

BoxSoft
Corporation

Super Invoice - Documentation

Copyright © 1989-2014, BoxSoft Corporation, All Rights Reserved.

Table of Contents

Foreword	0
Part I Getting Started	3
1 Introduction.....	3
2 RTFM Warning!!!.....	5
3 Installation.....	6
4 Upgrading From Earlier Versions.....	10
Part II Template Usage	11
1 Adding SuperInvoice to you Applications.....	11
2 Invoice Control Template.....	12
Invoice Control Template	12
Populating an InvoiceControl Template	12
Template Settings - ABC Chain	12
Template Settings - Clarion/Legacy Chain	26
Adding the Regular Controls for "In-Place" Entry	29
Formula Classes	30
Embed Points	30
3 Shift Buttons.....	34
4 Child Window Button.....	35
5 Child Window OK Button.....	37
Part III Appendices	38
1 Example Programs.....	38
2 Project Defines.....	39
3 Troubleshooting.....	41
4 Contacting Technical Support.....	42
5 License Agreement.....	43
Index	44

1 Getting Started

1.1 Introduction

The Super Invoice template is designed to support "in-place" data-entry on a list box. The entire transaction (the parent and all children) are held in memory until the [OK] button is pressed, which is much superior to the regular "Form with a Browse" implementation. All features have been developed with the following goals:

1. The feature must be easy to implement.
2. It must work with the regular Clarion templates.
3. All code must be visible in the template (i.e. no black-box DLLs).
4. The resulting programs should work intuitively and quickly.

Event though we build our Edit In Place (EIP) on top of Clarion's foundation, you'll discover that ours is much better. It's easier to implement, more flexible, and far more powerful. Of course, we're never finishing making improvements. If you have ideas, suggestions or comments (either positive or negative), please pass them along. They're greatly appreciated.

Although it is called an "Invoice" template, it is useful in any situation where you have a parent file and one or more child files, you want to provide in-place data entry, and you want to treat all changes as a single transaction. Another example would be a purchase order.

In those situations where you can't fit all the information in the listbox, you can use a separate "ChildWindow" to access these fields. For full-screen validation, we've also provided an "OkButton" template that helps you perform this task.

Super Invoice versus Super Browse

Many of you may wonder about the difference between "Super Invoice" and "Super Browse". Both templates are designed to support "in-place" data-entry on a list box. With Super Invoice, however, the entire transaction (the parent and all its Super Invoice children) is held in memory until the [OK] button is pressed. All disk updates are performed within a single LOGOUT+COMMIT transaction. (If the machine is turned off during the edit, then a blank auto-inc placeholder parent record would remain in the file, just as it would during a normal update of a basic auto-incremented file.)

With Super Browse, the changes to the children are saved to the disk as they are entered. If the machine is turned off before the update is complete, then the changes to the child will be saved, while the parent retains its original values. Even with the "Restore Children after Cancel" template, it is still better to use Super Invoice when possible. The one situation where Super Browse would be preferable is if there are too many child records to load practically in memory.

ABC and Legacy Template Chains

This documentation pertains to both the ABC and Legacy (a.k.a. "Clarion") Super Template sets. In some situations we've implemented features in ABC that are not in Legacy, primarily because the old template chain was to be phased out. Due to customer pressures, however, Soft Velocity

decided to reinstate support for the Legacy/Clarion chain.

Some of the Super Template features that are only in the ABC chain would be very difficult to implement in the legacy chain. However, we'll attempt to do this wherever it seems feasible to us. We apologize if this causes you any inconvenience. Please feel free to contact us if there's a particular feature in ABC that you would like to see in the Legacy chain, and we'll see if your needs can be accommodated.

For more information, see the following topics:

Adding a Super Invoice to your Application

Invoice Control

Shift Buttons

Child Window Button

Child Window OK Button

1.2 RTFM Warning!!!

It is very important that you read this documentation. If you follow the instructions step-by-step, then the usage is very simple. It is almost *impossible* if you try to do it on your own!

1.3 Installation

Installation Directory Structure

NOTE: As of version 6.6, we've changed our installation to the defacto "3rdParty" directory structure. (In Clarion 7 this is actually the "Accessory" directory.) Your old CLARIONx\SUPER directory has been renamed to CLARIONx\SUPER-OLD.

Once you've finished running the installation program, you should see the following structure under your C55 or Clarion6 directory:

```

C:\CLARION6, C:\C55, etc.
+-LibSrc      STA*.INC      (ABC headers)
+-3rdParty
| +-Bin ST_*.HLP, ST_*.CNT, STAB_CNV.DLL
| +-Template  STA?*.TPL, STA*.TPW      (ABC chain)
| |          STC?*.TPL, STC*.TPW      (Clarion chain)
| +-LibSrc    STA*.INC, STA*.CLW, STA*.TRN (ABC chain)
| |          STC*.INC, STC*.CLW, STC*.TRN (Clarion chain)
+-Images
| `--Super   *.ICO, *.CUR, *.WMF, *.GIF
+-Docs
| `--Super   *.PDF      (Documentation)
`--Vendor
   `--Super
      +-QBE
      | +-Examples
      | | +-ABC *.DCT, *.APP, *.TPS (Examples) (ABC chain)
      | | `--Clarion *.DCT, *.APP, *.TPS (Examples) (Clarion chain)
      | `--Source
      | | +-ABC *.TXD, *.TXA, *.DCT (Source) (ABC chain)
      | | `--Clarion *.TXD, *.TXA, *.DCT (Source) (Clarion chain)
      +-Tagging
      | +-Examples
      | | +-ABC *.DCT, *.APP, *.TPS (Examples) (ABC chain)
      | | `--Clarion *.DCT, *.APP, *.TPS (Examples) (Clarion chain)
      | `--Source
      | | +-ABC *.TXD, *.TXA, *.DCT (Source) (ABC chain)
      | | `--Clarion *.TXD, *.TXA, *.DCT (Source) (Clarion chain)
      `--Etc.
      +- . . .
`--SUPER-OLD      (ABC headers)
   `-- . . .

