Title: Comprehensive Approach to Field Level Validation/Security in the Event Driven UI

Author: Hansdip Singh Bindra

The Event Driven UI Model presents a robust and dynamic interface for developing Business Applications. This model allows the developer to provide a customizable – "User Friendly Interface" that allows the user to enter valid information (data) into the underlying Database. The old, and often not used computer phrase – "GIGO – Garbage in, Garbage Out" is more applicable to the Event Driven UI, than other environments, as the flexibility provided by this UI does provide more challenges in ensuring proper validated information is sent from the UI to the Database.

The Event Driven UI Model is based on Widgets (Fields) – Buttons, Fill-Ins, Combo-Boxes, etc., that reside in a Window (in Progress – a Field Group, which in turn resides in a Frame, whose parent could be a Window). Thus, the lowest and most comprehensive level of information validation can only be done at the Field Level. Validation encompasses, both, the actual checking of the information, and also the Security associated with the Field. If the user is not allowed to work with certain Fields on the Window, then those Fields should not be enabled for that user, thereby, reducing the possibility of invalid information from entering into the Database.

This article is based on Progress Version 9, and higher, on the GUI Windows Platform, but enterprising developers are more than welcome to make the changes to make it work in earlier Versions of Progress, or on other Event Driven UI Platforms.

The code supplied with this article is extremely easy to read, as I have made the Widget and Procedure names self-explanatory and provided comments and instructions in the programs.

The Field Level Validation/Security Approach presented in the article is based on the use of SUPER-PROCEDURES (Internal Procedures in them) and associating them with the SESSION Handle, Persistent Triggers, "Walking the Widget-Tree", PRIVATE-DATA of Widgets, PROGRAM-NAME(), Text Validation Files (.Val), and Temp-Tables.

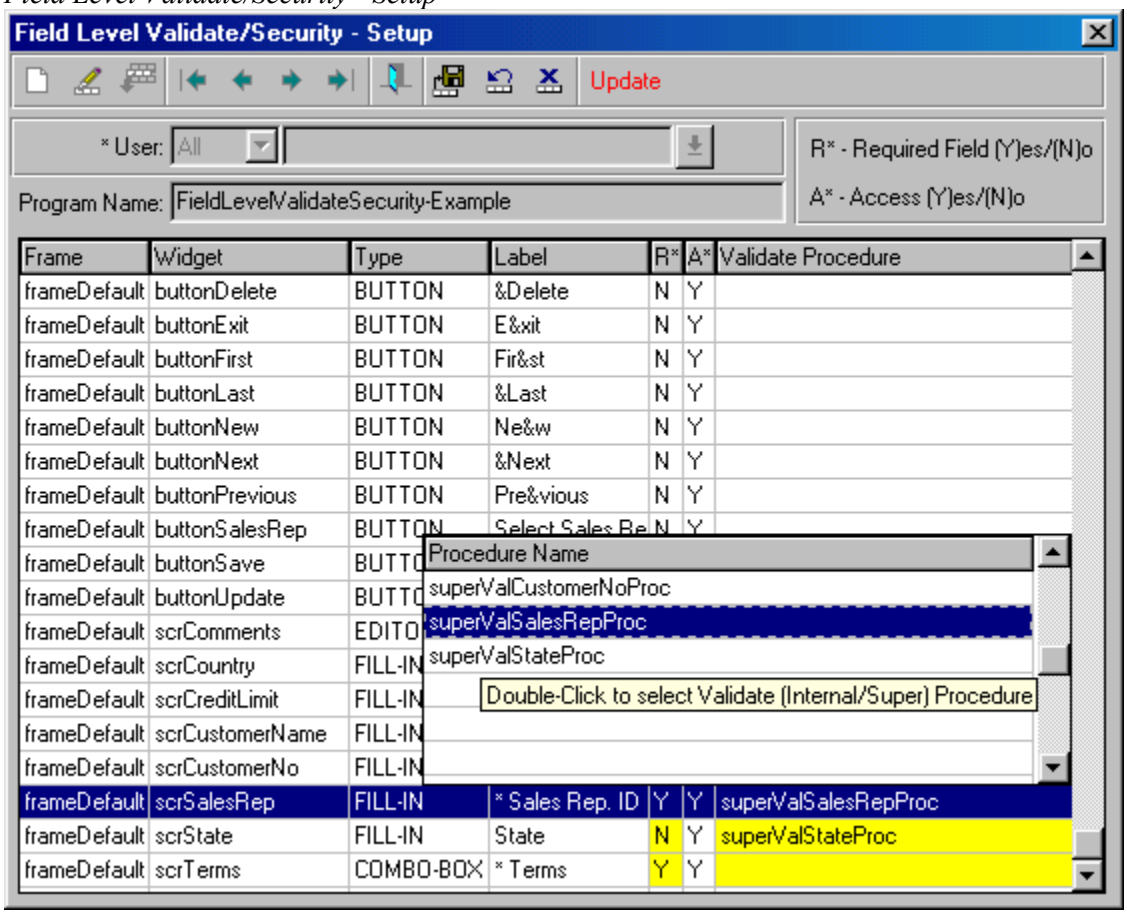
This approach can be divided into two sections, the Setup/Infrastructure of Field Level Validation/Security and the actual Implementation/Usage of it. I will explain these with the use of an example – Maintaining Customer Information. Lets say we need to maintain a few Fields in the Customer Table – Customer Number, Name, Sales Rep., Country, State, Credit Limit and Comments. We have also determined that Customer Number, Sales Rep., Country and Terms are Required Fields (have to be filled in). In addition, we need to validate (information has to conform to allowed values) the Sales Rep., Country, State, and Credit Limit Fields.

