMS and third party SQL compliant databases. BBj's direct access to native BASIS DBMS file formats remain unchanged from earlier versions of BBx, while BBj supports new file formats such as the Journaled File type in the BASIS DBMS. No

data modification or conversion is required for BBj to access legacy filesystems. To learn more about mixed mode deployments of BBj and the PRO/5 family and their associated databases, refer to the Technical Resource Zone feature "Lock, Lock, Who's Got The Lock," in the online BASIS International Advantage magazine.

Graphical User Interfaces

Back to top

The graphical user interface differs from the traditional character-based interface in two significant ways: It accepts input from a pointing device, usually a mouse, and it allows the user to generate a sequence of events that the programmer cannot predict.

The graphical environment is sufficiently complex to require a runtime driver quite different from the character-based screen drivers used by traditional applications. In order to use the graphical environment, an entirely new set of commands is necessary to accommodate the needs of the programmer.

BBj supports the character-oriented TermConsole device, carried forward from UNIX versions of PRO/5, as well as the character-oriented SysWindow device and the GUI-oriented SYSGUI device, both carried forward from Visual PRO/5 for Windows.

Terminology

Back to top

Term	Definition
Context	In BBj, a developer creates each graphical window with its own context, which is simply an integer; and references all objects and events associated with a window using its context.
Control	A control is an object that the developer uses as part of a graphical interface. Some typical controls include push buttons, text boxes, list buttons, and radio buttons.
Dialog	A dialog is a special window that requires a response from the user. In most languages, the developer creates dialogs and graphical windows through a discrete syntax. BBj offers a simple, uniform syntax for creating dialogs and windows. In BBj, one simply creates dialogs by assigning dialog attributes to windows.
Event	An event is the report of a graphical object or device that the user operates or manipulates such as by clicking a mouse button, choosing a menu item, or pressing a key on the keyboard.
GUI	GUI is an acronym for Graphical User Interface.
Mnemonic	A mnemonic is an abbreviated reference intended to trigger memory to some command or concept. This term was adapted into traditional Business BASIC languages for some classes of I/O commands. BBj extends the mnemonic classes to include the creation and management of graphical objects.
Object	An object consists of data and specific rules that control operations performed on that data.

SYSGUI	The SYSGUI device, defined in the configuration file, provides a method of communicating with the graphic-based screen interface provided with BBj.
SysWindow	The SysWindow is a device defined in the configuration that represents the character-based screen interface provided by BBj. It represents an extension to the traditional BBx terminal device.
Window	A window is a logical screen used for many purposes. Graphical windows created via the SYSGUI device differ from the character-based windows created with the SysWindow device. Both devices, however, use windows.

Interactive Devices

Back to top

BBj builds on features introduced in earlier versions of the BBx language, including PRO/5 for UNIX and Visual PRO/5 for Windows. It implements several different input devices.

- TermConsole
- SysConsole/SysWindow
- WinConsole
- SYSGUI

The TermConsole Device

Back to top

The TermConsole is a character-based user interface available for dumb terminals or terminal emulators. As in PRO/5, it relies on termcap. A "console" interface gives users the ability to type commands at a ready prompt, edit and list BBj program source code, and display the output of character-based programs.

The SysConsole/SysWindow Device

Back to top

The SysWindow is a character-oriented display device with some graphical capabilities. In addition to the traditional character interface implemented by the TermConsole, the SysWindow adds optional mouse-sensitive areas, a MSGBOX() dialog, and standard "file open" and "file save" dialogs. Traditional character-based applications can run in this environment without modification, thus facilitating the gradual replacement of CUI programs with programs written to take advantage of the graphical environment.

The WinConsole Device

Back to top

The WinConsole is a debugging environment for developers. It enables the developer to set breakpoints, dynamically watch variables, and view multiple levels of running programs in a tabbed display.

The SYSGUI Device

Back to top

The SYSGUI device provides an environment that accepts a full range of graphical commands for the creation of full-featured graphical applications. The SYSGUI device supports several kinds of windows and GUI controls, along with extensive drawing and plotting commands.

The SYSGUI device requires a special alias line in the config.bbx file. The installation process automatically creates this alias line:

```
ALIAS X0 SYSGUI
```

To use the SYSGUI device, one just opens a channel to X0 and starts interacting with it. For example, the following code fragment will open X0, retrieve a resource from a resource file, and display it:

```
0010 OPEN (1)"X0"
0020 LET H=RESOPEN("cust.arc")
0030 LET R$=RESGET(H,1,101)
0040 PRINT (1)'RESOURCE'(LEN(R$)),R$
0050 ESCAPE
```

Commands sent to the SYSGUI device driver are essentially print mnemonics. Traditional Business BASIC programmers will find this command format familiar and similar in concept to the mnemonics they use in character-based environments such as 'CS' (clear screen) and 'SB' (start background or dim mode). Even programmers new to BBj and its predecessor languages will find it easy to use mnemonics.

Most SYSGUI mnemonics are procedure calls with lists of parameters. BBj employs concepts similar to other graphical languages. It inherits from traditional Business BASICs the compact and powerful I/O verbs, which have made rapid application development a reality. Since Business BASIC programmers are accustomed to using I/O verbs and functions that do not require extensive supporting code, BASIS has extended this ease of use and simplicity wherever possible to the SYSGUI device.

In particular, the distinctions between windows and dialogs, as known in other languages, blur to the point where developers can design BBj windows to exploit some of the best of both windows and dialogs simultaneously. This simplified approach reduces the set of commands required to support full-featured applications.

Because applications frequently need to work with several windows at a time, BBj uses the concept of contexts to differentiate between windows. Each window has its own context; each event returned from the SYSGUI device has a context ID. Setting the context is not necessary if only one graphical window is needed at a time. However, if you want to display one window, then display another window, and manipulate them simultaneously, the programs must be context aware.

Valid context IDs are integers from 0 to 32767. When the SYSGUI device is first opened, the context is automatically assigned an ID of 0. If it is not changed, the first window created is associated with context 0. To create another window, specify a new context by using the ‘CONTEXT’ mnemonic with an unused ID. The next window created is associated with the most recent 'CONTEXT' value.

Although there can be many contexts, only one context is considered to be the current context. Many commands operate on the current context while others specify the context ID as a parameter. To switch contexts, issue the 'CONTEXT' mnemonic with the desired context ID.

It is possible to query the SYSGUI device for an unused context ID to avoid runtime errors or logic problems. Developers should write programs as "add-on" modules so other applications can make use of this feature. To query the SYSGUI device, use the FIN() and CTRL() functions.

Using the SYSGUI Device

Back to top

The purpose of the SYSGUI device is to facilitate the creation and manipulation of custom graphical windows from a BBj program. To use the SYSGUI device, add an alias line like the following to config.bbx:

```
ALIAS X0 SYSGUI
```

Any alias starting with X can be used, but there is no advantage to using anything other than X0, which BBj provides by default upon installation. Note that the installation process should add this line to the config.bbx file.

Open X0 and start interacting with it. Create and destroy multiple windows, controls, etc., without closing or reopening X0.

Querying the SYSGUI Device

Back to top

At any time, the CTRL() function can be used to get information about the current state of the SYSGUI device, including current context, contents of controls, and the like.

FIN(chan) also returns useful information including current context, number of active contexts, and first available context. Get the TMPL(chan,IND=0) to see the template. For more information, see FIN() function.

Event Driven Programming

Back to top

A graphical interface typically offers a variety of controls that the user may operate at random using the keyboard and/or the pointing device. Any of these user choices can potentially

generate events, some of which require the application to deliver an appropriate response. Since the user can initiate an event from any active control on the screen, the software is said to be event driven.

Character-based applications have used event-driven methods for some time. Applications that incorporate pre-assigned function keys or menu-driven interfaces are good examples.

Visual PRO/5 Event Loop Model

Back to top

Visual PRO/5 and BBj implement GUI programming using an event loop. The SYSGUI device reports all events through a single event queue. BBj returns each event as a fixed-length string in a standard format. To minimize the risk of generating preventable runtime errors, BASIS recommends using string templates to retrieve event data. This ensures that the application program will always have a correct reference to the control that passed the most recently received event. A template for the event structure is available with the TMPL(channel) function for retrieving template data on open channels. For example:

```
SYSGUI=unt
OPEN (SYSGUI)"X0"
PRINT (SYSGUI)'WINDOW'(100,100,100,100,"Window",$00010003$)
PRINT (SYSGUI)'BUTTON'(1,10,30,80,30,"OK",$$)
DIM EVENT$:TMPL(SYSGUI)
REPEAT
READ RECORD (SYSGUI,SIZ=LEN(EVENT$))EVENT$
IF EVENT.CODE$="B" AND EVENT.ID=1 THEN GOSUB OK
UNTIL EVENT.CODE$="X"
STOP
OK:
I=MSGBOX("User clicked the OK button")
RETURN
```

The format for an event returned by TMPL(sysgui) is:

Template Field	Meaning
context	Event Context ID
code	Code for the type of event
id	ID of the control sending the event
flags	Flags – interpretation varies with type of control
x	X location or dimension
y	Y location or dimension

Each newly created BBj window is in a context that serves as the frame of reference for objects and events associated with that window. When taken with the context field, the id field uniquely identifies the control that is associated with the event.

A program recognizes events by the values of the code field in the event messages. However, this is often insufficient information to determine the application's response. Additional information about an event is passed in the flags and x and y parameters.
The following table contains a list of the events that can be passed and the corresponding event mask flag bits:

Event Description	Code	Eventmask
Push button operated	B	00000000
Tool button operated	b	00000000
Menu selection made	C	00000000
Notify (extended events on various controls)	N	00000000
Popup menu item selected	P	00000000
Close box operated	X	00000000
Popup requested event	r	00000001
Window focus change	F	00000004
Window resized	S	00000008
Mouse wheel scroll	w	00000020
Mouse button down	d	00000040
Mouse button up	u	00000080
Mouse moved	m	00000100
Mouse button double click	2	00000200
Keypress	t	00000400
Scroll bar or slider position changed	p	00100000
Edit control modified	e	00400000
Control focus gained or lost	f	00800000
Click or double click on list item	l	01000000
Check or uncheck of check box or radio button	c	02000000
Mouse enter/exit	E	10000000
Right mouse button down	R	20000000

Activate or deactivate application or window	A	40000000
System color change	s	80000000

BBj always reports several common events. It is the developer's responsibility to specify in the event masks which additional optional events to report. For example, if the window will contain radio buttons to which the program must respond, then the event mask should enable the reporting of radio button selection.

BBj allows the programmer to disable/enable the windows/dialogs for each application or to evaluate the reported events by context. Each application can itself contain several window and/or dialogs. For ease of programming, BBj reports all events from all windows through a single queue.

It is the developer's responsibility to scan the event queue, identify the reported events, and initiate the appropriate responses. Typically, developers use a looping structure such as SWITCH to scan the SYSGUI device for events, and then use a selection structure to determine what the events were and what responses are appropriate from the application.

BBjSysGui Object

Back to top

BBj provides an object-oriented interface for interacting with various BBj system objects, including the SYSGUI device. This object-oriented interface provides much of the same functionality as the original Visual PRO/5 mnemonics and functions, but in a more intuitive syntax, as shown in the following example:

Mnemonic syntax	Object-oriented syntax

`GUI=UNT`
`OPEN (GUI)"X0"`
 	
`PRINT(GUI)'WINDOW'(99,99,100,100,"",$00010003$)`
`PRINT(GUI)'BUTTON'(1,10,30,80,30,"OK",$$)`
`DIM INFO$:"X:I(2),Y:I(2),W:U(2),H:U(2)"`
`LET INFO$=CTRL(GUI,1,0)`
`PRINT "Button size is",INFO.W,INFO.H`
	`GUI=UNT`
	`OPEN (GUI)"X0"`
	`GUI!=BBJAPI().getSysGui()`
	`WINDOW!=GUI!.addWindow(99,99,100,100," ")`
	`BUTTON!=WINDOW!.addButton(1,10,30,80,30,"OK")`
	`LET W=BUTTON!.getWidth()`
	`LET H=BUTTON!.getHeight()`
	`PRINT "Button size is",W,H`

BBj Callback Model

Back to top

In addition to supporting the Visual PRO/5 event loop paradigm, BBj also implements an object-oriented event model based on callbacks. In this model, the developer registers callbacks (basically, subroutines) to invoke when detecting a particular event. Events only fire for registered callbacks, reducing communications overhead. This is an important consideration over slower client/server connections.