```

For Clarion 7 and later versions it should look like this (note the two trees):

```

C:\Program Files\SoftVelocity\Clarion 7
`--Accessory
   +-Bin          ST_*.HLP, ST_*.CNT, STAB_CNV.DLL
   +-Template
   | `--Win
   | |          STA?*.TPL, STA*.TPW      (ABC chain)
   | |          STC?*.TPL, STC*.TPW      (Clarion chain)
   | |          STMH*.TPW                (Shared)
   +-LibSrc
   | `--Win
   | |          STA*.INC, STA*.CLW, STA*.TRN (ABC chain)
   | |          STC*.INC, STC*.CLW, STC*.TRN (Clarion chain)
   +-Images
   | `--Super
   | |          *.ICO, *.CUR, *.WMF, *.GIF
   `--Docs
      `--Super
         *.PDF      (Documentation)

```

```

"My Documents" or "Shared Data" (depending on OS)
^-Clarion 7\Accessory
  ^-Super
    +-QBE
      | +-Examples
      | | +-ABC          *.DCT, *.APP, *.TPS (Examples)      (ABC chain)
      | | ^-Clarion     *.DCT, *.APP, *.TPS (Examples)      (Clarion chain)
      | | ^-Source
      | | +-ABC          *.TXD, *.TXA, *.DCT (Source)         (ABC chain)
      | | ^-Clarion     *.TXD, *.TXA, *.DCT (Source)         (Clarion chain)
    +-Tagging
      | +-Examples
      | | +-ABC          *.DCT, *.APP, *.TPS (Examples)      (ABC chain)
      | | ^-Clarion     *.DCT, *.APP, *.TPS (Examples)      (Clarion chain)
      | | ^-Source
      | | +-ABC          *.TXD, *.TXA, *.DCT (Source)         (ABC chain)
      | | ^-Clarion     *.TXD, *.TXA, *.DCT (Source)         (Clarion chain)
    ^-Etc.
    +- . . .

```

To prevent conflicts between old Super Template files and same-named files in our new directory structure, the new installers attempt to delete the old files. If it encounters problems, then an error will be reported during the installation. Then you must delete any of the following files from the old directories, if they also exist in the new directory structure:

```

C:\CLARION6, C:\C55, etc.
+-LibSrc          STAB*.CLW, STCL*.CLW, STAM*.CLW, STCM*.CLW,
|                STAB*.TRN, STCL*.TRN, STAM*.TRN, STCM*.TRN,
|                STDEBUG.*
+-Template        STAB*.TP?, STCL*.TP?, STAM*.TPW, STCM*.TPW,
|                STGROUPS.TPW, STDEBUG.TPW
^-Bin             ST_*.HLP, ST_*.CNT, ST_*.GID

```

For example, you can use a tool like the indispensable Beyond Compare (www.scootersoftware.com) to investigate the contents of C:\Clarion6\LibSrc and C:\Clarion6\3rdParty\LibSrc. View only files matching the mask ST*.* and hide all "orphans", which will show the files that exist in both directories. Delete the files from C:\Clarion6\LibSrc, and then do the same for the Template and Bin directories.

Filenames and Product Abbreviations

- Super Template filenames generally start with the letters "ST". That's about all you can go on most of the time. (Our image files don't follow this convention, but they are sequestered in the Image\Super subdirectory.)
- The next two letters are usually AB (for the ABC chain) or CL (for the Clarion/legacy chain). One exception is Super Stuff (MH), which uses AM, CM and MH. Also, if both the ABC and Clarion chain share a TPW, then the AB/CL are skipped and it goes on to the product abbreviation. (Again, Super Stuff is an exception, as it uses MH for the shared files.)
- There are several TPWs that are shared by multiple Super Templates: STGROUPS.TPW, STABABC.TPW, STBLDEXP.TPW, STDEBUG.TPW
- The last four characters:
 - For TPL files, the last four letters are an underscore, followed by one of the following

suffices. The exceptions are STABAEQB.TPL and ST?M_STF.TPL.

- For TPW files, the last four letters may match one of these in its entirety, or be followed by additional characters denoting the special purpose files.
- Super Stuff (MH) is an exception, in that it uses STcMxxxx, where "c" is the chain of A or C, and "xxxx" denotes the special purpose.

AEQB	Super QuickBooks-Export (i.e. Accounting-Export QuickBooks)
BRW/BW	Super Browse
DIA	Super Dialer
FF	Super Field-Filler
IE	Super Import-Export
INV	Super Invoice
LIM	Super Limiter
PCD	Super Passcode
QBE	Super QBE
SEC	Super Security
TAG	Super Tagging
MH/STF	Super Stuff (MH) (a.k.a. <i>The "MikeHanson" Templates</i>)

Update the Redirection File

The installation program is able to update your redirection file automatically. If you decline the option during the installation, then you will have to edit the redirection file yourself. The three things that must be found are the Templates, LibSrc and Images. For example, you might make the following changes to the the *.* entry in Clarion 6:

```
*.* = .; %ROOT%\examples; %ROOT%\libsrc; %ROOT%\images; %ROOT%\template; %ROOT%
\3rdParty\template; %ROOT%\3rdParty\libsrc; %ROOT%\3rdParty\images\super
```

In Clarion 7 and above it will be more like this:

```
*.* = %ROOT%\Accessory\images; %ROOT%\Accessory\resources; %ROOT%\Accessory\libsrc\win; %
ROOT%\Accessory\template\win; %ROOT%\Accessory\images\Super
```

There are *.RED examples in the SUPER\DOC directory.