Based on this example, we will develop a dynamic Field Level Validation/Security mechanism for the Customer Maintenance. Not only will this mechanism allow us to validate the information entered on the UI, but furthermore, allow us to validate it based on the User, or the User's Group properties.

SETUP AND INFRASTRUCTURE

The Setup and Infrastructure consists of activating the Editing of Field Level Validate/Security for the SESSION (usually based on some User rights, i.e. Administrator), by launching a Persistent Trigger for the SESSION. This Trigger, once activated will execute when the developer presses the ATL-CTRL-V keys, or Right-Mouse Double-Clicks on the appropriate Window/Dialog-Box UI. The resulting effect of this executing is the Field Level Validation/Security – Setup Dialog-Box, which displays the Program Name of the source Window/Dialog-Box, and all the low level usable Widgets (i.e., Fill-In, Combo-Box, Browse, Button, Radio-Set, Selection-List, Editor, Toggle-Box and Slider) in it. If the source Program has multiple Frames, all its Frames are traversed to get the appropriate Widget List.

Field Level Validate/Security - Setup



On this window the developer can setup the entire Validation and Security mechanism for all the displayed Widgets. Validation is only allowed for enterable Fields, such as Fill-Ins, Combo-Boxes and Editors. These three Widget types can be set as a Required Field (Yes/No), and can have a Validation (Super – Internal) Procedure associated with it. This Validation Procedure must reside in a SUPER-PROCEDURE that is attached to the User's SESSION, and take in one Input Parameter – the value of the Widget, and return one Output Parameter – a logical (Yes/No) indicating the validity of the value. A Field does not have to be Required Field in order to have a Validation Procedure associated with it. This allows us to validate Fields that are not required to be filled in, but if they are, then validate the filled in value (e.g. in Customer Maintenance, the State Field). The Validation Procedure that resides in a Session Super Procedure is launched

from the Main Super Procedure for Field Level Validate/Security (which will be discussed in the Implementation/Usage section).

The Security mechanism is setup using the Access (Yes/No) associated with each of these Widgets. This is where the User (All, Group, or ID) Field on the screen comes into play. The developer has the flexibility to setup the Access (Yes/No) and if so desired, the Validation configuration, for All Users (referred to as Common), a User Group, or even an individual User ID. All this is done without writing any additional code (apart from three calls to Super Internal Procedures) in the source Window/Dialog-Box.

