Input Validation: Veto Power

By Jim Douglas

This article references sample programs available for download at:
www.basis.com/advantage/mag-v9n1/validation.html

Input validation seems like the simplest thing in the world: A user enters a value into a field and the program decides whether to accept it. Character-mode programs have been handling it like this for decades:

```
0010 INPUT "Print to (S)creen, (P)rinter, or (F)ile? ",OUTPUT$
0020 IF OUTPUT$<>"S" AND OUTPUT$<>"P" AND OUTPUT$<>"F" THEN GOTO 0010
```

The INPUTE verb improves on that, allowing masking and using input validation instead of a secondary test:

```
0010 PRINT 'CS',"Print to (S)creen, (P)rinter, or (F)ile? ",
0020 INPUTE (0,ERR=0020)41,0,"Z","",OUTPUT$:("S"=0030,"P"=0030,"F"=0030)
```

In either case, the concept is the same: The user cannot proceed until he or she enters a legal value.

How does this translate to an event-driven GUI program?

GUI Validation in BBj

The intuitive approach is to validate the input when the user tries to leave the field, forcing valid input before continuing. Here is an example:

```
rem ' Validation Sample

open (unt)"X0"
sysgui! = bbjapi().getSysGui()
text$ = "Validation"
window!= sysgui!.addWindow(100,100,230,170,text$,$00010083$)
window!.setCallback(sysgui!.ON_CLOSE,"Close")

rem ' Output to Screen, Printer or File?
text$ = "Output to Screen, Printer, or File?"
window!.addStaticText(100,10,20,120,25,text$,$8000$)
output!=window!.addInputE(101,140,20,20,25,$000E$,"Z")
output!.setCallback(sysgui!.ON_LOST_FOCUS,"OutputValidation")
output!.focus()

rem ' Focus is allowed here when the first field is valid
text!=window!.addEditBox(102,140,60,60,25,"")

process_events

rem ' Output Validation

OutputValidation:
output$=output!.getText()
if output$<>"S" and output$<>"P" and output$<>"F" then
   output!.focus()
   sysgui!.flushEvents()
endif
return

Close:
release
```

Though this example contains more code than the equivalent character-mode program, it is comparable. The user must enter “S,” “P,” or “F” in the first field before proceeding to the second field. What could be wrong with this approach? As we add more fields to the screen, each with its own validation rules, the problems become clear.

Validation in BBj 4.03

The example `OldValidation.txt` creates the dialog box in **Figure 1** that contains the four controls in **Figure 2**:

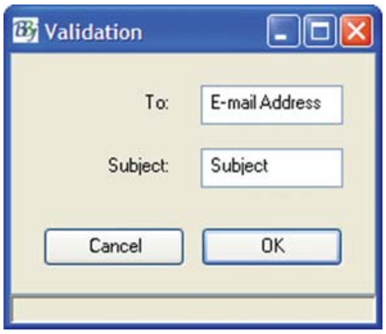


Figure 1. `OldValidation.txt` results.

CONTROL	COMMENTS
“To” field	Requires an “@” character for the e-mail address to be valid
“Subject” field	Accepts any non-blank input as valid
[OK] button	Validates the entire screen, then does something with the data
[Cancel] button	Leaves this window and bypasses validation, even if a control contains invalid data

Figure 2. `OldValidation.txt` controls.

Jim Douglas
Software Engineer Contractor

How well does this work?

After clicking in the "E-mail Address" field, then clicking in or tabbing to the "Subject" field, the address field loses focus and the subject field gains focus. This triggers the ON_FOCUS_LOST event handler, AddrValidation, for the e-mail address field. The e-mail address, lacking the required @ symbol, is invalid, so the program forces focus back to the e-mail address field. The user cannot proceed until this field contains an "@." So far, so good.

Clicking the [OK] button validates the e-mail address field, then the subject field. If either field is invalid, BBj® forces focus back to the invalid field for correction. Then the user can proceed.

Clicking the [Cancel] button also triggers an ON_FOCUS_LOST event for the control that had focus. This is potentially a problem because clicking [Cancel] should skip all validation and let the user out. The program allows for this scenario by including this special rule in each of the validation routines: If the [Cancel] button has focus, consider the control to be valid. This solves the problem, but at the cost of complicating the validation routines.