`SYSGUI=UNT`

```
OPEN (SYSGUI)"X0"
SYSGUI!=bbjapi().getSysGui()
WINDOW!=SYSGUI!.addWindow(100,100,100,100,"Window")
OK!=WINDOW!.addButton(1,10,30,80,30,"OK")
WINDOW!.setCallback(WINDOW!.ON_CLOSE,"EOJ")
OK!.setCallback(BUTTON!.ON_BUTTON_PUSH,"OK")
PROCESS_EVENTS; REM ' Event Loop
EOJ:
STOP
OK:
I=MSGBOX("User clicked the OK button")
RETURN
```

BBj Custom Object Event Model

Back to top

BBj 6.0 introduced user-defined custom objects and event objects. These new features enable the developer to write completely object-oriented programs in BBj. The following sample shows a simple window and push button implemented as a custom object:

```
sysgui = unt
open (sysgui)"X0"

declare  Sample  Sample!
Sample!  =  new  Sample()
Sample!.go()
release

class public Sample

field private BBjSysGui sysgui!
field private BBjTopLevelWindow Window!
field private BBjButton OK!

method public Sample()
#sysgui! = bbjapi().getSysGui()
methodend

method public void initControls()
#Window! = #sysgui!.addWindow(100,100,100,100,"Window")
#OK! = #Window!.addButton(1,10,30,80,30,"OK")
methodend

method public void initEvents()
#OK!.setCallback(#OK!.ON_BUTTON_PUSH,#this!,"doOK")
methodend

method public void go()
#initControls()
#initEvents()
#Window!.setCallback(#Window!.ON_CLOSE,"eoj")
```

```
process_events

eoj:
#Window!.destroy()
methodend

method public void doOK(BBjButtonPushEvent event!) i =
msgbox("User clicked OK")
methodend

classend
```

At first glance, this sample does not appear to be an improvement over the previous version; it requires more than twice as many lines of code. In practice, this syntax offers two benefits:

- The code can be easier to understand, especially for programmers who are more comfortable with modern object-oriented syntax.
- BBj custom object methods implement variable scoping. This eliminates many hard-to-isolate bugs involving variable conflicts, particularly in very large programs.

Refer to A Primer for Using BBj Custom Objects for an introduction to BBj custom objects.

Sample Programs

Back to top

Each of the following samples implements a small customer master file maintenance program. The description of these programs covers the key points of each program and how it differs from the previous version. The sample programs are available for download to facilitate experimentation.

Program	Comments
cust-cui.txt	This traditional character-oriented program can run in all versions of PRO/5, Visual PRO/5, and BBj.
cust-gui.txt	This GUI program can run in all versions of Visual PRO/5 and BBj. It creates windows and controls using procedural code, and it processes events using an event loop.
cust-gui.src*	This program is equivalent to `cust-gui.txt`, but it creates windows and controls using a resource file named `cust.arc`.
cust-bbj.txt	This GUI program can run in all versions of BBj (not Visual PRO/5). It creates windows and controls using procedural code, and it processes events using callbacks to subroutines.
cust-bbj.src*	This program is equivalent to `cust-bbj.txt`, but it creates windows and controls using a resource file named `cust.arc`.
cust-obj.txt	This GUI program, organized as a BBj custom object, can run in BBj versions 6.0 and above. It creates windows and controls using procedural code, and it processes events using callbacks to object-oriented methods.
cust-obj.src*	This program is equivalent to `cust-obj.txt`, but it creates windows and controls using a resource file named `cust.arc`.
cust.gbf/src*	This AppBuilder project file (`cust.gbf`) and resulting program (`cust.src`) implement the same functionality as `cust-gui.src`.

* These GUI programs use the `cust.arc` resource file.

Character-Oriented Procedural Programming (cust-cui.txt)

Back to top

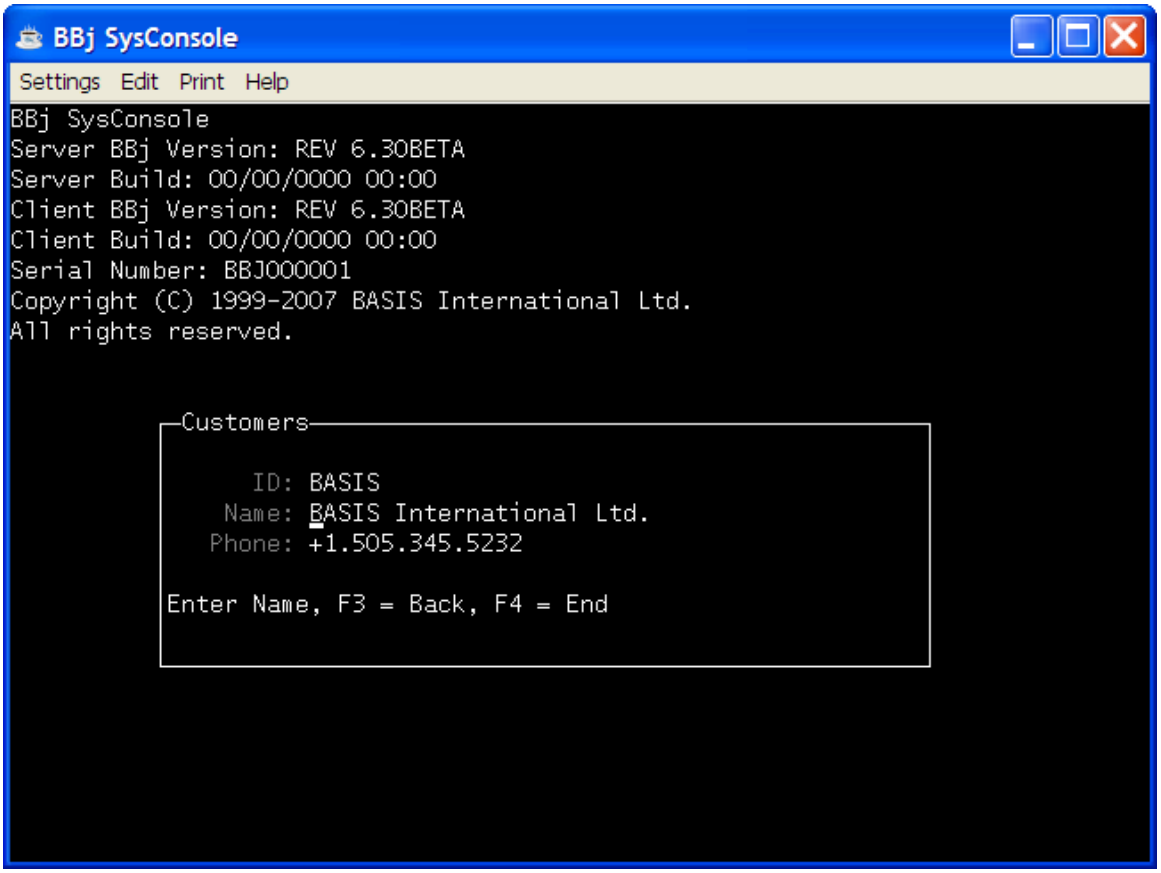
The following program implements a sample customer master file maintenance as a traditional character-oriented procedural program. This program will run in all versions of BBx from BBxProgression/3 through PRO/5, Visual PRO/5, and BBj. The developer controls the user's path through a procedural program. In this program, the user is prompted for a customer ID, customer name, phone number, and then for a selected action (update, delete, exit, etc). In a procedural program, it locks the user into the path chosen by the developer. For example, users cannot jump directly from the customer ID field to the phone number field; they must step through the customer name field first.

```
rem ' Customer master file maintenance (Character user interface)

dim customer$:"id:c(6),name:c(32),phone:c(24)"
filename$ = "customer.dat"
customer = unt
open (customer,err=makefile)filename$
goto init

makefile:
mkeyed filename$,[0:1:6],0,64
open (customer)filename$

while 1
    dread customer.id$,customer.name$,customer.phone$,err=eof
    write record(customer)customer$
    continue
eof:
    break
wend

data "BASIS","BASIS International Ltd.","+1.505.345.5232"
data "CHILE","Chile Company","+1.555.555.1212"

init:
print 'window'(10,10,55,9,"Customers")
print 'sb',
print @(6,1),"ID:"
print @(4,2),"Name:"
print @(3,3),"Phone:"
print 'cf',

id$="BASIS"
gosub fetch

while 1
id:
```

```
    print @(0,5),'cl',"Enter ID, F4 = End"
    input @(10,1),'uc',id$,'lc'
    if ctl=2 or ctl=3 then goto id
    if ctl=4 then break
    if id$="" then id$=customer.id$
    gosub fetch
name:
    print @(0,5),'cl',"Enter Name, F3 = Back, F4 = End"
    input @(10,2),name$
    if ctl=2 then goto name
    if ctl=3 or ctl=4 then goto id
    if len(cvs(name$,3)) then customer.name$ = name$
    print @(10,2),name$
phone:
    print @(0,5),'cl',"Enter Phone, F3 = Back, F4 = End"
    input@(10,3),phone$
    if ctl=2 then goto phone
    if ctl=3 then goto name
    if ctl=4 then goto id
    if len(cvs(phone$,3)) then customer.phone$ = phone$
    print @(10,3),phone$
action:
    input @(0,5),'cl',"F1 = Update, F2 = Delete, F3 = Back, F4 = End: ",*
    if ctl<2 then gosub update; continue
    if ctl=2 then gosub remove; continue
    if ctl=3 then goto phone
    if ctl=4 then break
wend

release

clear:
dim customer$:fattr(customer$)
gosub display
return

fetch:
id$ = pad(cvs(id$,7),6)
dim customer$:fattr(customer$)
let customer.id$ = id$
read record(customer,key=customer.id$,dom=notfound)customer$
notfound:
gosub display return

display:
print 'cf',
print @(10,1),cvs(customer.id$,3)
print @(10,2),cvs(customer.name$,3)
print @(10,3),cvs(customer.phone$,3)
return

update:
write record (customer)customer$
i = msgbox("Customer "+customer.id$+" updated.",0,"Updated")
gosub clear
```

```
return

remove:
remove (customer,key=customer.id$,dom=nodelete)
i = msgbox("Customer "+customer.id$+" deleted.",0,"Deleted")
nodelete:
gosub clear
return
```

This is how the program looks on a BBj SysConsole:

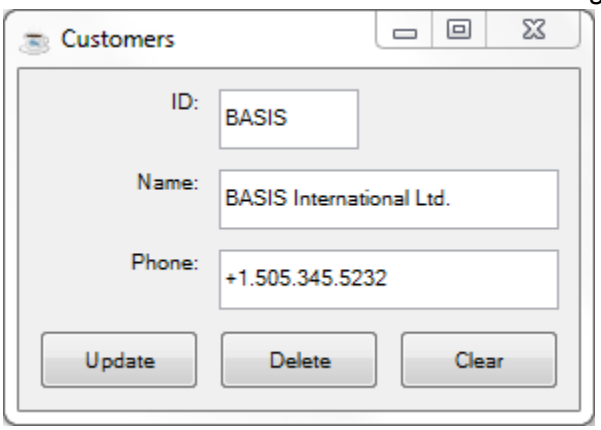


Visual PRO/5 GUI Using Mnemonics (cust-gui.txt)

Back to top

This program implements a customer master file maintenance as a Visual PRO/5 event-driven GUI program. This program will run in Visual PRO/5 and all versions of BBj. From the user's perspective, the key difference between this event-driven GUI program and the previous character-oriented program is that the user determines the flow of control through the program. The program creates a window, puts some controls on it, and waits for the user. The user is free to click or type in any control on the window; the program must be prepared to respond to the user's choices.