Register the Templates

Clarion allows you to have multiple template sets accessible in the same application. It does this with the Template Registry. To use a Super Template, you must register it first. The installation program attempts to do this for you, but in case it fails, or if your registry becomes corrupted, then you must register them manually.

1. Load Clarion, then select the "Setup / Template Registry" pulldown menu option.
2. Press the [Register] button.
3. Select C:\CLARION\3rdParty\Template\ST_*.TPL (ABC) or ST_*.TPL (Clarion). The directory name may not exactly match your system.

Assuming this all went without a hitch, you're ready to start using the templates.

1.4 Upgrading From Earlier Versions

Changing your APPs from the Clarion/Legacy to the ABC chain

With the release of Clarion ABC, we've decided to tow the line and work with Clarion's form of Edit In Place. This makes some things easier and more powerful, and makes other things impossible. Therefore, edit in place will not work the same as it did before. Some of the key differences are:

1. You don't need to declare the associated edit controls for each column (or specify the control range for the template). Instead, you declare the settings for each individual column. If you wish, you can still create your own control for each column, or you can let the template create it for you, or use any combination of the two.
2. You are not always in "edit mode". This means that you scroll around like a normal list box, then hit the [Change] button (or double-click) to go into edit mode. (The "Auto Edit" option can streamline this.) The old template did not include a Change button with the auto-populated controls. To add a change button, add a normal button, then edit the screen source and copy the #SEQ() attribute from the Insert or Delete button over to the new Change button.
3. Some of the navigation keys while in edit mode (like PgUp, PgDn, CtrlPgUp, etc.) are not supported any longer. This is because we don't have control of it. (It's handled in the Clarion EditClass.TakeEvent method.) We've requested that TopSpeed make it possible to add these with a future version of Clarion.
4. You have many new types of edit controls now, including ENTRY, SPIN, CHECK, LIST+DROP, Lookup Button, FileDrop, FileDropCombo, TEXT, etc.
5. Your EIP embed code will be much different. For example, instead of issuing a CYCLE to stay in the same field, you must return LEVEL:Notice from the TakeAccepted method.
6. Most of the old embeds are now obsolete. They have been included as Legacy embeds, so that you can move your code into the "proper" ABC embed points.
7. In the legacy Super Invoice, your Accepted and Selected events were fired often. Every time that you exited a row, they were called. When you saved the Invoice, they were called again, and they occurred at others times as well. This has been improved in the ABC version. TakeSelected is called only when you are arriving in a field. TakeAccepted is called when you are leaving the field, and all TakeAccepted for the entire row are called when you are leaving the row.

2 Template Usage

2.1 Adding SuperInvoice to you Applications

The SuperInvoice template is implemented as a control template. This means that you can place it on any form that has a "SaveButton" template already present for the parent file. In fact, if you have multiple child files for the parent, you can have a separate Super Invoice control for each file. For example, you may have an invoice header file, a child file for products sold, and another child for services rendered.

To add a Super Invoice to your form, you must perform the following five steps:

1. Populate the Invoice's control template, which gives you the Listbox, Insert, Change and Delete Buttons.
2. Set the Options in the Invoice's template settings.
3. Populate regular Entry fields for things like file drops and specially formatted controls.
4. Add any desired code to the Embeds.
5. Add any desired Formulas.

If you wish to have a separate form for entering child information, then you can use the Child Window Button control template. This window can contain as many or as few fields from the child file as you wish. For any additional fields on the ChildWindow that are not in the EIP Invoice, make sure that you specify that these fields are "hot fields" for the Invoice template.

Finally, there is an OkButton Control Template to be used on the ChildWindow to scan the whole form for correct data entry. It's similar to the "SaveButton" in Clarion's standard templates, except that it doesn't write the data to the file.

2.2 Invoice Control Template

2.2.1 Invoice Control Template

(Procedure Control Template)

The Super Invoice template is implemented as a control template. This means that you can place it on any form that has a "SaveButton" template already present for the parent file. In fact, if you have multiple child files for the parent, you can have a separate Super Invoice control for each file. For example, you may have an invoice header file, a child file for products sold, and another child for services rendered.

To add a Super Invoice to your form, you must perform the following five steps:

1. Populate the InvoiceControl's Listbox and associated Buttons
2. Set the options in the InvoiceControl's Template Settings:
 - a. ABC Chain
 - b. Clarion/Legacy Chain
3. Populate the regular Entry fields for In-Place data entry
4. Add any desired Formulae
5. Add any desired code to the Embeds

2.2.2 Populating an InvoiceControl Template

When you populate the InvoiceControl template onto your window, you will be given the opportunity to populate the list box immediately. Feel free to do this now or later. Don't forget, though, that after finishing with the list box, you must also place the [Insert], [Change] and [Delete] buttons.

After populating the control template, don't forget the remaining steps:

2. Set the options in the InvoiceControl's Template Settings:
 - a. ABC Chain
 - b. Clarion/Legacy Chain
3. Populate the regular Entry fields for In-Place data entry
4. Add any desired Formulae
5. Add any desired code to the Embeds

2.2.3 Template Settings - ABC Chain

The Invoice control template has its prompts organized in a series of tabs. You can access these by pressing the [Extensions] button in the procedure properties window, or by accessing the "Actions" item from the listbox's pop-up while editing the window.

General Tab

Child File - This is the "detail" or "line items" file for the transaction. Normally you would add this directly beneath the Parent file in your procedure's file schematic window. The Child *must* be on the "MANY" side of a relationship with the Parent. If it isn't, then the template will warn you of this during code generation.