Once the developer has determined the exact configuration, the information on the Setup screen is written out to an appropriate .Val Text File. The .Val File will reside in a "\ValidateCommon" Folder for All Users, or in a "\ValidateCommon\User<first letter of Group or ID> Sub-Folder, if setup for a particular User Group or ID. The Common .Val File Name is simply the name of the source Program, with the .w, .p, or .r extension replaced with .Val extension. The File Name for User Group/ID based .Val File is simply the Common File Name with the appropriate prefix ("G-" + <User Group> or "I-" + <User ID>). This approach eliminates the need to change the Database associated with the application, and also the several reads from the Database for each Field on the source Window (and all its Frames). Furthermore, the "\ValidateCommon" Folder should be setup as Read-Only and/or Hidden for Production systems, thereby adding another layer of security to the application.

The .Val File has the following information for each Widget EXPORTED to it:

.Val File Layout

Frame Name	Widget Name	Widget Type	Access	Required Field	Validate Procedure
frameDefault	scrCountry	FILL-IN	YES	YES	superValCountryProc

This information is IMPORTED into a Temp-Table in the Implementation/Usage section of the entire Validation and Security process. Once the developer has saved the .Val File(s), exit from the Setup Dialog-Box, then exit from the source Program (the one for which we just set up the Validate/Security information), and then launch the source Program again, the source Program is loaded with the Validate/Security information.

Of course, the same information is also IMPORTED into the Temp-Table in this Setup Dialog-Box so that the developer can edit this information using the Browse Widget that displays the entire Field Level Widget List for the source Program. The imported information in the Temp-Table is synchronized with the actual Widgets that are obtained from "Walking the Widget-Tree", as follows:

TempWidgetGroupProc – Internal Procedure in FieldLevelValidateSecurity-Setup.w

```
PROCEDURE TempWidgetGroupProc :
/*------------------------------------------------------------------------------
  Purpose:   Walk through Widget-Tree - Field Group and synchronize with Temp-Table
  Parameters:  <none>
  Notes:
------------------------------------------------------------------------------*/
  DEFINE INPUT PARAMETER inFrameHandle AS WIDGET-HANDLE NO-UNDO.

  DEF VAR localGroupHandle AS WIDGET-HANDLE NO-UNDO.
  DEF VAR localWidgetHandle AS WIDGET-HANDLE NO-UNDO.

  /* First Field Group in Frame */
  ASSIGN localGroupHandle = inFrameHandle:FIRST-CHILD.
```

```
  /* Field Group - contains Widgets */
  DO WHILE (localGroupHandle <> ?):

     /* First Widget in Field Group */
     ASSIGN localWidgetHandle = localGroupHandle:FIRST-CHILD.

     /* Widgets */
     DO WHILE (localWidgetHandle <> ?):

        /* Widget is FRAME - get Frame's Widgets */
        IF localWidgetHandle:TYPE = "FRAME":U THEN
           DO:
              RUN TempWidgetGroupProc(INPUT localWidgetHandle).
           END.
        ELSE
           DO:
              /* Synchronize Widget with Import Temp-Table */
              IF LOOKUP(localWidgetHandle:TYPE,localListType) > 0 THEN
                 DO TRANSACTION:

                    FIND FIRST bufferTempWidget WHERE
                         bufferTempWidget.FrameName = inFrameHandle:NAME AND
                         bufferTempWidget.WidgetName = localWidgetHandle:NAME
                         EXCLUSIVE-LOCK NO-ERROR.

                    IF NOT AVAILABLE bufferTempWidget THEN
                       DO:
                          CREATE bufferTempWidget.

                          ASSIGN bufferTempWidget.FrameName = inFrameHandle:NAME
                                 bufferTempWidget.WidgetName = localWidgetHandle:NAME.
                       END.

                    ASSIGN bufferTempWidget.Complete = YES
                           bufferTempWidget.WidgetType = localWidgetHandle:TYPE.

                    IF CAN-QUERY(localWidgetHandle,"LABEL":U) THEN
                       DO:
                          IF localWidgetHandle:LABEL <> ? THEN
                             ASSIGN bufferTempWidget.WidgetLabel =
                                                    localWidgetHandle:LABEL.
                       END.

                    RELEASE bufferTempWidget.
                 END. /* TRANSACTION */
           END.

        /* Next Widget */
        ASSIGN localWidgetHandle = localWidgetHandle:NEXT-SIBLING.
     END.

     /* Next Field Group */
     ASSIGN localGroupHandle = localGroupHandle:NEXT-SIBLING.
  END.
END PROCEDURE.
```