This is how the GUI form for all of the following sample programs looks under Windows 7:




```
rem ' Customer master file maintenance (Visual PRO/5 GUI user interface)

dim customer$:"id:c(6),name:c(32),phone:c(24)"
filename$ = "customer.dat"
customer = unt
open (customer,err=makefile)filename$
goto init

makefile:
mkeyed filename$,[0:1:6],0,64
open (customer)filename$
while 1
    dread customer.id$,customer.name$,customer.phone$,err=eof
    write record(customer)customer$
    continue
eof:
    break
wend

data "BASIS","BASIS International Ltd.","+1.505.345.5232"
data "CHILE","Chile Company","+1.555.555.1212"

init:
sysgui = unt
open (sysgui)"X0"; rem ' ALIAS X0 SYSGUI
```

```
print (sysgui)'window'(100,100,280,170,"Customers",$00010003$,$00c00000$)
print (sysgui)'text'(101,10,10,80,30,"ID:",$8000$)
print (sysgui)'edit'(102,100,10,70,30,$$,$$)
print (sysgui)'text'(103,10,50,80,30,"Name:",$8000$)
print (sysgui)'edit'(104,100,50,170,30,$$,$$)
print (sysgui)'text'(105,10,90,80,30,"Phone:",$8000$)
print (sysgui)'edit'(106,100,90,170,30,$$,$$)
print (sysgui)'button'(201,10,130,80,30,"Update",$$)
print (sysgui)'button'(202,100,130,80,30,"Delete",$$)
print (sysgui)'button'(203,190,130,80,30,"Clear",$$)

print (sysgui)'title'(102,"BASIS"),'focus'(102)
gosub fetch

dim event$:tmpl(sysgui)
repeat
    read record (sysgui,siz=10)event$
    if event.code$ = "e" and event.id = 102 then gosub toggle
    if event.code$ = "f" and event.id = 102 and event.flags = 0 gosub fetch
    if event.code$ = "B" and event.id = 201 then gosub update
    if event.code$ = "B" and event.id = 202 then gosub remove
    if event.code$ = "B" and event.id = 203 then gosub clear
until event.code$="X"

release

toggle:
id$ = cvs(ctrl(sysgui,102,1),7)
if len(id$)then print (sysgui)'enable'(201,202)
:    else print (sysgui)'disable'(201,202)
return

fetch:
id$ = pad(cvs(ctrl(sysgui,102,1),7),6)
if customer.id$ = id$ then return
dim customer$:fattr(customer$)
let customer.id$ = id$
read record(customer,key=customer.id$,dom=notfound)customer$
notfound:
gosub display
return

update:
customer.id$ = ctrl(sysgui,102,1)
customer.name$ = ctrl(sysgui,104,1)
customer.phone$ = ctrl(sysgui,106,1)
write record (customer)customer$
i = msgbox("Customer "+customer.id$+" updated.",0,"Updated")
gosub clear
return

remove:
remove (customer,key=customer.id$,dom=nodelete)
i = msgbox("Customer "+customer.id$+" deleted.",0,"Deleted")
nodelete:
```

```
gosub clear
return

clear:
dim customer$:fattr(customer$)
gosub display
print (sysgui)'focus'(102)
return

display:
print (sysgui)'title'(102,cvs(customer.id$,3))
print (sysgui)'title'(104,cvs(customer.name$,3))
print (sysgui)'title'(106,cvs(customer.phone$,3))
return
```

This is how the GUI form looks under Windows 7:

Visual PRO/5 GUI Using a Resource File (cust-gui.src)

Back to top

The previous program embedded commands to create the window and controls directly in the program. The following program is identical to the previous one, except that this version defines the window and controls in an external resource file. A resource file has many advantages:

- The same window can be used from many different programs with no duplication of code
- Multiple resource files can be defined for different languages, all affecting the program
- It enables you to make minor aesthetic changes to a program display without changing the program

```
rem ' Customer master file maintenance (Visual PRO/5 GUI user interface)

dim customer$:"id:c(6),name:c(32),phone:c(24)"
filename$ = "customer.dat"
customer = unt
open (customer,err=makefile)filename$
goto init

makefile:
mkeyed filename$,[0:1:6],0,64
open (customer)filename$
while 1
    dread customer.id$,customer.name$,customer.phone$,err=eof
    write record(customer)customer$
    continue
eof:
    break
wend

data "BASIS","BASIS International Ltd.","+1.505.345.5232"
data "CHILE","Chile Company","+1.555.555.1212"

init:
sysgui = unt
open (sysgui)"X0"; rem ' ALIAS X0 SYSGUI

if pos(" 5 "=sys) then cust = resopen("cust.brc")
if pos(" 6 "=sys) then cust = resopen("cust.arc")
cust$ = resget(cust,1,101)
print (sysgui)'resource'(len(cust$)),cust$
resclose (cust)

print (sysgui)'title'(102,"BASIS"),'focus'(102)
gosub fetch

dim event$:tmpl(sysgui)
repeat
    read record (sysgui,siz=10)event$
    if event.code$ = "e" and event.id = 102 then gosub toggle
```

```
    if event.code$ = "f" and event.id = 102 and event.flags = 0 gosub fetch
    if event.code$ = "B" and event.id = 201 then gosub update
    if event.code$ = "B" and event.id = 202 then gosub remove
    if event.code$ = "B" and event.id = 203 then gosub clear
until event.code$="X"

release

toggle:
id$ = cvs(ctrl(sysgui,102,1),7)
if len(id$) then print (sysgui)'enable'(201,202)
:    else print (sysgui)'disable'(201,202)
return

fetch:
id$ = pad(cvs(ctrl(sysgui,102,1),7),6)
if customer.id$ = id$ then return
dim customer$:fattr(customer$)
let customer.id$ = id$
read record(customer,key=customer.id$,dom=notfound)customer$
notfound:
gosub display
return

update:
customer.id$ = ctrl(sysgui,102,1)
customer.name$ = ctrl(sysgui,104,1)
customer.phone$ = ctrl(sysgui,106,1)
write record (customer)customer$
i = msgbox("Customer "+customer.id$+" updated.",0,"Updated")
gosub clear
return

remove:
remove (customer,key=customer.id$,dom=nodelete)
i = msgbox("Customer "+customer.id$+" deleted.",0,"Deleted")
nodelete:
gosub clear
return

clear:
dim customer$:fattr(customer$)
gosub display
print (sysgui)'focus'(102)
return

display:
print (sysgui)'title'(102,cvs(customer.id$,3))
print (sysgui)'title'(104,cvs(customer.name$,3))
print (sysgui)'title'(106,cvs(customer.phone$,3))
return
```

This is the cust.arc resource file:

```
VERSION "4.0"

WINDOW 101 "Customers" 100 100 280 170
BEGIN
KEYBOARDNAVIGATION
EVENTMASK 12582912
NAME "Customer"

STATICTEXT 101, "ID:", 10, 10, 80, 30
BEGIN
JUSTIFICATION 32768
NAME "ID Label"
END

EDIT 102, "", 100, 10, 70, 30 BEGIN
NAME "ID"
CLIENTEDGE
END

STATICTEXT 103, "Name:", 10, 50, 80, 30
BEGIN
JUSTIFICATION 32768
NAME "Name Label"
END

EDIT 104, "", 100, 50, 170, 30
BEGIN
NAME "Name"
CLIENTEDGE
END

STATICTEXT 105, "Phone:", 10, 90, 80, 30
BEGIN
JUSTIFICATION 32768
NAME "Phone Label"
END

EDIT 106, "", 100, 90, 170, 30
BEGIN
NAME "Phone"
CLIENTEDGE
END

BUTTON 201, "Update", 10, 130, 80, 30
BEGIN
NAME "Update"
END

BUTTON 202, "Delete", 100, 130, 80, 30
BEGIN
NAME "Delete"
```

```
END

BUTTON 203, "Clear", 190, 130, 80, 30
BEGIN
NAME "Clear"
END
END
```

BBj GUI Using Callbacks (cust-bbj.txt)

Back to top

BBj inherits the functionality of previous versions of the BBx language, but it also adds many new features. BBj 1.0 introduced object variables and an object-oriented syntax to the BBx language. In this sample, callbacks replace the event loop and deliver two distinct advantages:

- Easier-to-understand code, especially for programmers who are more comfortable with modern object-oriented syntax
- Overhead reduction, particularly over slow client/server connections, because BBj only fires events for the callbacks the developer had defined

```
rem ' Customer master file maintenance (BBj GUI user interface)

dim customer$:"id:c(6),name:c(32),phone:c(24)"
filename$ = "customer.dat"
customer = unt
open (customer,err=makefile)filename$
goto init

makefile:
mkeyed filename$,[0:1:6],0,64
open (customer)filename$
while 1
    dread customer.id$,customer.name$,customer.phone$,err=eof
    write record(customer)customer$
    continue
eof:
    break
wend

data "BASIS","BASIS International Ltd.","+1.505.345.5232"
data "CHILE","Chile Company","+1.555.555.1212"

init:
sysgui = unt
open (sysgui)"X0"; rem ' ALIAS X0 SYSGUI

sysgui! = bbjapi().getSysGui()
window! = sysgui!.addWindow(100,100,280,170,"Customers",$00010003$,$00c00000$)
window!.addStaticText(101,10,10,80,30,"ID:",$8000$)
id! = window!.addEditBox(102,100,10,70,30,$$,$$)
window!.addStaticText(103,10,50,80,30,"Name:",$8000$)
name! = window!.addEditBox(104,100,50,170,30,$$,$$)
window!.addStaticText(105,10,90,80,30,"Phone:",$8000$)
phone! = window!.addEditBox(106,100,90,170,30,$$,$$)
update! = window!.addButton(201,10,130,80,30,"Update",$$)
delete! = window!.addButton(202,100,130,80,30,"Delete",$$)
clear! = window!.addButton(203,190,130,80,30,"Clear",$$)

id!.setText("BASIS")
id!.focus()
```

```
gosub fetch

id!.setCallback(id!.ON_EDIT_MODIFY,"toggle")
id!.setCallback(id!.ON_LOST_FOCUS,"fetch")
update!.setCallback(update!.ON_BUTTON_PUSH,"update")
delete!.setCallback(delete!.ON_BUTTON_PUSH,"remove")
clear!.setCallback(clear!.ON_BUTTON_PUSH,"clear")
window!.setCallback(window!.ON_CLOSE,"eoj")

process_events

eoj:
release

toggle:
id$ = cvs(id!.getText(),7)
update!.setEnabled(len(id$))
delete!.setEnabled(len(id$))
return

fetch:
id$ = pad(cvs(id!.getText(),7),6)
if customer.id$ = id$ then return
dim customer$:fattr(customer$)
let customer.id$ = id$
read record(customer,key=customer.id$,dom=notfound)customer$
notfound:
gosub display
return

update:
customer.id$ = ctrl(sysgui,102,1)
customer.name$ = ctrl(sysgui,104,1)
customer.phone$ = ctrl(sysgui,106,1)
write record (customer)customer$
i = msgbox("Customer "+customer.id$+" updated.",0,"Updated")
gosub clear
return

remove:
remove (customer,key=customer.id$,dom=nodelete)
i = msgbox("Customer "+customer.id$+" deleted.",0,"Deleted")
nodelete:
gosub clear
return

clear:
dim customer$:fattr(customer$)
gosub display
id!.focus()
return

display:
```

```
id!.setText(cvs(customer.id$,3))
name!.setText(cvs(customer.name$,3))
phone!.setText(cvs(customer.phone$,3))
return
```

BBj GUI Using Callbacks and Resource File (cust-bbj.src)