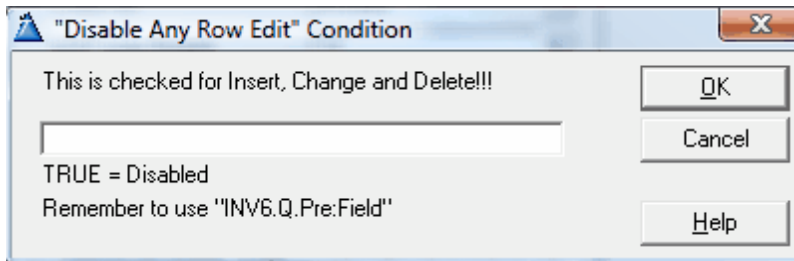
LIST Line Height (*NEW in Super Invoice 6.01*) - If you want to space out the rows, enter a number here, like 12. You can also use a variable, to control it at run-time.

Disable Popup Menu (*NEW in Super Invoice 6.01*) - By default, a popup is supported. If you wish to completely turn it off, you can disable it here. Alternatively, you can set the UsePopup class property; when you set "Disable" here, then UsePopup=0.

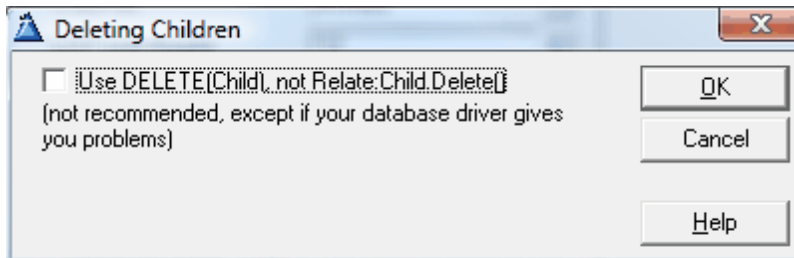
Allow Edit via Popup (*NEW in Super Invoice 6.01*) - By default, the user may use the popup to insert, change and delete records. If you wish to disable this feature, turn this option OFF.

Allow Empty Invoice - If you wish to permit the user to create a transaction without any line items, then turn this ON. If it is off, the user will be forced to add at least one row to the child file. If they try to save the empty invoice, they will be presented with the "Cannot be empty" message.

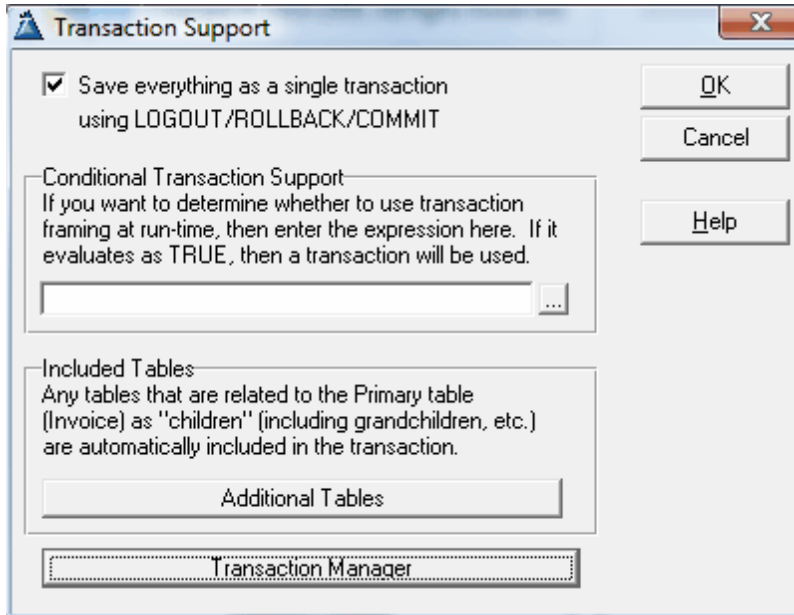
"Saving in Progress" Window Text - If you will have many child records, and updating them all will take a significant period of time, then you might want to have a window appear telling the user what's happening.



"Disable Any Row Edit" Condition - If you want to prevent all editing operations (insert, change and delete), then specify the condition here.



Use DELETE(Child), not Relate:Child.Delete() - Sometimes Btrieve and SQL can be a bit antsy in their handling of relational deletes. In some situations turn this ON will alleviate troubles with deleting invoices. ***We made significant changes to the way we handle transactions and SQL in Super Invoice 6.0, so this is will not be needed as often as it was before.***



Transaction Logout/Commit - These settings determine whether children of the parent are updated using transactions.

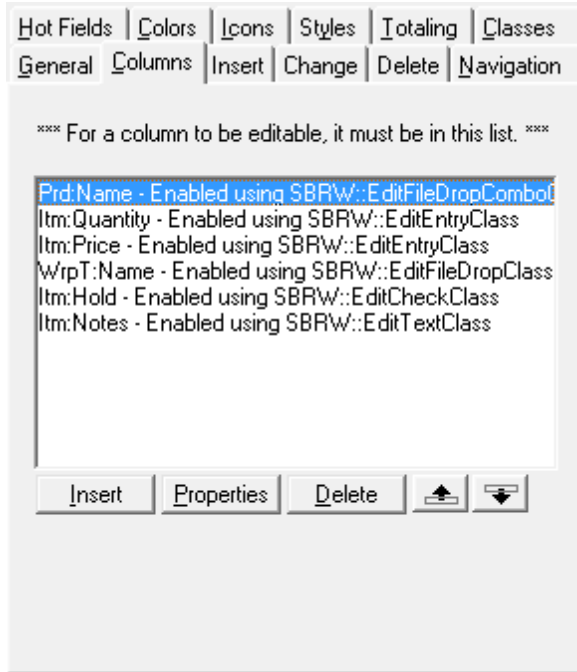
Save everything as a single transaction - Do you want to use a transaction frame?