That all sounds great, but there is a serious problem with using the loss of focus event. To see the problem, run this program, click in the "Subject" field, clear its contents, then click in the "E-mail Address" field. This is what happens:

1. The "Subject" field loses focus.

2. The loss of focus in the "Subject" field triggers the validation logic for the "Subject" field.

3. The subject is invalid so BBj forces focus back to the "Subject" field.

4. Since focus had already moved to the "E-mail Address" field, it now loses focus.

5. This loss of focus triggers the validation logic for the "E-mail Address" field.

6. The e-mail address is invalid, so BBj forces focus back to the "E-mail Address" field.

7. And so on…

The program is now in an infinite loop, jumping back and forth between the two fields.

Validation in BBj 5.0: Veto Power

In previous versions of BBj, field-level validation was very difficult because it was impossible to stop the user from moving out of a field. Forcing the user back to an invalid field after some other control gained focus caused a potential cascade of side effects. That cascade could include many different events and state changes, for example by clicking a radio button or check box. Changes like this were extremely difficult to reverse.

In BBj 5.0, loss of focus is a *vetoable* event, meaning the user can only move out of a control if it contains valid data based upon whatever rules are defined in the application. Giving the developer a way to stop the user from leaving the control until it is valid eliminates the potential cascade of side effects. For a complete example, see **`NewValidation.txt`**.

Compare these two approaches to field-level validation shown in **Figure 3**.

BBj 4.03 reacts to the loss of focus. If it determines that the e-mail address is invalid, it forces focus back to the address field.

BBj 5.0 reacts to the *attempted* loss of focus. If the data is invalid, the developer vetoes the attempted loss of focus by calling **`addr!.accept(0)`**, which flushes the event queue and keeps the user in the e-mail address field. If the data is valid, the developer approves the loss of focus by calling **`addr!.accept(1)`**. While the program is in the validation subroutine, BBj locks the window. The window stays locked until the application calls **`control!.accept`***`(boolean)`*.

Making the loss of focus vetoable eliminates all of the side effects caused by using the loss of focus event.

Form Validation

Previous versions of BBj handled form validation in the event handler for the [OK] button. BBj 5.0 adds a new form validation event as shown in the example below:

```
rem ' OK Button causes form validation

OK!=window!.addButton(1,380,300,90,25,"OK")
OK!.setCallback(sysgui!.ON_BUTTON_PUSH,"OK")
OK!.setCallback(sysgui!.ON_FORM_VALIDATION,"FormValidation")
...
FormValidation:
control!=sysgui!.getLastEvent().getControl()
rem ' do whatever is necessary here to validate the form contents
valid = 1;
control!.accept(valid)
return
...
OK:
rem ' If the program gets here, form validation was successful
return
```

continued...

BBj 4.03

```
addr!.setCallback(sysgui!.ON_LOST_FOCUS,"AddrValidation")
...
AddrValidation:
valid=window!.getFocusedControlID()=2 or pos("@"=addr!.getText())
if valid
   statbar!.setText("")
   addr!.setBackColor(white!)
else
   statbar!.setText("E-mail address must contain '@'")
   addr!.setBackColor(red!)
   addr!.focus()
   sysgui!.flushEvents()
endif
return
```

BBj 5.0

```
addr!.setCallback(sysgui!.ON_CONTROL_VALIDATION,"AddrValidation")
...
AddrValidation:
valid=pos("@"=addr!.getValidationText())
if valid
   statbar!.setText("")
   addr!.setBackColor(white!)
else
   statbar!.setText("E-mail address must contain '@'")
   addr!.setBackColor(red!)
endif
addr!.accept(valid)
return
```

Figure 3. Field level validation comparison.

There are two benefits to this approach:

1. While the FormValidation subroutine is validating the form, the window is locked and the user is prevented from making any changes until the call to `control!.accept(boolean)` accepts or rejects the data. This eliminates the infinite loop problem caused by forcing focus back and forth between two invalid controls, and it removes the need to reverse state changes caused by clicking in controls like check boxes or radio buttons.