Back to top

In this sample, the mnemonics have been replaced by object method calls to retrieve window and control information from the `cust.arc` file:

```
rem ' Customer master file maintenance (BBj GUI user interface)

dim customer$:"id:c(6),name:c(32),phone:c(24)"
filename$ = "customer.dat"
customer = unt
open (customer,err=makefile)filename$
goto init

makefile:
mkeyed filename$,[0:1:6],0,64
open (customer)filename$
while 1
    dread customer.id$,customer.name$,customer.phone$,err=eof
    write record(customer)customer$
    continue
eof:
    break
wend

data "BASIS","BASIS International Ltd.","+1.505.345.5232"
data "CHILE","Chile Company","+1.555.555.1212"

init:
sysgui=unt
open(sysgui)"X0"
sysGui!=BBjAPI().getSysGui()
window!=sysGui!.createTopLevelWindow(sysGui!.resOpen("cust.arc"),101)

id!=window!.getControl(102)
name!=window!.getControl(104)
phone!=window!.getControl(106)
update!=window!.getControl(201)
delete!=window!.getControl(202)
clear!=window!.getControl(203)

id!.setText("BASIS")
id!.focus()

gosub fetch

id!.setCallback(id!.ON_EDIT_MODIFY,"toggle")
id!.setCallback(id!.ON_LOST_FOCUS,"fetch")
update!.setCallback(update!.ON_BUTTON_PUSH,"update")
delete!.setCallback(delete!.ON_BUTTON_PUSH,"remove")
clear!.setCallback(clear!.ON_BUTTON_PUSH,"clear")
window!.setCallback(window!.ON_CLOSE,"eoj")
```

```
process_events

eoj:
release

toggle:
id$ = cvs(id!.getText(),7)
update!.setEnabled(len(id$))
delete!.setEnabled(len(id$))
return

fetch:
id$ = cvs(id!.getText(),6)
if customer.id$ = id$ then return
dim customer$:fattr(customer$)
let customer.id$ = id$
read record(customer,key=customer.id$,dom=notfound)customer$
notfound:
gosub display
return

update:
customer.id$ = id!.getText()
customer.name$ = name!.getText()
customer.phone$ = phone!.getText()
if len(cvs(customer_id$,3))
    write record (customer)customer$
    i = msgbox("Customer "+customer.id$+" updated.",0,"Updated")
else
    i = msgbox("Empty ID. Record not written.",0,"Not Written")
endif
gosub clear
return

remove:
remove (customer,key=customer.id$,dom=nodelete)
i = msgbox("Customer "+customer.id$+" deleted.",0,"Deleted")
nodelete:
gosub clear
return

clear:
dim customer$:fattr(customer$)
gosub display
id!.setEnabled(1)
id!.focus()
return

display:
id!.setText(cvs(customer.id$,3))
name!.setText(cvs(customer.name$,3))
phone!.setText(cvs(customer.phone$,3))
return
```

BBj GUI Using Objects (cust-obj.txt)

Back to top

BBj 6.0 introduced custom objects – developer-defined objects implemented in BBj. This version of the sample program implements the same GUI interface as the previous versions, it uses a custom object. For an introduction to BBj custom objects, refer to the BBjCustomObjectsTutorial.

```
rem ' Customer master file maintenance (BBj GUI user interface - objects)

sysgui = unt
open (sysgui)"X0"; rem ' ALIAS X0 SYSGUI

declare Sample Sample!
Sample! = new Sample()
Sample!.edit()
release

class public Sample

field private BBjSysGui sysgui!
field private BBjNumber CustFile
field private BBjTemplatedString Customer!
field private BBjTopLevelWindow Window!
field private BBjEditBox ID!
field private BBjEditBox Name!
field private BBjEditBox Phone!
field private BBjButton Update!
field private BBjButton Delete!
field private BBjButton Clear!

method public Sample()
    #sysgui! = bbjapi().getSysGui()
methodend

method public void openFile()
    Template$ = "ID:c(6),name:c(32),phone:c(24)"
    #Customer! = bbjapi().makeTemplatedString(Template$)
    Filename$ = "Customer.dat"
    #CustFile = unt
    open (#CustFile,err=makefile)Filename$
methodret
makefile:
    mkeyed Filename$,[0:1:6],0,64
    open (#CustFile)Filename$
    dim Customer$:Template$
    while 1
        dread Customer.ID$,Customer.name$,Customer.phone$,err=eof
        write record(#CustFile)Customer$
        continue
eof:
```

```
        break
    wend
    data "BASIS","BASIS International Ltd.","+1.505.345.5232"
    data "CHILE","Chile Company","+1.555.555.1212"
methodend

method public void initControls()
    #Window! = #sysgui!.addWindow(100,100,280,170,
:       "Customers",$00010003$,$00c00000$)
    #Window!.addStaticText(101,10,10,80,30,"ID:",$8000$)
    #ID! = #Window!.addEditBox(102,100,10,70,30,$$,$$)
    #Window!.addStaticText(103,10,50,80,30,"Name:",$8000$)
    #Name! = #Window!.addEditBox(104,100,50,170,30,$$,$$)
    #Window!.addStaticText(105,10,90,80,30,"Phone:",$8000$)
    #Phone! = #Window!.addEditBox(106,100,90,170,30,$$,$$)
    #Update! = #Window!.addButton(201,10,130,80,30,"Update",$$)
    #Delete! = #Window!.addButton(202,100,130,80,30,"Delete",$$)
    #Clear! = #Window!.addButton(203,190,130,80,30,"Clear",$$)
methodend

method public void initEvents()
    #ID!.setCallback(#ID!.ON_EDIT_MODIFY,#this!,"doToggle")
    #ID!.setCallback(#ID!.ON_LOST_FOCUS,#this!,"doFetch")
    #Update!.setCallback(#Update!.ON_BUTTON_PUSH,#this!,"doUpdate")
    #Delete!.setCallback(#Delete!.ON_BUTTON_PUSH,#this!,"doDelete")
    #Clear!.setCallback(#Clear!.ON_BUTTON_PUSH,#this!,"doClear")
methodend

method public void initDemo()
    #ID!.setText("BASIS")
    #fetch()
    #ID!.focus()
methodend

method public void edit()
    #openFile()
    #initControls()
    #initEvents()
    #initDemo()
    #Window!.setCallback(#Window!.ON_CLOSE,"eoj")

    process_events

eoj:
    #Window!.destroy()
methodend

method public void doToggle(BBjEditModifyEvent event!)
    ID$ = cvs(#ID!.getText(),7)
    #Update!.setEnabled(len(ID$))
    #Delete!.setEnabled(len(ID$))
methodend

method public void doFetch(BBjLostFocusEvent event!)
```

```
    #fetch()
methodend

method public void fetch()
    ID$ = pad(cvs(#ID!.getText(),7),6)
    if #Customer!.getFieldAsString("ID") = ID$ then methodret
    #Customer!.setFieldValue("ID",ID$)
    read record(#CustFile,key=ID$,dom=notfound)Customer$
    #Customer!.setString(Customer$)
notfound:
    #display()
methodend

method public void doUpdate(BBjButtonPushEvent event!)
    #Customer!.setFieldValue("ID",#ID!.getText())
    #Customer!.setFieldValue("name",#Name!.getText())
    #Customer!.setFieldValue("phone",#Phone!.getText())
    write record (#CustFile)#Customer!.getString()
    i = msgbox("Customer "+#Customer!.getFieldAsString("ID")+
:         " updated.",0,"Updated")
    #clear()
methodend

method public void doDelete(BBjButtonPushEvent event!)
    ID$ = #Customer!.getFieldAsString("ID")
    remove (#CustFile,key=ID$,dom=nodelete)
    i = msgbox("Customer "+ID$+" deleted.",0,"Deleted")
nodelete:
    #clear()
methodend

method public void doClear(BBjButtonPushEvent event!)
    #clear()
methodend

method public void clear()
    #Customer! = bbjapi().makeTemplatedString(#Customer!.fattr())
    #display()
    #ID!.focus()
methodend

method public void display()
    #ID!.setText(cvs(#Customer!.getFieldAsString("ID"),3))
    #Name!.setText(cvs(#Customer!.getFieldAsString("name"),3))
    #Phone!.setText(cvs(#Customer!.getFieldAsString("phone"),3))
methodend

classend
```

BBj GUI Using Objects and Resource File (cust-obj.src)

Back to top

This version of the sample program is the same as the previous one in that it uses a custom object, but this version also makes use of the cust.arc file.

```
rem ' Customer master file maintenance (BBj GUI user interface - objects)

sysgui = unt
open (sysgui)"X0"; rem ' ALIAS X0 SYSGUI

declare Sample Sample!
Sample! = new Sample()
Sample!.edit()
release

class public Sample

field private BBjSysGui sysgui!
field private BBjNumber CustFile
field private BBjTemplatedString Customer!
field private BBjTopLevelWindow Window!
field private BBjControl ID!
field private BBjControl Name!
field private BBjControl Phone!
field private BBjControl Update!
field private BBjControl Delete!
field private BBjControl Clear!

method public Sample()
    #sysgui! = bbjapi().getSysGui()
methodend

method public void openFile()
    Template$ = "ID:c(6),name:c(32),phone:c(24)"
    #Customer! = bbjapi().makeTemplatedString(Template$)
    Filename$ = "Customer.dat"
    #CustFile = unt
    open (#CustFile,err=makefile)Filename$
methodret
makefile:
    mkeyed Filename$,[0:1:6],0,64
    open (#CustFile)Filename$
    dim Customer$:Template$
    while 1
        dread Customer.ID$,Customer.name$,Customer.phone$,err=eof
        write record(#CustFile)Customer$
        continue
eof:
        break
    wend

    data "BASIS","BASIS International Ltd.","+1.505.345.5232"
```

```
    data "CHILE","Chile Company","+1.555.555.1212"
methodend

method public void initControls()
    #Window!=#sysgui!.createTopLevelWindow(#sysgui!.resOpen("cust.arc"),101)
    #ID!=#Window!.getControl(102)
    #Name!=#Window!.getControl(104)
    #Phone!=#Window!.getControl(106)
    #Update!=#Window!.getControl(201)
    #Delete!=#Window!.getControl(202)
    #Clear!=#Window!.getControl(203)
methodend

method public void initEvents()
    #ID!.setCallback(#ID!.ON_EDIT_MODIFY,#this!,"doToggle")
    #ID!.setCallback(#ID!.ON_LOST_FOCUS,#this!,"doFetch")
    #Update!.setCallback(#Update!.ON_BUTTON_PUSH,#this!,"doUpdate")
    #Delete!.setCallback(#Delete!.ON_BUTTON_PUSH,#this!,"doDelete")
    #Clear!.setCallback(#Clear!.ON_BUTTON_PUSH,#this!,"doClear")
methodend

method public void initDemo()
    #ID!.setText("BASIS")
    #fetch()
    #ID!.focus()
methodend

method public void edit()
    #openFile()
    #initControls()
    #initEvents()
    #initDemo()
    #Window!.setCallback(#Window!.ON_CLOSE,"eoj")

process_events

eoj:
    #Window!.destroy()
methodend

method public void doToggle(BBjEditModifyEvent event!)
    ID$ = cvs(#ID!.getText(),7)
    #Update!.setEnabled(len(ID$))
    #Delete!.setEnabled(len(ID$))
methodend

method public void doFetch(BBjLostFocusEvent event!)
    #fetch()
methodend

method public void fetch()
    ID$ = pad(cvs(#ID!.getText(),7),6)
    if #Customer!.getFieldAsString("ID") = ID$ then methodret
    #Customer!.setFieldValue("ID",ID$)
    read record(#CustFile,key=ID$,dom=notfound)Customer$
```