Conditional Transaction Support - This expression will determine at run-time whether a transaction frame is used.

Additional Tables - When the Parent+Child transaction is saved, all associated files are included in a LOGOUT+COMMIT transaction frame. If you have additional files that you want to include in the transaction update operation, then you can add these here. For example, if you are adjusting inventory levels in your Product table, you would include this here.

Transaction Manager - Class settings for the transaction manager.

Columns Tab



Each editable column *must* be included in this list. Usually these will be fields from the child file, although you can also specify fields from other files or local variables. Be aware, however, that only child record is saved in the end, so any other fields will be current only during the editing session. The individual columns entries contain the following settings:

Column: General Tab

Field - This is the name of the field in the list box that you wish to edit. This must be a field in your "Child" file, specified earlier. If you specify a field that does not exist in the list box, or if the field is from a file other than the "Child" file, then the template will warn you during code generation.

Use Existing Control - Normally you will let the template create entry controls at runtime. There are situations where you may want to use your own control. For example:

- FileDrop and FileDropCombo control templates
- Many control settings, like a different picture, color, etc.
- Checkbox description different than the column header.

Class Definition - These are the standard class settings that Clarion uses in its own EIP. If you are deleting and adding columns to the list, be careful that your Object Names are unique. It is strongly suggested that you do *not* change the default class unless you *really* know what you're doing! One common reason would be to use SBRW::EditEntryLUClass instead of SBRW::EditEntryClass, to add an ellipsis [...] button beside an entry control, which you could trap with TakeButtonAccepted.

Column: File Lookup Tab

The screenshot shows a configuration window with the following settings:

- Lookup Key: Prd.NoKey
- Lookup Field: Prd.No
- Lookup Procedure: BrowseProduct
- Procedure Parameters: (empty)
- Lookup When: Accepted
- Provide Lookup Button
- Goto next field after lookup button
- More Field Assignments...

File Lookup - Use these settings to perform file lookup validation on the column. If you are using an existing entry control on your window, you can place the lookup information in that field's Actions tab instead. However, if you place it here you can get an ellipsis button beside your entry control.

Provide Lookup Button - This will place an ellipsis button [...] beside your field.

Goto next field after lookup button - When the user presses the button and then selects an item, do you want focus to automatically jump to the next column (as if they pressed Tab)?

Column: Date Entry Tab

General	File Lookup	Date Entry	Misc.	Embeds
<input checked="" type="checkbox"/> Provide Quicken Date Hot Keys (+ - T M H Y R) <input checked="" type="checkbox"/> Provide Calendar Lookup Button <input type="checkbox"/> Goto next field after calendar button				
Calendar Procedure: <input type="text" value="MH::Calendar"/>				
The calendar must be callable from this procedure. It must take the current value as a parameter. It must return the new date, or zero if cancelled.				

Date Entry - This adds quicken style date hotkeys to hop around the dates, as well as a calendar lookup button. Of course, this is only useful for a date field.

Provide Calendar Lookup Button - If you decide to use the calendar lookup, you must import CALENDAR.TXA from CLARION\SUPER\SRC_ABC\INVOICE\. Alternatively, you can use your own Calendar procedure. However, it must take the current date as a LONG parameter. It must return the newly selected date, or zero to indicate "Cancel" (stay in the column and don't change the value).

Goto next field after calendar button - When the user presses the button and then selects a date, do you want focus to automatically jump to the next column (as if they pressed Tab)?

Column: Misc Tab

The screenshot shows a dialog box with several tabs: General, File Lookup, Date Entry, Misc (selected), and Embeds. Under the Misc tab, there are three main sections:

- Place button beyond right border (if applicable)
- Checkbox uses header text (if applicable)
- A text area containing: "Disable Column Edit" Condition (True=Disabled)
Child fields must be prefixed with "INV6.Q.", as in "INV6.Q.Itm:Price".
Below this text is an empty text input field.
- Hide column when condition is true
The hiding applies when the window opens, and NOT dynamically to each row based upon its current contents.

Place button beyond right border - If you don't have room for the lookup button (or drop button) in your column, or you just like the alternate appearance, then this setting will force the button to appear on the right-hand side of the column (temporarily overlapping the column to the right).

Checkbox uses header text - If your column contains a checkbox, then this setting will cause it to copy the header text for use in the EIP checkbox control. If this is OFF, then there will be no text.

NOTE: To create a checkbox control, turn on the one of the Icon settings in the listbox formatter, but don't specific any icons. Super Invoice will handle that for you.

"Disable Column Edit" Condition - If a particular column is conditionally edited, place that condition here. Don't forget to refer to the variables in the queue (e.g. INV1.Q.Pre:Field). If the condition is True, then editing is prevented.

NOTE: The Clarion/Legacy chain handles skipped fields differently.

Column: Embeds Tab

General	File Lookup	Date Entry	Misc.	Embeds
<p>TakeSelected The return value is ignored</p> <p><input type="button" value="Data"/> <input type="button" value="Code"/></p>				
<p>Original Super TakeAccepted(),BYTE</p> <ul style="list-style-type: none"> - Return LEVEL:Notify to remain in column - Return LEVEL:Benign to proceed <p><input type="button" value="Data"/> <input type="button" value="Code"/></p>				
<p>New Clarion TakeAccepted(BYTE Action).BYTE</p> <ul style="list-style-type: none"> - Return EditAction:None to remain in field - Return EditAction:Cancel: <ul style="list-style-type: none"> - to abort EIP during normal editing - to remain in Field during "AcceptAll" validation - Return Action parameter or other EditAction:X equate <p><input type="button" value="Data"/> <input type="button" value="Code"/></p>				
<p>TakeEvent Return EditAction:X equates to indicate action</p> <p><input type="button" value="Data"/> <input type="button" value="Code"/></p>				

These buttons let you quickly jump to the embeds associated with this column. In most cases you will be using TakeAccepted, occasionally TakeSelected, and rarely TakeEvent. There are other EIP methods that can be accessed from the main embeds window, but these buttons are handy for the most common needs.