The Setup and Infrastructure process consists of the following Programs:

Setup and Infrastructure Programs – Description

Program Name	**Description**
FieldLevelValidateSecurity-Activate.p	Activates the Edit – Setup/Infrastructure process by Persistent Running the next Program using the ON ANYWHERE UI Trigger. Normally run though a Menu Item on Main Menu/Window for entire SESSION if the User is the administrator and needs to Edit Field Level Validation/Security. Regular Users will not get this Menu Item thereby not allowing

	them to Edit Field Level Validation/Security.
FieldLevelValidateSecurity-Load.p	Once activated, this Program runs for the entire Session. Its purpose is to get the First Frame of the Window where it is launched from using ALT-CTRL-V or Right-Mouse Double-Click, and the name of that Program. The First Frame and Program Name is then input into the next Program. If the Program Name cannot be determined because it is run from the AppBuilder, the developer is prompted for the name through FieldLevelValidateSecurity-FileName.w.
FieldLevelValidateSecurity-Setup.w	The Dialog-Box that was described earlier where the developer assigns the Field Level Validate and Security information for the source Window or Dialog-Box.

IMPLEMENTATION AND USAGE

This Implementation and Usage of Field Level Validation/Security information setup in the previous section involves the IMPORT of that data from the appropriate .Val File, and assigning that to the PRIVATE-DATA of each Widget (Field). In addition, depending on the Access and Required values of the Widget, the appearance (e.g. Label and Label Color, Enable/Disable, etc.) of the Widget is determined.

The entire Implementation and Usage mechanism is controlled by the SESSION SUPER-PROCEDURE named FieldLevelValidateSuper-Main.p. This is indeed the "Brains" behind the Field Level Validation and Security mechanism described in the article. It determines the First Frame Handle for the source Program, Launches all the Session Super Procedures that contain the Validation Procedures (described in the previous section), Maintain the list of all Session Validation Procedures in a Temp-Table – TempProcedureName, for the Setup and Infrastructure section. Moreover, for this section of the Field Level Validate/Security process, it determines the First Frame Handle, Launches and Maintains the list of all Session Validation Procedures, Loads the .Val File information for the current source Window/Dialog-Box into the Temp-Table – TempWidget, "Walks the Widget-Tree" assigning the PRIVATE-DATA and setting the display properties using the Validation/Security data to each Widget, and finally "Walks the Widget-Tree" to Validate the Fields.

Using the Customer Maintenance example, we make four basic additions to the Program, one in the Definitions section, two in the Main Block, and one in the ValidateFieldsProc Internal Procedure. These four additions fully implement the Field Level Validate/Security mechanism into this source Window. The code for these four additions is presented using Include Files thereby providing a "Black-Box" approach. All the developer has to do is place the Include Files in the appropriate places in the source Program. Each Include File simply contains a call to an Internal Session Super Procedure residing in the "Brain".