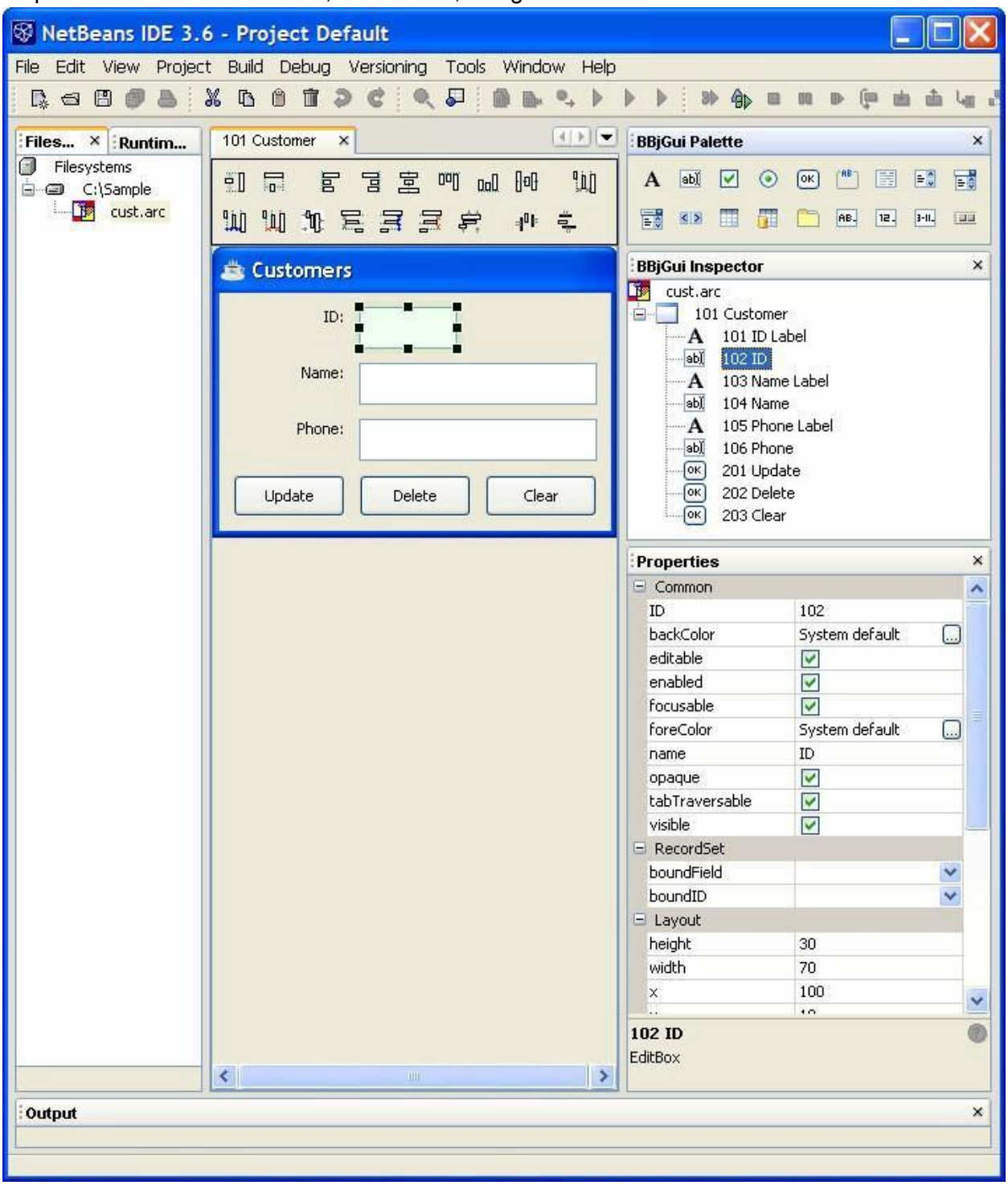
```
    #Customer!.setString(Customer$)
notfound:
    #display()
methodend

method public void doUpdate(BBjButtonPushEvent event!)
    #Customer!.setFieldValue("ID",#ID!.getText())
    #Customer!.setFieldValue("name",#Name!.getText())
    #Customer!.setFieldValue("phone",#Phone!.getText())
    if cvs(#ID!.getText(),3)<>""
        write record (#CustFile)#Customer!.getString()
        i = msgbox("Customer "+#Customer!.getFieldAsString("ID")+
:           " updated.",0,"Updated")
    else
        i = msgbox("Empty ID. Record not written.",0,"Not Written")
    endif
    #clear()
methodend

method public void doDelete(BBjButtonPushEvent event!)
    ID$ = #Customer!.getFieldAsString("ID")
    remove (#CustFile,key=ID$,dom=nodelete)
    i = msgbox("Customer "+ID$+" deleted.",0,"Deleted")
nodelete:
    #clear()
methodend

method public void doClear(BBjButtonPushEvent event!)
    #clear()
methodend

method public void clear()
    #Customer! = bbjapi().makeTemplatedString(#Customer!.fattr())
    #display()
    #ID!.focus()
methodend

method public void display()
    #ID!.setText(cvs(#Customer!.getFieldAsString("ID"),3))
    #Name!.setText(cvs(#Customer!.getFieldAsString("name"),3))
    #Phone!.setText(cvs(#Customer!.getFieldAsString("phone"),3))
methodend

classend
```

AppBuilder Project (cust.gbf)

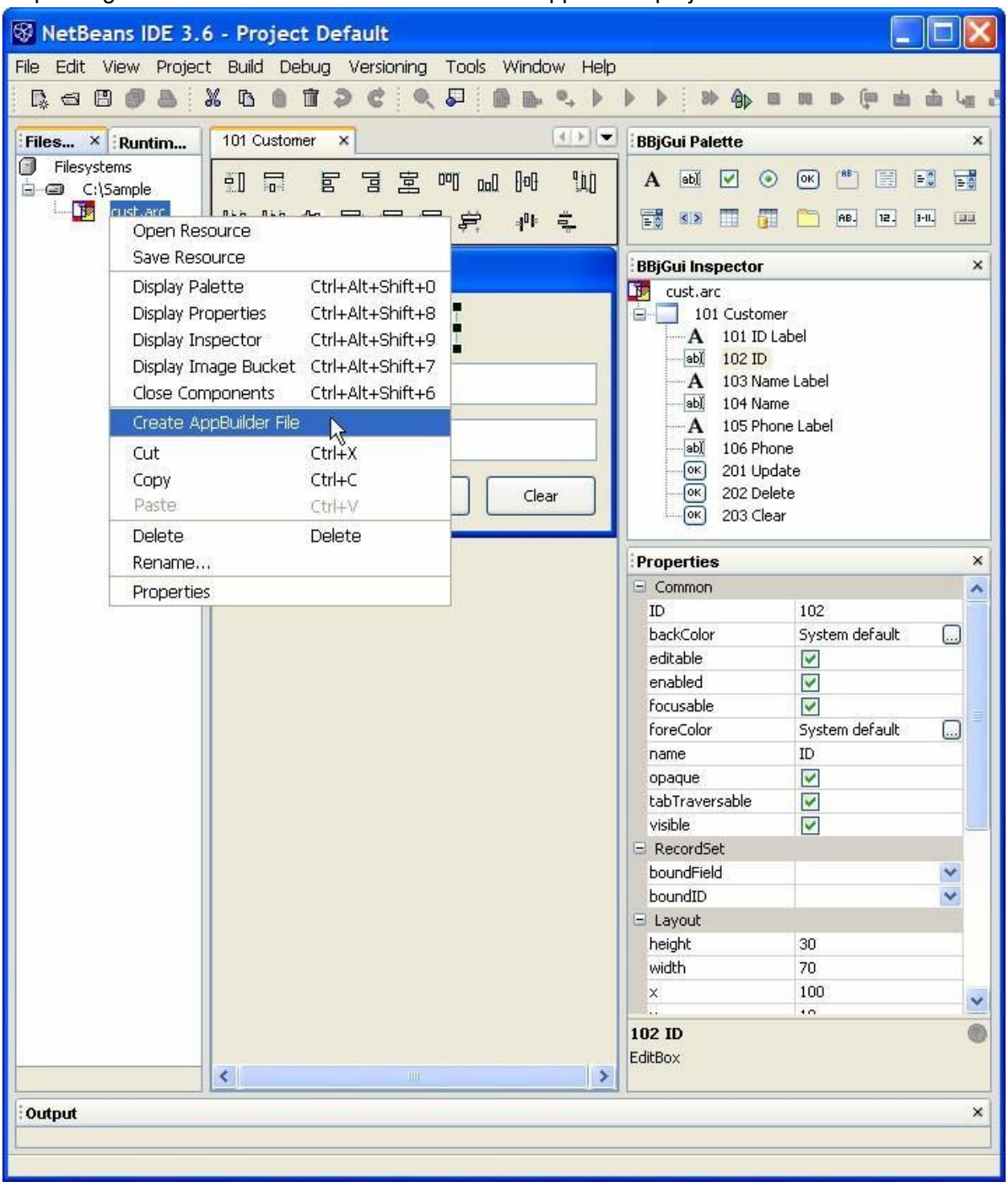
Back to top

All of the previous samples assume that the developer will type a program into a text editor, compile it, and run it. This final example shows how to implement the same customer master file maintenance program using GUIBuilder in Visual PRO/5 or BBj, or AppBuilder, a component of the BASIS IDE first introduced in BBj 6. Using the BASIS IDE, the developer would go through the following steps:

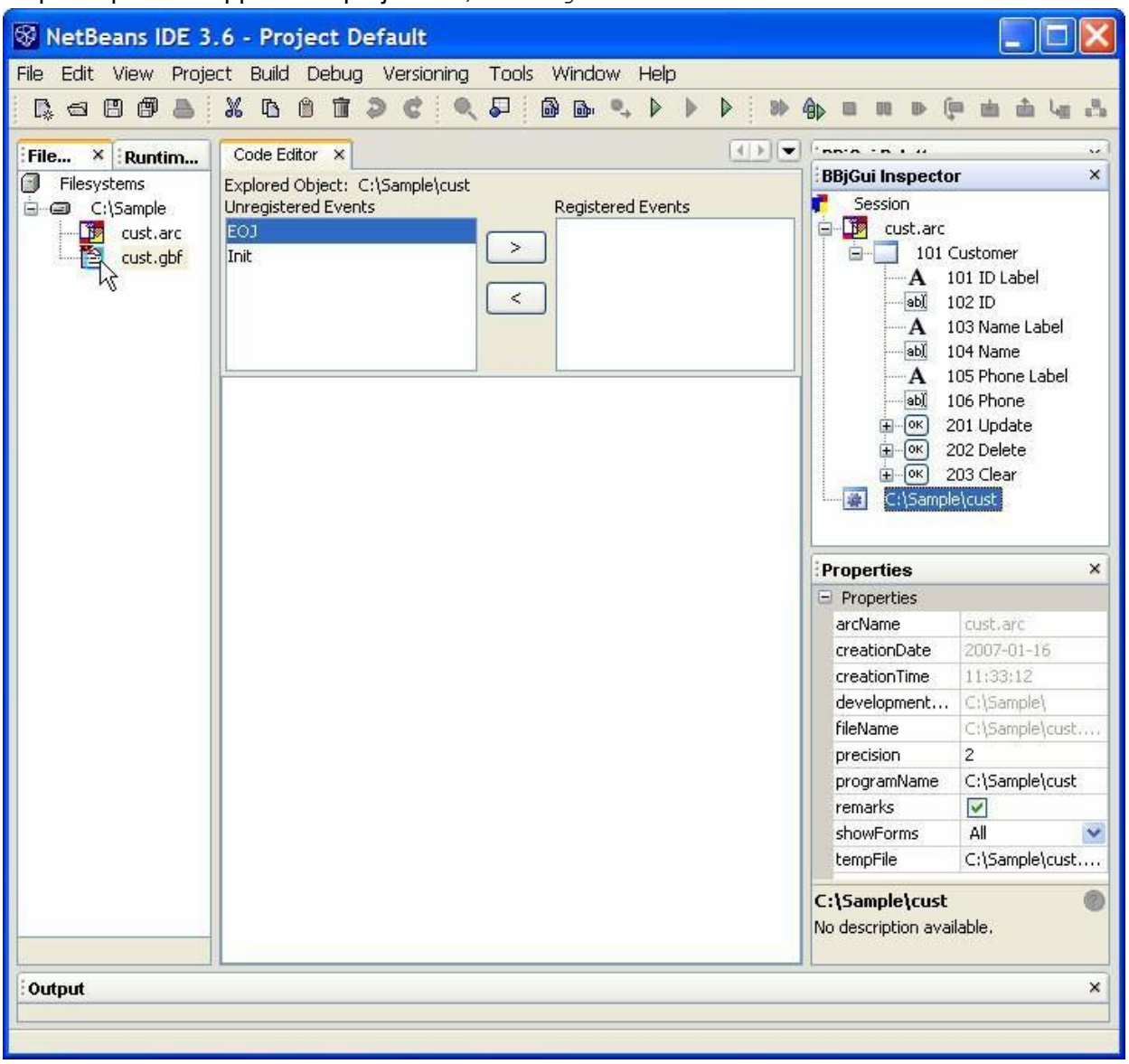
Step 1: Create the resource file, `cust.arc`, using the BASIS IDE FormBuilder.