NOTE: The individual child records are stored in a queue while the invoice is being edited. When you are referencing the fields in your embeds, you must make sure that you prefix the field names with the name of the Super Invoice object (from the Classes tab described below), plus "Q". For example, `Itm:Quantity` would be `INV#.Q.Itm:Quantity`. The "INV#" is dependent on the object name.

Insert Tab

"Disable Insert" Condition - If it's sometimes invalid to add more rows to the invoice, then put that condition here. Any non-zero value indicates "disabled". You can also display a message.

Insert Position - When the user presses Insert, do you want the new row to appear at the top of the list, at the bottom, before the current row, or after the current row?

Automatically add when at bottom - When you get to the bottom of the existing rows (or the first row on an empty transaction), you can press the [Down] key (or [Tab] from the last field in the row) to create a new entry. If this option is turned off, then you must press the [Insert] button to add a new row. You can control this option at run-time using the AutoInsert property.

"Bottom Row is Empty" Condition - If the user accidentally presses [DownKey] or [TabKey] too many times (and "Automatically add when at bottom" is turned on above), then one or more new rows will be created. If you place a condition here, the user will be able to press [UpKey] or [ShiftTab] to automatically delete the rows. For example, if the row required a product number to be valid, then your condition might be *Itm:PrdNo=0*. Notice that this is the "empty" condition, not the "full" one.

Be forewarned, though, that if you also have a validation check on that field, then they may not be able to exit the field before entering a valid value. In that case, you must hit the [Delete] button to remove the bottom row.

If the user creates a bunch of blank rows, then adds a valid one at the bottom, the "empty" rows in the middle will not be deleted. The user will have to use the [Delete] button to get rid of them.

Field Priming For New Child - This is identical to field priming in a regular form, except that it applies whenever a new row is created, instead of when the form is called for an insert. The system will automatically use any initial values from the dictionary as well. Finally, you can also use the embed point to accomplish this task.

Change Tab

Hot Fields	Colors	Icons	Styles	Totaling	Classes
General	Columns	Insert	Change	Delete	Navigation

"Disable Change" Condition

TRUE = Disabled
Remember to use "INV6.Q.Pre:Field"

Beep when blocked

Message (optional):

"Disable Change" Condition - If it's sometimes invalid to change rows in the invoice, then put that condition here. Any non-zero value indicates "disabled". You can also beep and/or display a message.

Delete Tab

Hot Fields	Colors	Icons	Styles	Totaling	Classes
General	Columns	Insert	Change	Delete	Navigation

"Disable Delete" Condition

TRUE = Disabled
Remember to use "INV6.Q.Pre:Field"

Beep when blocked

Message (optional):

Confirm when row is deleted

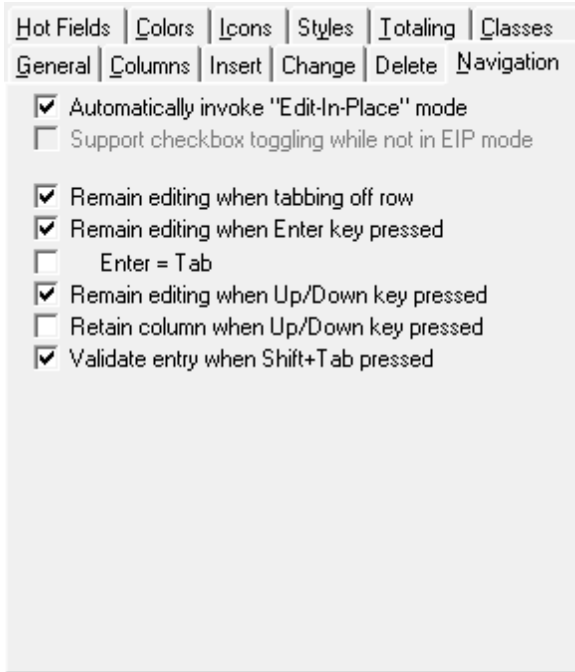
Title:

Message:

"Disable Delete" Condition - If it's sometimes invalid to change rows in the invoice, then put that condition here. Any non-zero value indicates "disabled". You can also beep and/or display a message.

Confirm when row is deleted - Do you want to warn the user when they delete a record?

Navigation Tab



Automatically invoke "Edit-In-Place" mode - When the list box is selected, do you want to automatically instigate Edit-In-Place? This is equivalent to highlighting the row and pressing [Change] or [Ctrl-Enter]. It also jumps to the clicked column.

Support checkbox toggling while not in EIP mode - If you choose not to auto-invoke EIP, then you may still want the user to be able to toggle checkboxes by clicking on them. If so, then turn this ON.

Remain Editing when tabbing off row, when Enter key pressed, when Up/Down key pressed - These three options control what happens when you leave the row using these methods. Do you want remain in EIP mode, or to revert to a regular selector bar?

Enter = Tab - Should the Enter key be interpreted as a Tab?

Retain column when Up/Down key pressed - Do you want stay in the same column when the user is scrolling up and down, or do you want to jump back to the first column when you move to a new row?

Validate entry when Shift+Tab pressed - Do you want to validate entries when the user is moving backwards through the columns?