Include File/Internal Super Procedure – Description

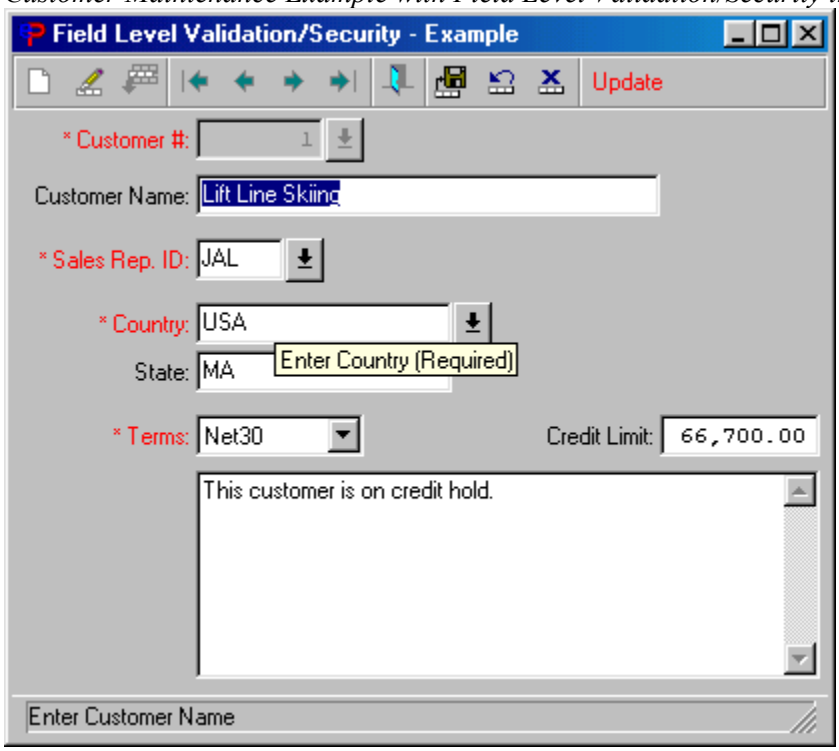
Include File/ Internal Super Procedure Name	**Description**
SuperProcedure<UIB>-Definitions RUN FieldLevelValidateSuper-Main.p	Normally included once in the Session Launch Application, like Main Menu, or System Login. However, include in each Window/Dialog-Box to Run and Associate the "Brain" with the SESSION, as a SUPER-PROCEDURE.
SuperProcedure-MainBlock.i RUN **superFieldLevelLoadProc**	Requires the Program Name as Input Parameter in order to IMPORT the Validate/Security information from the appropriate .Val File into the TempWidget Temp-Table for

	use in the next Internal Procedure. It must be executed before enable_UI, so that the Disk I/O is done before Window is Viewed (Realized) so that Window does not appear to hang after being displayed on the screen.
SuperProcedure-Enable_UI.i RUN **superFieldLevelAssignProc**	Requires the CURRENT-WINDOW Handle as Input Parameter. It must be executed after enable_UI, so that the Window is Viewed (Realized), in order to get the proper Window Handle and all the relationships between its Widgets. It "Walks the Widget-Tree" and assigns the Validate/Security information from Temp-Table to the Widgets.
SuperProcedure-ValidateSave.i RUN **superFieldLevelValidateProc**	Usually placed in the code associated with Save Button – ValidateFieldsProc. Validate the Fields using the information in their PRIVATE-DATA by "Walking the Widget-Tree". INPUT CURRENT-WINDOW, INPUT <Display Error Messages – Yes/No>, INPUT <Include Hidden Frames in Validation – Yes/No>, INPUT <Combined Error Message – Yes/No>, OUTPUT <Result of Validation Process – Yes/No>

The end result of these Include Files for the Customer Maintenance example is that no additional code is required to ensure that the Required Fields – Customer Number, Sales Rep., Country and Terms are filled in and valid when the User clicks on the Save Button. Also, no code is needed in the source Program to validate the Sales Rep., Country, State, and Credit Limit Fields. Finally, the Required Fields have an (*) –Asterisk in the Label, are Red in Color, and have the word – "(Required)" in its Tooltip and Help. These UI enhancements to differentiate the Required Fields from the other Fields are done by the "Brain" and thus eliminating the need for the developer to enhance the UI while developing the Window/Dialog-Box.

In fact the UI enhancements themselves are customizable. This example simply uses a fixed set of values for using the Asterisk in the Label, the Color of the Label and modifying the Tooltip/Help, in the "Brain" Super Procedure. Normally, one would set these using a Menu Control Table, where the application would allow choices for Label Color (e.g. 12-Red, ? – Default, 14-Yellow, and 15-White), whether or not to put the Asterisk in the Label (Yes/No), and whether to indicate (Yes/No) a Field is Required by putting the word "(Required)" in the Tooltip/Help. The following Window shows the enhanced Customer Maintenance example (compare it with the original Program when opened in AppBuilder – you will notice that the Asterisks are missing, the Label Color for all Fields are same, and the Tooltip/Help for Required Fields like Country does not have the "(Required)" word in it.