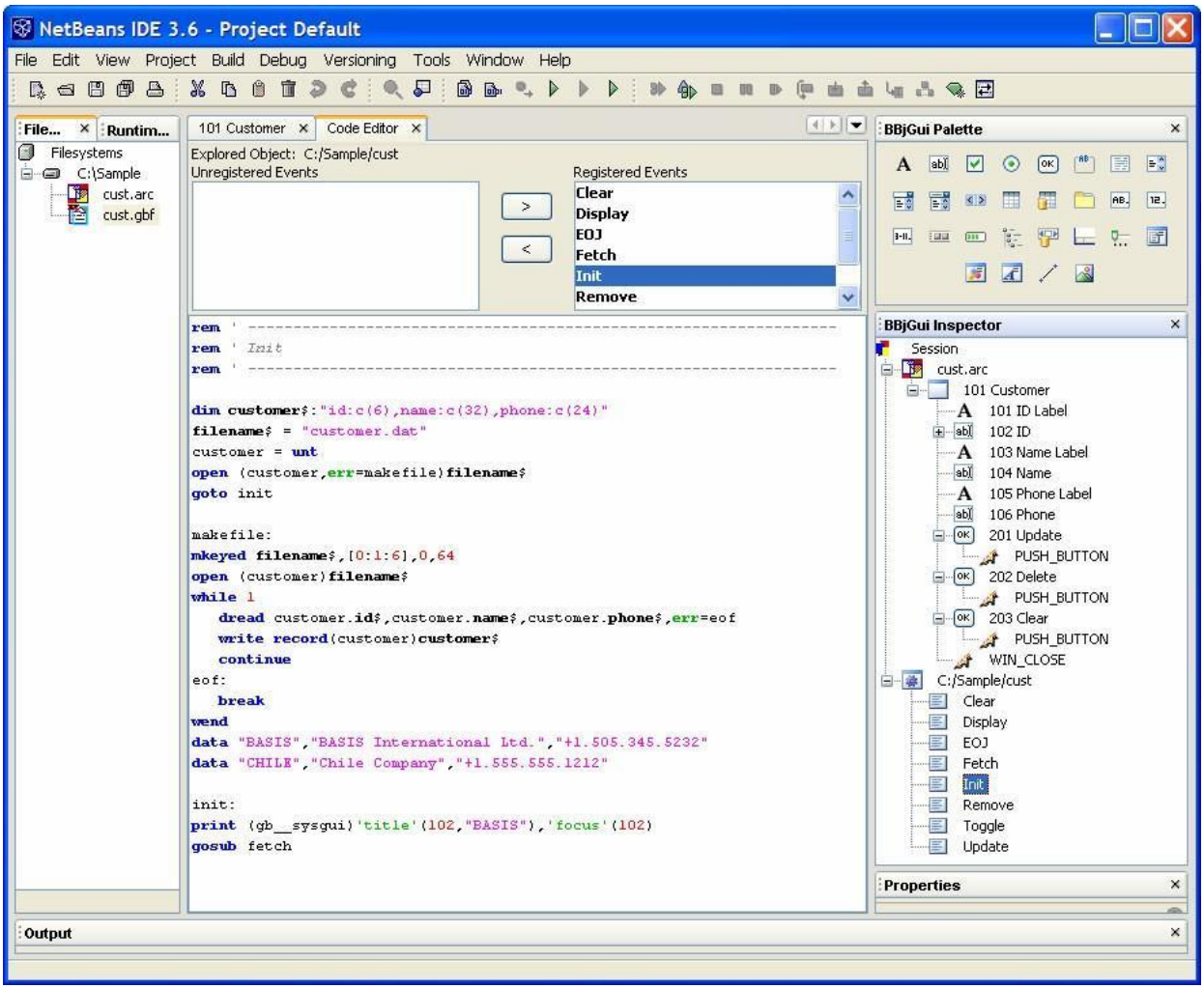
Step 2: Right-click on the resource file to create an AppBuilder project file.



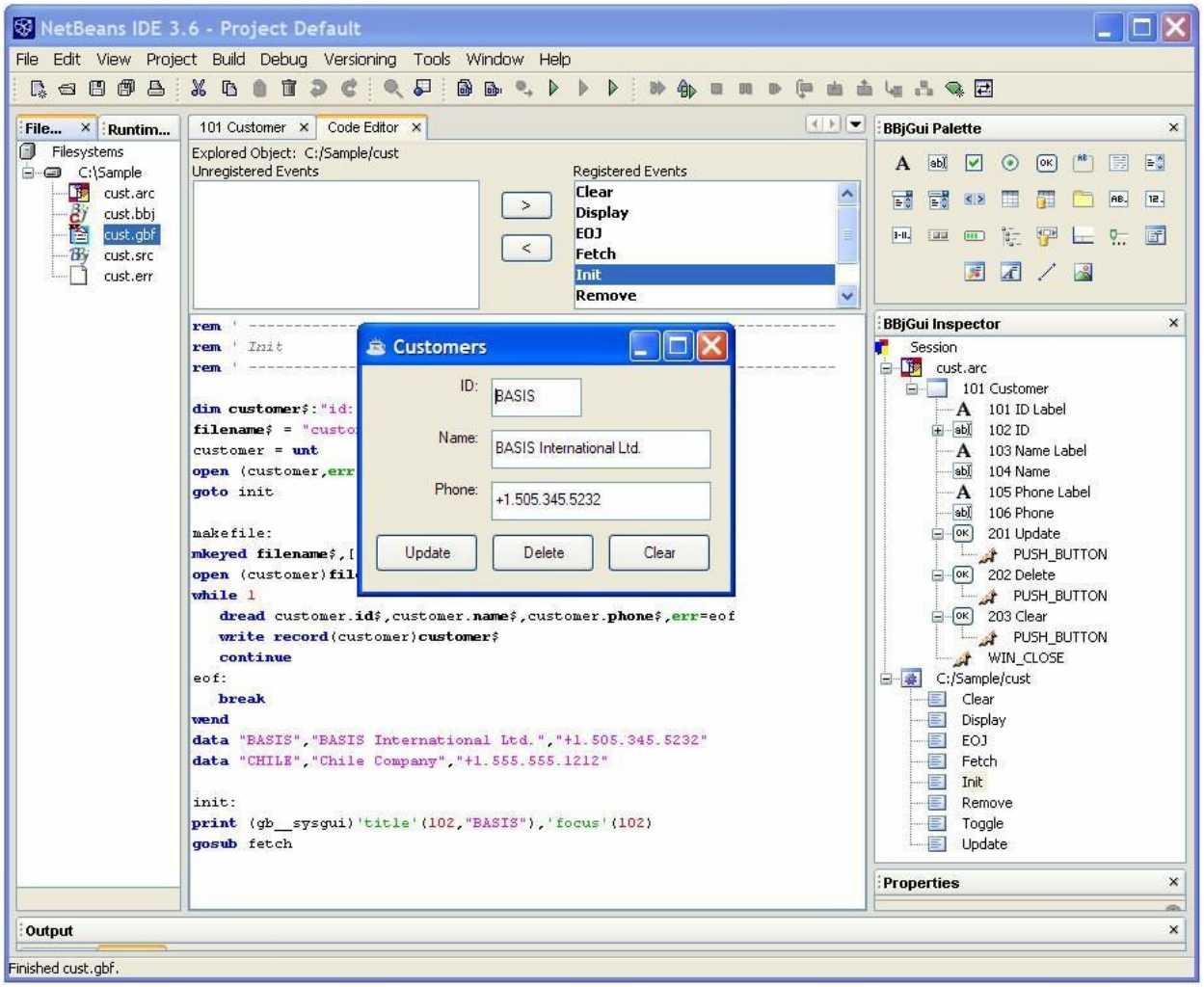
Step 3: Open the AppBuilder project file, `cust.gbf`.



Step 4: Define all required subroutines and event handlers.



Step 5: Build ([F11]) the program (cust.src) and run it ([F6]).



For more information, refer to the following tutorials in the online documentation:

- AppBuilder Tutorial
- FormBuilder Tutorial
- Working With Child Windows Tutorial