Hot Fields Tab

If there are fields that you wish to be included that are not visible in the list box, then include them here. The most common example is if you are using the ChildWindow template, and you have fields on that window that are not in the list box.

Colors Tab

General | Columns | Insert | Change | Delete | Navigation
 Hot Fields | **Colors** | Icons | Styles | Totaling | Classes

List Selector Color During Edit-In-Place
 Use color dialog to select colors
 Text Color: COLOR:Black
 Fill Color: COLOR:Silver

Column Color Assignments
 Your SuperInvoice is not set to use custom colors. To activate custom colors for your SuperInvoice, follow these steps.
 1. If you aren't in the window formatter, go there.
 2. Right-click on the ListBox control, and select "List Box Format..."
 3. Select the field you want to "colorize", and click the "Properties" button.
 4. Check the "Color cells" Checkbox.
 5. Repeat steps 3 and 4 as necessary.
 6. When you call up the SuperInvoice actions window, this tab will show the fields you've colored.

List Selector Color During Edit-In-Place - The bold highlighting on the row can be distracting to the user during EIP. To change the selector bar color during EIP, use these settings. It makes it much easier to see which field they are in.

The rest of the Color settings are the same as a regular BrowseBox.

Icons Tab

The Icon settings are the same as a regular BrowseBox.

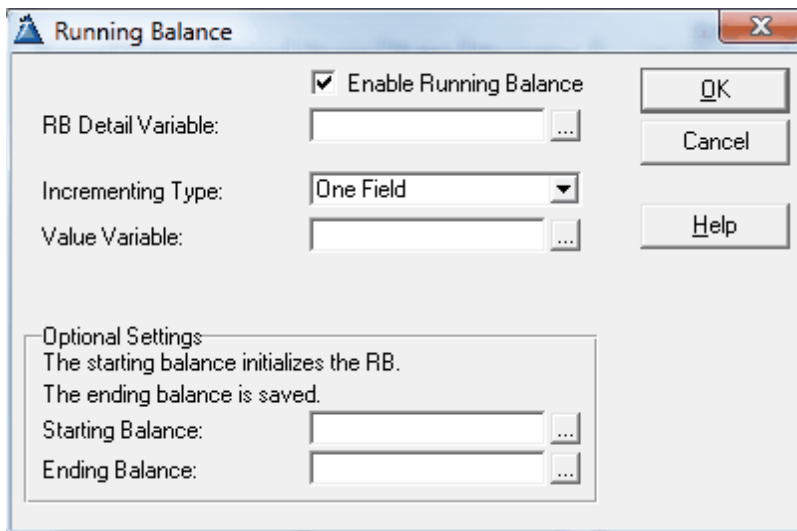
Styles Tab

The Style settings are the same as a regular BrowseBox.

Totaling Tab

These totals are set up in the same was as regular totals on a browse box. You set the "Target Field" and the "Field to Total". You can total "Each Record Read", or only when a "Specified Condition" is true.

You can also do a **running balance** total:



Enable Running Balance - Turn on the running balance feature.

RB Detail Variable - This is the field that will hold the running balance on each row. Usually you would use a local variable, although you may want to store the running balance in a child field.

Incrementing Type - This specifies the type of source:

One Field - You are totalling a single field, potentially with both positive and negative values.

Two Fields - You separate Positive and Negative value variables (e.g. Debit and Credit).

Expression - Your situation is more complex, so enter the expression representing the value for each row. Your expression must use the invoice queue variables, not the original fields.

Starting Balance - If your balance doesn't start with zero, then specify the source for that here.

Ending Balance - The final balance is stored in this variable.

Classes Tab

These settings control some of the ABC Class attributes. Normally you will not need to change anything here. The most important element for the average developer is the "Object Name". This is the label that you use to reference the fields in the queue. For example: `INV6.Q.Itm:Quantity`. If you don't like to remember "INV6", then you can change it to something else like "Invoice" or "ItemList".

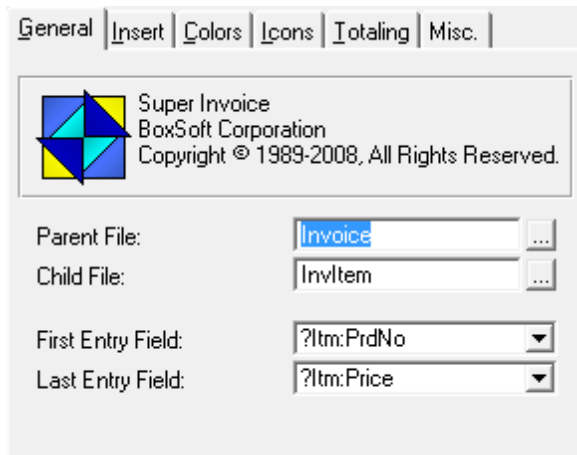
After setting the template options, don't forget the remaining steps:

3. Populate the regular Entry fields for In-Place data entry
4. Add any desired Formulae
5. Add any desired code to the Embeds

2.2.4 Template Settings - Clarion/Legacy Chain

The Invoice control template has its prompts organized in a series of tabs. You can access these by pressing the [Extensions] button in the procedure properties window, or by accessing the "Actions" item from the listbox's pop-up while editing the window.

General Tab



General | Insert | Colors | Icons | Totaling | Misc.

Super Invoice
BoxSoft Corporation
Copyright © 1989-2008, All Rights Reserved.

Parent File: Invoice ...

Child File: InvItem ...