Customer Maintenance Example with Field Level Validation/Security implemented



If we had set the Security Access to NO for the New and Delete Buttons, these Buttons would not have been enabled for use. While designing the source Window/Dialog-Box set the SENSITIVE attribute of all appropriate Widgets to YES, so that the default is full Access. Then using the Setup Dialog-Box, as explained earlier in the Setup and Infrastructure section, take Access away from the appropriate Fields. If the application requires the re-enabling/disabling of Fields, use the Access information in the PRIVATE-DATA to ensure that these Fields are never enabled for Users or User Groups who are not allowed to use these Fields.

The PRIVATE-DATA for each Field Level Validate/Security Widget is as follows:

PRIVATE-DATA Layout

Access	**Required Field**	**Validate Procedure**
Y or N	Y or N	<superVal…Proc>

This information actually is a comma separated list, so that ENTRY(1) is Access, while ENTRY(2) is Required Field, and ENTRY(3) is the name of the Widget Validation Super Internal Procedure. This Validation Procedure resides in a Super Procedure that is launched from the Main Block of the "Brain". In our Customer Maintenance example, we launch FieldLevelValidateSuper-Customer.p. This Program contains validation Internal Procedures for Customer Number, Country, State, Credit Limit, and other such Fields associated with Customer. The design philosophy behind this approach makes extensive use of the encapsulated approach provided by SUPER-PROCEDURES. Basically, the developers of an Application should create such Validation Procedures for all appropriate Fields in it, and place it in a "Central Bank". Now, this "Central Bank" is opened for use at the beginning of a User SESSION, and wherever one needs to validate such a Field, it is done using a call to a "superVal" Internal Procedure in this repository of Validation Procedures. The "Brain" knows of these "superVal" Procedures using

the INTERNAL-ENTRIES attribute of the SUPER-PROCEDURE, and then saving the Procedure Names in the TempProcedureName Temp-Table.

The "Brain" – FieldLevelValidateSuper-Main.p, contains Internal Procedures accessible to other Programs, the so called Super Procedures, like superFieldLevelLoadProc, superFieldLevelAssignProc, and superFieldLevelValidateProc.

Super Procedures in the "Brain" – Description

Super Internal Procedure Name	**Description**
superGetFirstFrameHandleProc	This procedure determines the First Frame of a given Window. The First Frame is needed as it is a starting point for "Walking" the entire "Widget-Tree".
superGetTempProcedureNameProc	Outputs the list of All "superVal" Procedures stored in TempProcedureName Temp-Table to the calling Program. Used by the Setup Dialog-Box to display Drop-Down Browse for Validate Procedures.
SuperValProcedureNameProc	Validation Procedure – "superVal" for the actual "superVal" Procedures. Determines if a "superVal" is valid by checking if an entry exists in the TempProcedureName Temp-Table.
SuperFieldLevelLoadProc	This procedure loads the Validation and Security data from the appropriate .Val File into the TempWidget Temp-Table. First, it determines the actual .Val File Name, based on User ID, User Group, or Common. It then empties the Temp-Table, then IMPORTS the information from the .Val File into the Temp-Table.
SuperFieldLevelAssignProc	Associates the TempWidget Temp-Table information with the Widgets in the source Program. First, it determines if a .Val File was setup for the source Window/Dialog-Box. If a File was created and loaded into the Temp-Table, then it "Walks the Widget-Tree". During this traversal of the "Widget-Tree", it sets the appropriate information in the PRIVATE-DATA of the Widget, sets the UI differentiator for a Required Field, and the sensitivity of the Widget based on whether Access is allowed.
SuperFieldLevelValidateProc	The Validation for all Widgets in the source Program is done by this Procedure. The developer can set two types of Error Messaging: Combined or Individual. The Individual method (normally used in Client/Server UI) simply displays the Error associated with one Field and places "Entry" into that Field, whereas the Combined Message (normally used in Web UI) displays a large Error message that shows the errors associated with all invalid Fields. The developer can also set whether to validate All Frames in a Window, or only non-Hidden Frames. This is useful in a multi-frame source Program when the developer would like to only validate a given displayed Frame. First, if Combined Messaging is turned on, the Procedure initializes the Combined Message. Next it determines the First Frame, and then "Walks the Widget-Tree" to validate the Fields. Field Level Validation is done at two levels – Required Fields are examined to determine whether they have been filled in, whereas Fields with "superVal" Procedures associated with them are validated using these Procedures.