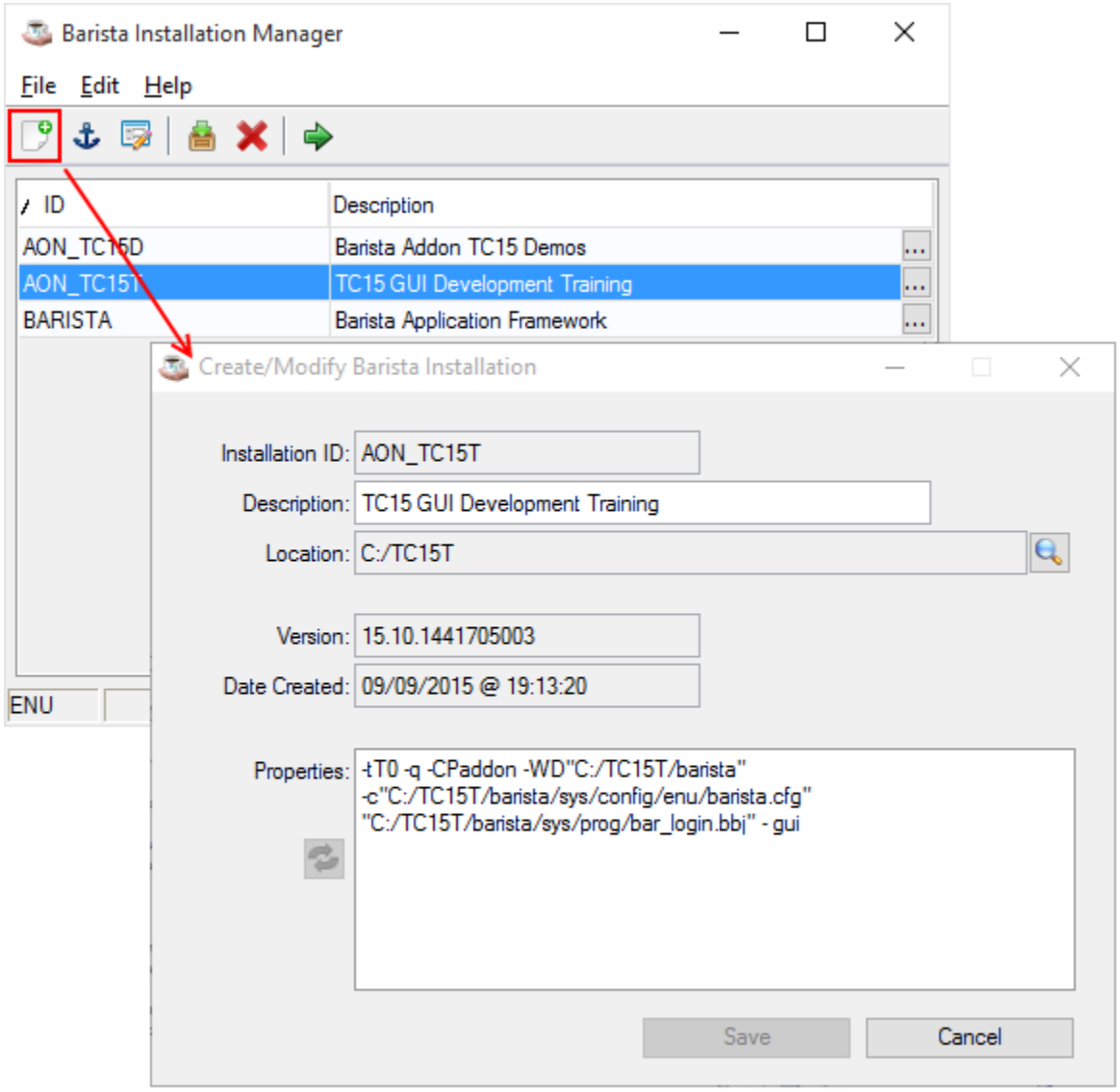
Creating the Customer Form in Barista

Back to top

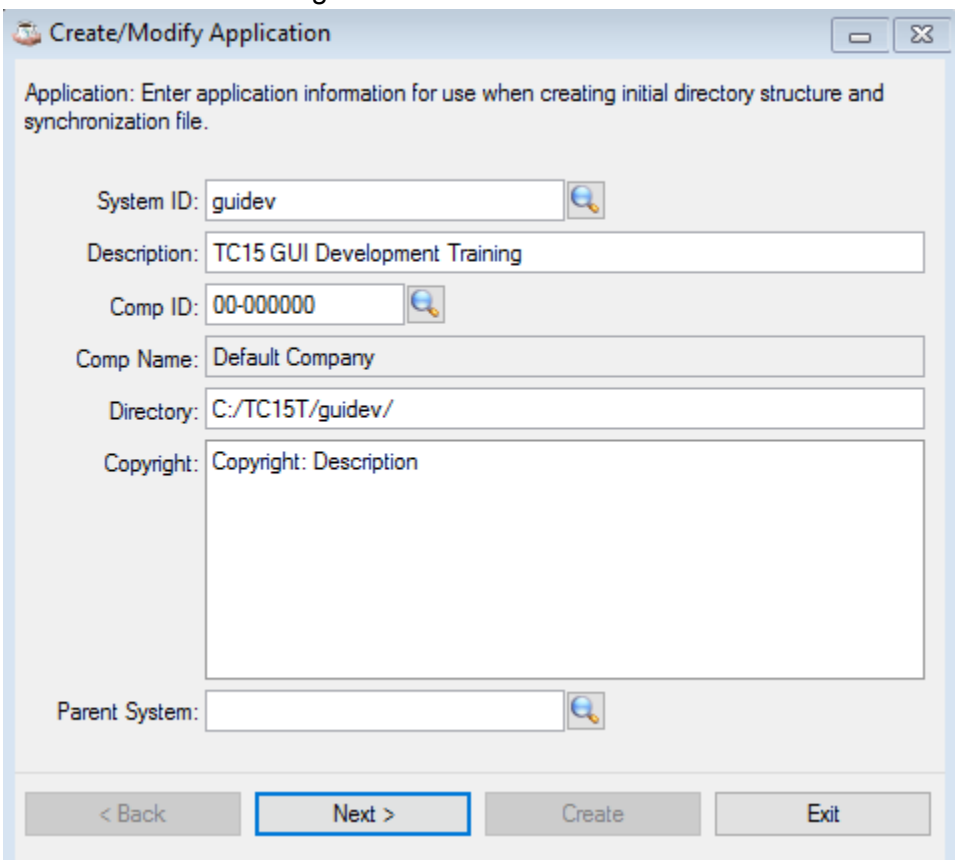
You can create the Customer form using one of two methods within the Barista Application Framework®. Because Barista is a dictionary-based tool, getting the Element and Table definitions into the dictionary is the key to either method.

- Build your dictionary from scratch using Barista's Element Types and Table definition forms.
- Import from the BASIS DD into the Barista dictionary, using an existing DD, or creating one by supplying string templates for your tables.

Step 1: Create a new instance of Barista with the Barista Installation Manager. Each instance of Barista is associated with a new Barista and BASIS dictionary.



Step 2: Barista's Create Application Wizard builds the project structure and incorporates it into Barista and BASIS configuration files.

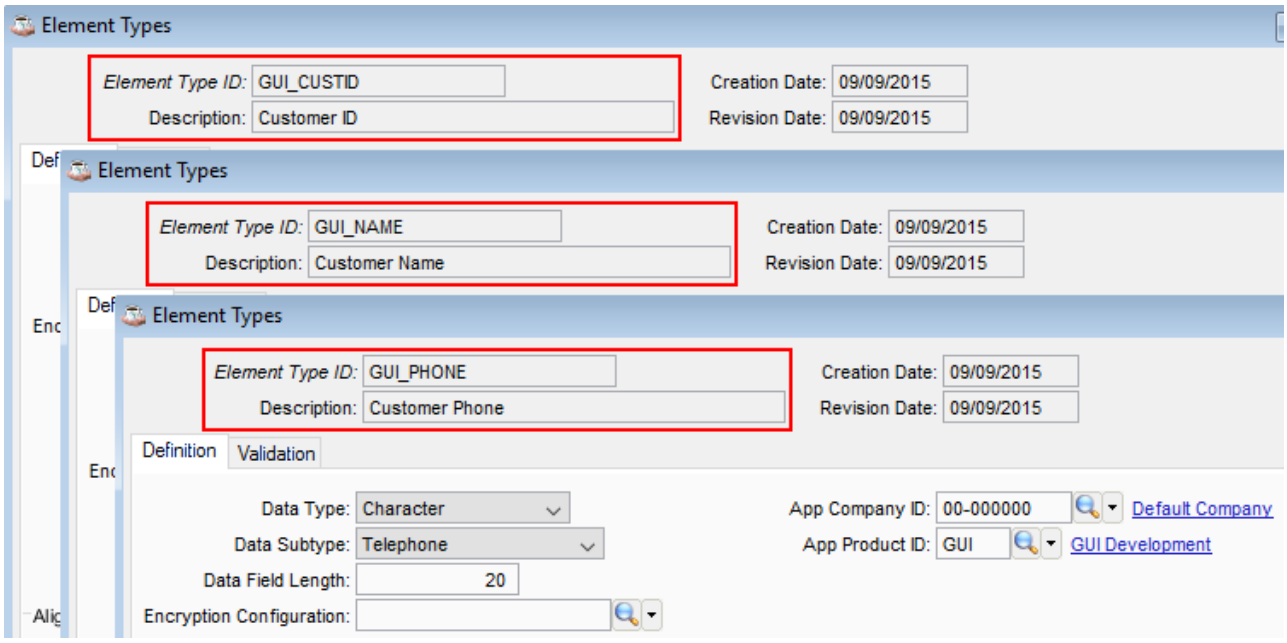


Build from Scratch

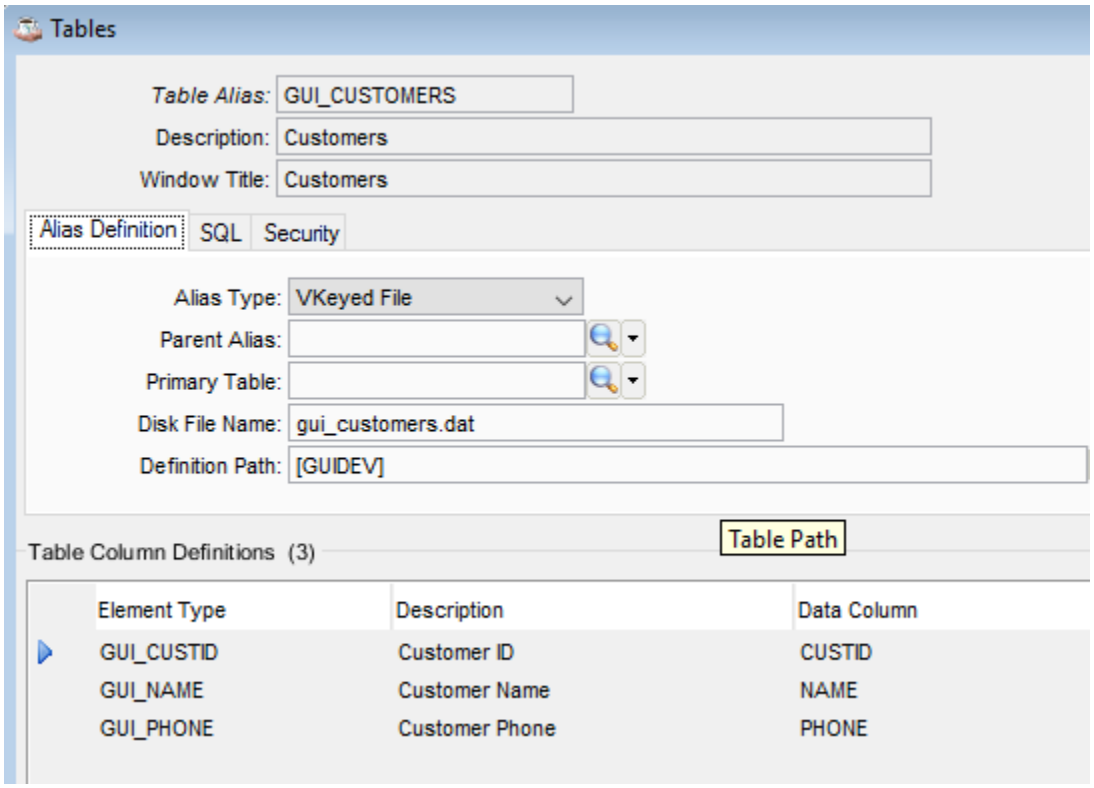
Back to top

If you don't have an existing BASIS DD, SQL database, or text file schema from which to import element and table definitions, or if you simply prefer to create your database from scratch, begin by creating Element Types and then use those elements in a Table definition. Once the table definition is in place, you have a functional form as well.

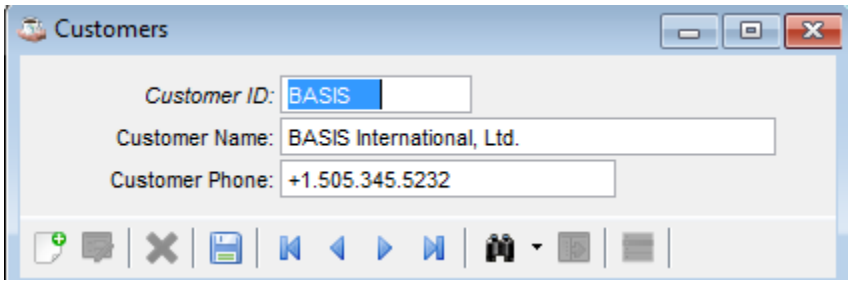
Step 1: Lay the foundation by creating your Element Types.



Step 2: Next, use those elements to create a Table Definition:



Step 3: Build and run the form. Barista handles record I/O and navigation, so you have a functional form without writing any code. In addition, the framework provides built-in inquiry and reporting.