2. The program is more modular by breaking validation into two new separate subroutines. The FormValidation subroutine is responsible for validation while the OK subroutine is responsible for everything else (e.g. updating the data to a file). BBj invokes the OK subroutine only after the FormValidation subroutine accepts the form.

It Works With GUIBuilder

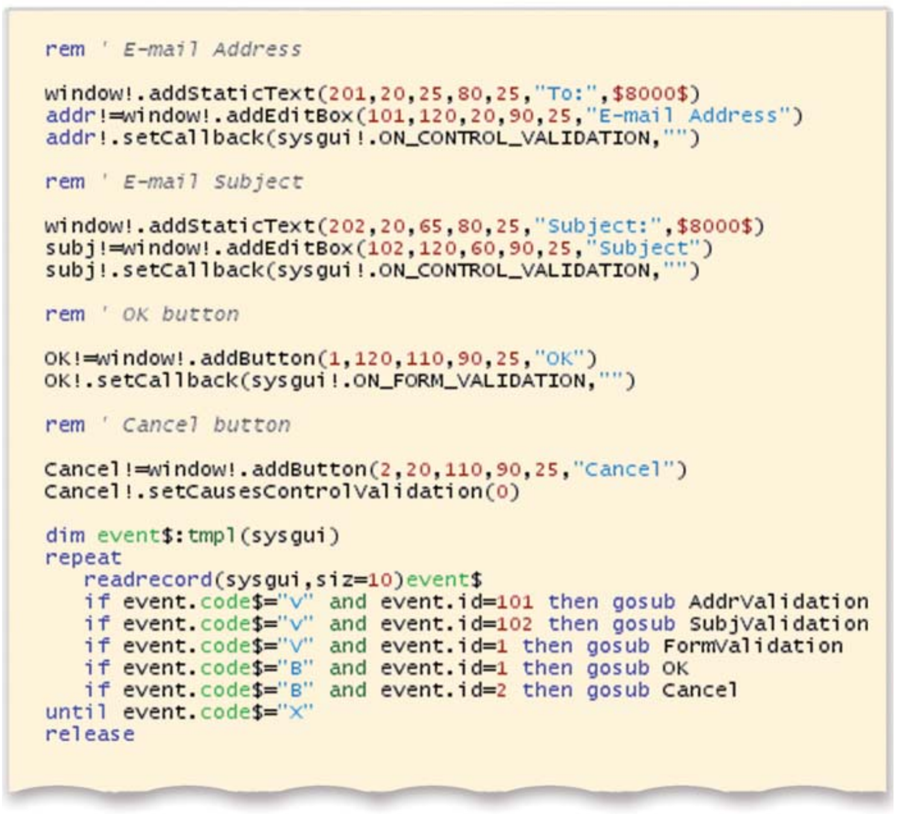
When adding form and field validation into programs originally developed in Visual PRO/5® or GUIBuilder®, it is not necessary to rewrite existing programs to use callbacks and PROCESS_EVENTS. The sample **`NewValidation2.txt`** processes the same form using an event loop instead of callbacks. The excerpt from that sample (see **Figure 4** on the next page) shows that event code "v" triggers field-level validation and event code "V" triggers form-level validation.

To incorporate validation into a GUIBuilder-generated program, add the various callbacks for ON_FORM_VALIDATION and ON_CONTROL_VALIDATION to the initialization routine, specifying any string for the callback subroutine (here we just used **`""`**) and create the corresponding Form Validation and Control Validation event handlers for those controls.

But Wait, There's More!

While the most important feature of BBj 5.0 control validation is the ability to veto loss of focus, there are other significant improvements that are also very useful.