	Next, if invalid Fields exist the appropriate Error Message is displayed depending on the chosen Combined or Individual messaging method. Finally, the Procedure returns the result (Yes/No) of the Validation process. If all Fields are valid, the result is Yes.

In addition, the "Brain" contains several Private Internal Procedures, where most of the "action" takes place. Since this "action" is only useful inside the "Brain", the PROCEDURES are defined as PRIVATE, again keeping with the "Black-Box" approach to Application development. Detailed explanation of these Private Internal Procedures can be obtained from examining the code in the "Brain".

privateTempWidgetFieldValidateProc – Private Internal Procedure in FieldLevelValidateSuper-Main.p

```
PROCEDURE privateTempWidgetFieldValidateProc PRIVATE:
/*------------------------------------------------------------------------------
  Purpose:   Validate Widget
  Parameters:  <none>
  Notes:
------------------------------------------------------------------------------*/
  DEFINE INPUT PARAMETER inWidgetHandle AS WIDGET-HANDLE NO-UNDO.
  DEFINE INPUT PARAMETER inCombinedErrorMsg AS LOGICAL NO-UNDO.
  DEFINE OUTPUT PARAMETER outValidFields AS LOGICAL INITIAL YES NO-UNDO.

  DEF VAR localRequiredField AS CHARACTER NO-UNDO.
  DEF VAR localValidateProc AS CHARACTER NO-UNDO.

  /* Required Widget Type */

  IF inWidgetHandle:HIDDEN = NO AND
     CAN-QUERY(inWidgetHandle,"SCREEN-VALUE":U) THEN
     DO:
        ASSIGN localRequiredField = ENTRY(2,inWidgetHandle:PRIVATE-DATA)
               localValidateProc = ENTRY(3,inWidgetHandle:PRIVATE-DATA).

        /* Required Field */
        IF localRequiredField = "Y":U AND
           (TRIM(inWidgetHandle:SCREEN-VALUE) = "" OR
            inWidgetHandle:SCREEN-VALUE = ?) THEN
           ASSIGN outValidFields = ?.

        /* Validate (Internal/Super) Procedure */
        IF outValidFields = YES AND
           TRIM(localValidateProc) <> "" AND
           TRIM(localValidateProc) <> "<None>":U THEN
           DO:
              RUN VALUE(localValidateProc) (INPUT inWidgetHandle:SCREEN-VALUE,
                                            OUTPUT outValidFields) NO-ERROR.
           END.

        IF outValidFields <> YES THEN
           DO:
              RUN privateSetCurrentErrorFieldsProc(INPUT inWidgetHandle:FRAME,
                                                   INPUT inWidgetHandle,
                                                   INPUT outValidFields,
                                                   INPUT inCombinedErrorMsg).

              /* Combined Error Message - Continue */
              IF inCombinedErrorMsg = YES THEN
                 ASSIGN outValidFields = YES.
           END.
     END.
END PROCEDURE.
```

The detailed workings of this Comprehensive Approach to Field Level Validate/Security in the Event Driven UI, described in this article can be further expanded to the Web UI, and can be made more dynamic and flexible by using database Tables to drive the UI customization for Required Fields. Furthermore, similar mechanisms could be applied to Browse Cells to provide a Cell Level Validate/Security Infrastructure.

ePROCESS, Inc.
Transforming Business Processes into Software...™

Progress/WebSpeed Development and Consulting
*** Client/Server, e-Commerce, CRM and ERP Specialists ***

Phone: + 1 (973) 714-8362
Fax: + 1 (908) 876-3849
E-Mail: **info@eprocessinc.com**
Web: **www.eprocessinc.com**