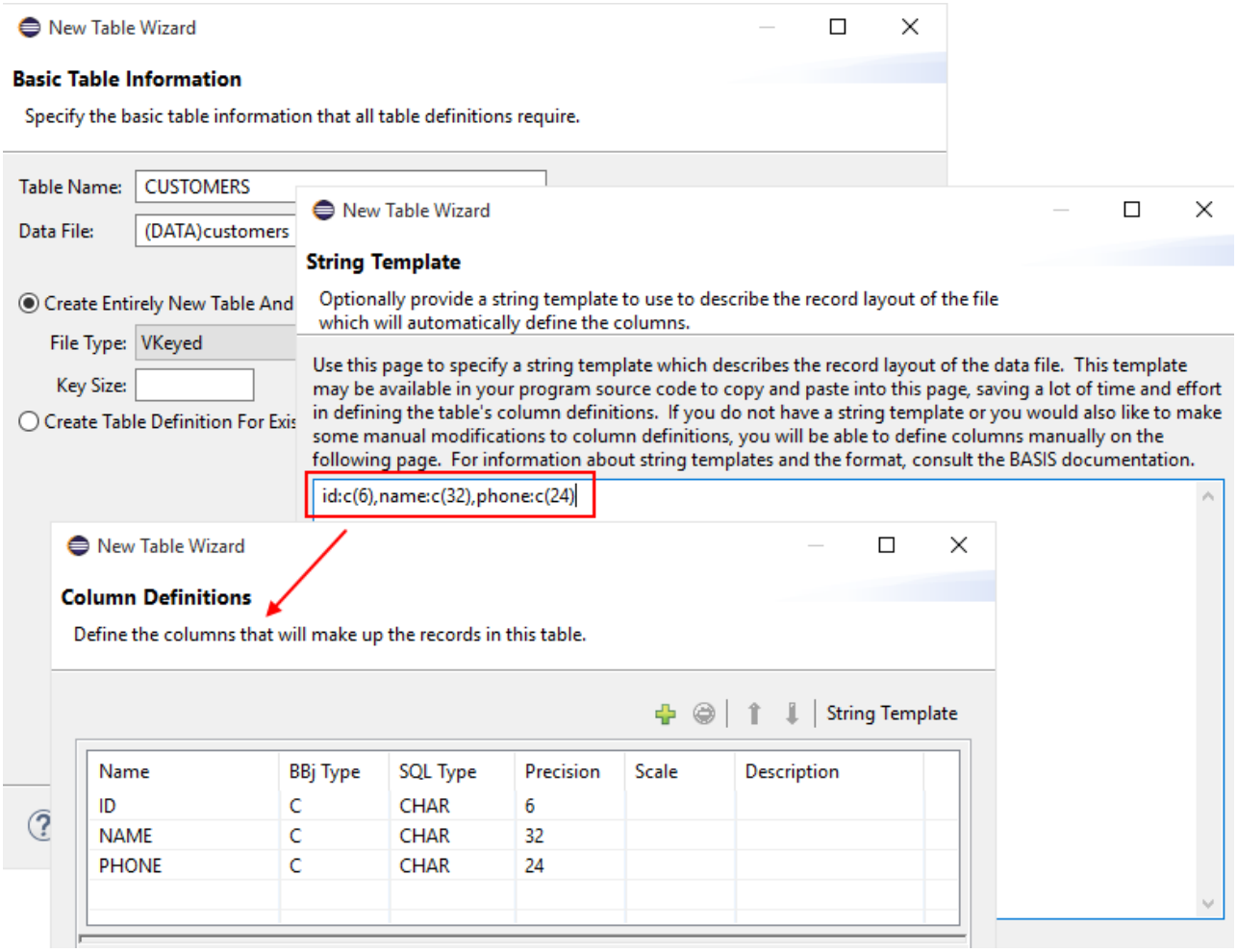


Import to Barista Dictionary

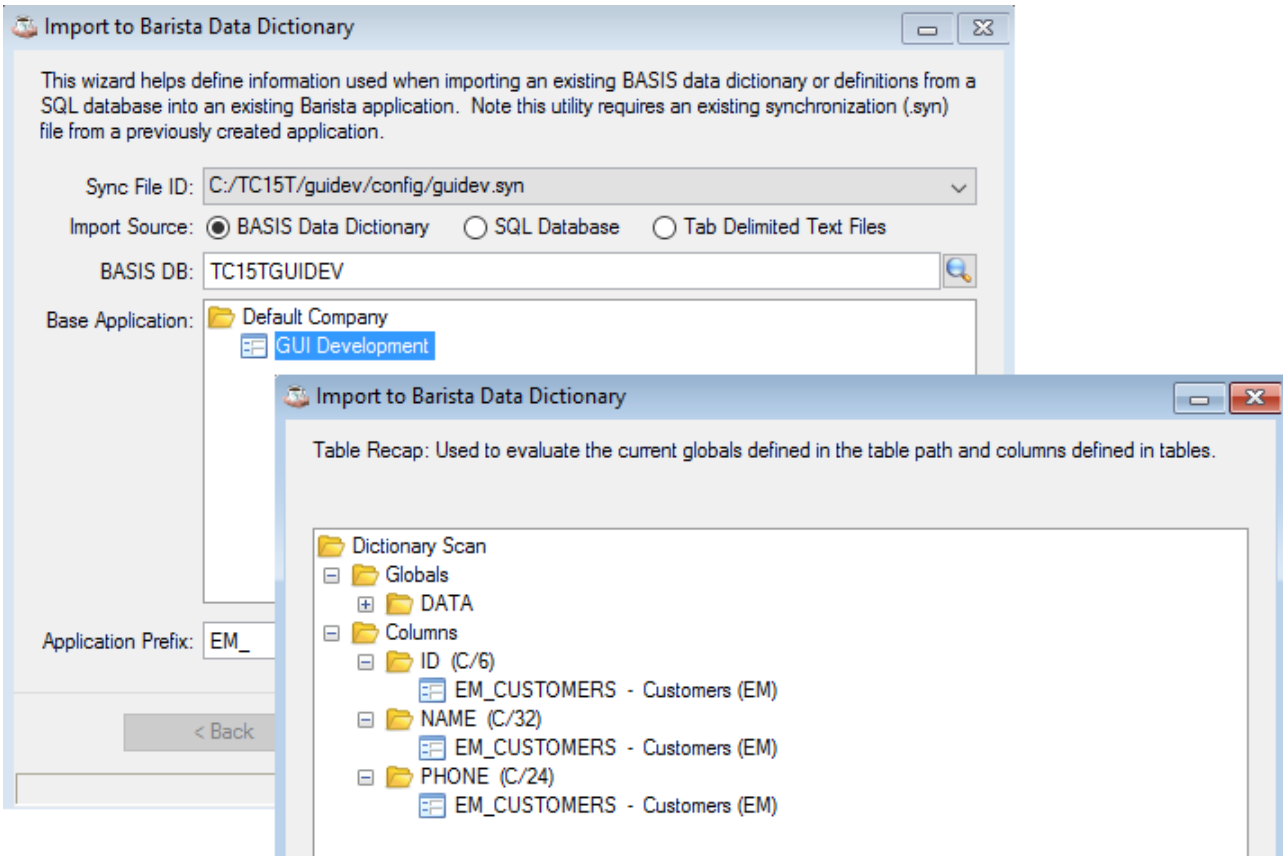
Back to top

Using the Import to Barista Dictionary tool, you can automate the task of getting the basic element and table definitions into the Barista dictionary. The most common option is to import from an existing BASIS DD. You can shortcut building a BASIS DD too, if you have string templates that you can harvest from your existing legacy code. In addition to importing from a BASIS DD, the import tool can also import from an SQL database or tab-delimited text files.

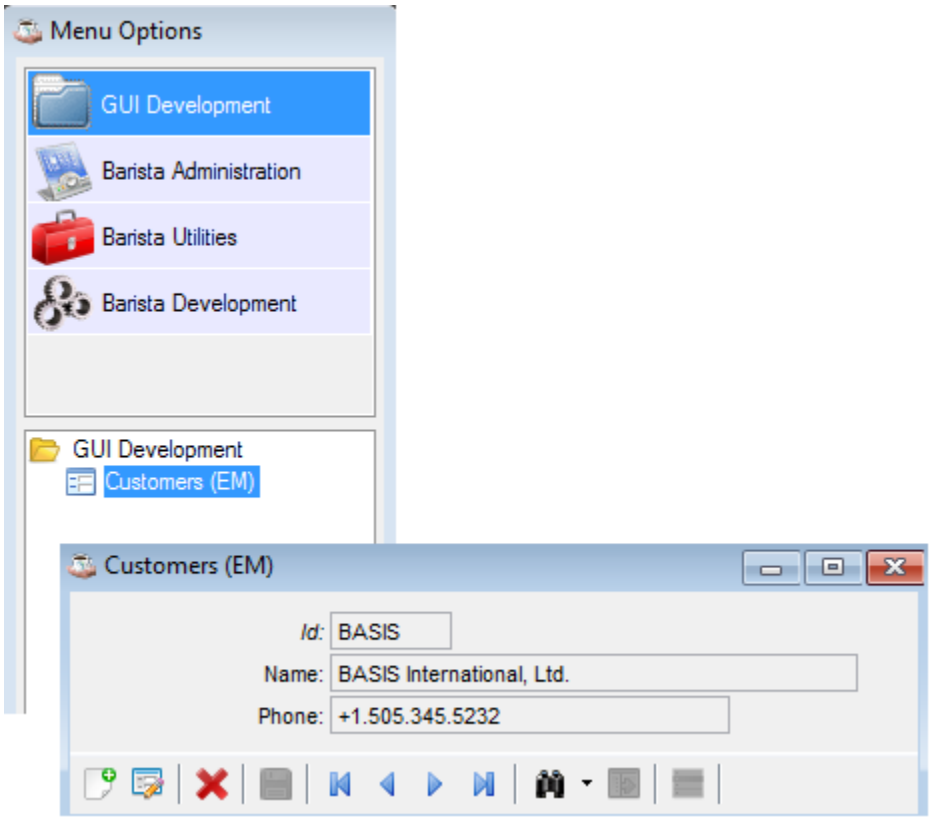
Step 1: Create a BASIS DD definition for the table by entering the individual column properties, or supply a string template to quickly create all of the columns.



Step 2: Run the Import to Barista Dictionary utility to create the Barista element and table definitions from the import source.



Step 3: The Import to Barista Dictionary wizard automatically creates the forms along with a Barista menu.

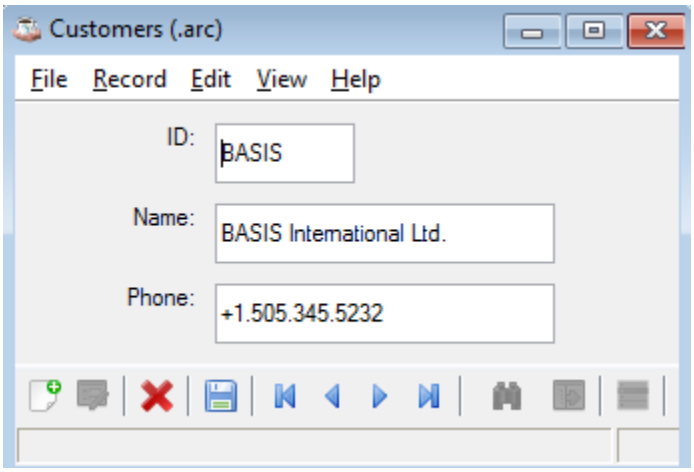


Plumb into Barista

Back to top

You may find that while you can use Barista for most of your application forms, you have a few forms that have a level of complexity or customization that go beyond the standard Barista model. This is a perfect use case for employing one of the other BASIS development tools to create your form, then launch it from the Barista menu. Optionally, you can adopt a hybrid approach and alter a form that was initially developed outside of the framework, replacing the navigation and record I/O buttons with those used by Barista. This preserves much of the complexity or custom nature of the form, while giving it a Barista look and feel.

Here we see how the original form, developed with AppBuilder, looks when we replace the original buttons with Barista's navigation bar.



Additional Barista resources:

- Barista Getting Started
- Create and Synchronize Applications
- BASIS DD to Barista App in a Flash
- Barista Plumbing Exposed

You can find the original version of this article on the BASIS FAQs page.

More Information

Please post any additional questions or comments to bbj-developer@basis.cloud. To join that discussion forum, subscribe online. For information on using BASIS' community forums, including the bbj-developer list, please see the BASIS Community Forum User's Guide.

Sample Programs

- Download the Customer Master File Maintenance code samples referenced in this article