continued...



```
rem ' E-mail Address

window!.addStaticText(201,20,25,80,25,"To:",$8000$)
addr!=window!.addEditBox(101,120,20,90,25,"E-mail Address")
addr!.setCallback(sysgui!.ON_CONTROL_VALIDATION,"")

rem ' E-mail Subject

window!.addStaticText(202,20,65,80,25,"Subject:",$8000$)
subj!=window!.addEditBox(102,120,60,90,25,"Subject")
subj!.setCallback(sysgui!.ON_CONTROL_VALIDATION,"")

rem ' OK button

OK!=window!.addButton(1,120,110,90,25,"OK")
OK!.setCallback(sysgui!.ON_FORM_VALIDATION,"")

rem ' Cancel button

Cancel!=window!.addButton(2,20,110,90,25,"Cancel")
Cancel!.setCausesControlValidation(0)

dim event$:tmpl(sysgui)
repeat
   readrecord(sysgui,siz=10)event$
   if event.code$="V" and event.id=101 then gosub AddrValidation
   if event.code$="V" and event.id=102 then gosub SubjValidation
   if event.code$="V" and event.id=1 then gosub FormValidation
   if event.code$="B" and event.id=1 then gosub OK
   if event.code$="B" and event.id=2 then gosub Cancel
until event.code$="X"
release
```

Figure 4. Event loop code excerpt from **`NewValidation2.txt`**.

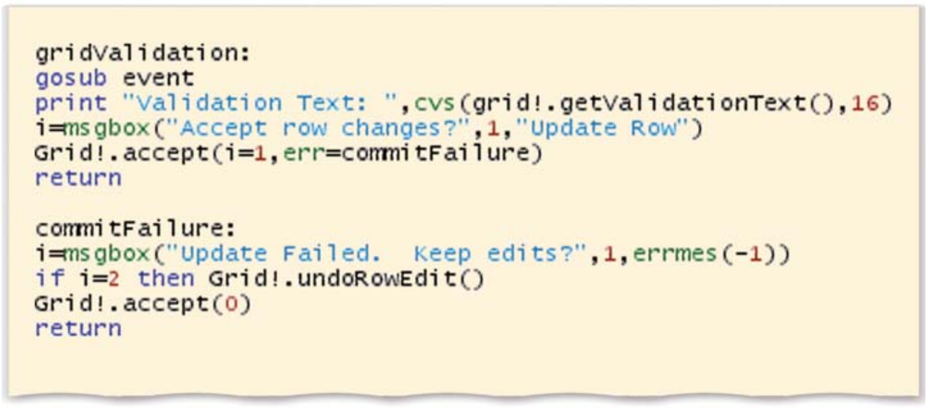
- **`Cancel!.setCausesControlValidation(0)`** specifies that a control such as the [Cancel] button does not trigger validation. This simplifies the validation routines since they do not have to explicitly check for the [Cancel] button.

- **`Control!.getValidationText()`** retrieves the current text from the control without incurring additional network traffic. This improves performance in a distributed environment, but field-level validation, by its nature, creates extra network traffic. In a high-latency-distributed environment, applications are significantly more responsive when performing validation at the form level, by clicking [OK]. Though this approach differs from legacy character-mode applications that usually validate fields immediately, it is familiar to users who run Web applications. Normally, in Web-based applications, the user fills in a complete form, then clicks [Submit] to validate the entire form. BBj offers both kinds of validation so you can choose the best method for your environment.

- The new data-aware grid event, ON_GRID_ROW_VALIDATION, validates grid row data changesbefore they are updated to the database. This event works like the control and form validationevents: The developer accepts the update by calling **`Grid!.accept(1)`** or rejects the update by calling **`Grid!.accept(0)`**.

continued...

The code fragment in **Figure 5** (from the **`GridValidation.txt`** sample) demonstrates how to use grid row validation.



```
gridValidation:
gosub event
print "Validation Text: ",cvs(grid!.getValidationText(),16)
i=msgbox("Accept row changes?",1,"Update Row")
Grid!.accept(i=1,err=commitFailure)
return

commitFailure:
i=msgbox("Update Failed.  Keep edits?",1,errmes(-1))
if i=2 then Grid!.undoRowEdit()
Grid!.accept(0)
return
```

Figure 5. Grid row validation code excerpt from **`GridValidation.txt.`**

Summary

Field and form validation is a valuable enhancement to BBj. As this article shows, it is easy to incorporate validation into new applications or retrofit it into existing applications whether originally developed using Visual PRO/5, GUIBuilder, or previous versions of BBj. This enhancement is now available in BBj 5.0.

Download the samples referenced in this article at: www.basis.com/advantage/mag-v9n1/validation.html

BBj 5.0 documentation is available online at www.basis.com/devtools/documentation/index.html.
Go to "Validation" in the index.