First Entry Field: ?!tm:PrdNo ▼

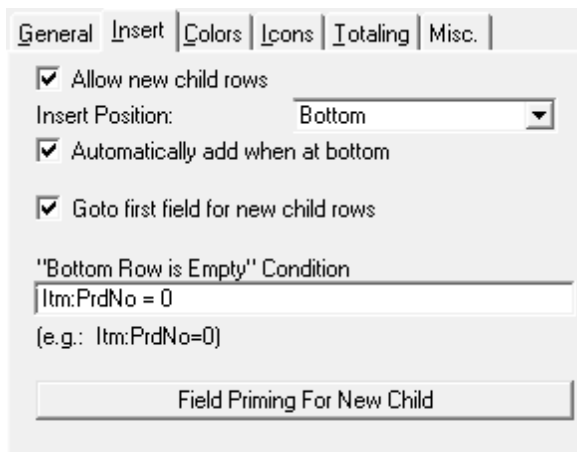
Last Entry Field: ?!tm:Price ▼

Parent File - This is the "header" file.

Child File - This is the "detail" or "line items" file for the transaction. Normally you would add this directly beneath the Parent file in your procedure's file schematic window. The Child *must* be on the "MANY" side of a relationship with the Parent. If it isn't, then the template will warn you of this during code generation.

First/Last Entry Field - These two fields specify a range of controls (in screen structure order) to be used for EIP.

Insert Tab



General | Insert | Colors | Icons | Totaling | Misc.

Allow new child rows

Insert Position: Bottom ▼

Automatically add when at bottom

Goto first field for new child rows

"Bottom Row is Empty" Condition

!tm:PrdNo = 0

(e.g.: !tm:PrdNo=0)

Field Priming For New Child

Allow new child rows - Turn this OFF to prevent the user from creating additional rows (beyond those that are pre-populated).

Automatically add when at bottom - When you get to the bottom of the existing rows (or the first row on an empty transaction), you can press the [Down] key (or [Tab] from the last field in the row) to create a new entry. If this option is turned off, then you must press the [Insert] button to add a new row.

Goto first field for new child rows - When a new row is added, should focus always go to the first EIP field in the row (rather than staying in the same column)?

"Bottom Row is Empty" Condition - If the user accidentally presses [DownKey] or [TabKey] too many times (and "Automatically add when at bottom" is turned on above), then one or more new rows will be created. If you place a condition here, the user will be able to press [UpKey] or [ShiftTab] to automatically delete the rows. For example, if the row required a product number to be valid, then your condition might be *Itm:PrdNo=0*. Notice that this is the "empty" condition, not the "full" one.

Be forewarned, though, that if you also have a validation check on that field, then they may not be able to exit the field before entering a valid value. In that case, you must hit the [Delete] button to remove the bottom row.

If the user creates a bunch of blank rows, then adds a valid one at the bottom, the "empty" rows in the middle will not be deleted. The user will have to use the [Delete] button to get rid of them.

Field Priming For New Child - This is identical to field priming in a regular form, except that it applies whenever a new row is created, instead of when the form is called for an insert. The system will automatically use any initial values from the dictionary as well. Finally, you can also use the embed point to accomplish this task.

Colors Tab

The Color settings are the same as a regular BrowseBox.

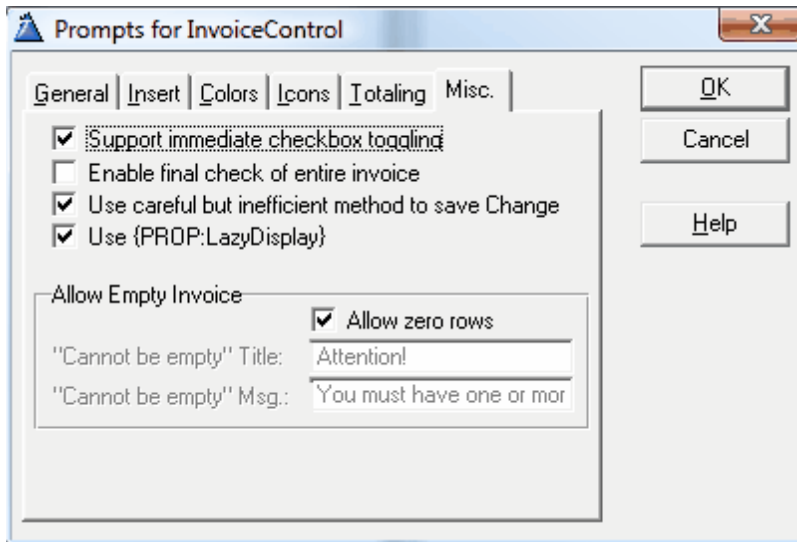
Icons Tab

The Icon settings are the same as a regular BrowseBox.

Totaling Tab

These totals are set up in the same way as regular totals on a browse box. You set the "Target Field" and the "Field to Total". You can total "Each Record Read", or only when a "Specified Condition" is true.

Misc Tab



Support immediate checkbox toggling - When the user clicks on a checkbox column, should the checkbox be toggled automatically? The alternative is for the focus to arrive, but not affect the value until it's clicked again or the user presses the Space key.

Enable final check of entire invoice - This instructs Super Invoice to validate all fields in all rows when the transaction is saved. If you turn this ON, be aware that the Accept processing will be executed during the save process, as well as during data entry.

Use careful but inefficient method to save Change - This option deletes all existing child rows from the table before resaving them. Otherwise, if a record hasn't changed, then it won't be rewritten.

Use {PROP:LazyDisplay} - Use the "Lazy Display" to reduce flicker. There are very few situation where you would need to turn this off.

Allow Empty Invoice - If you wish to permit the user to enter no line items, then turn this ON. If it is OFF, the user will be forced to add at least one row to the child file. If they try to save the empty invoice, they will be presented with the "Cannot be empty" message.

Skipping Columns

To skip a column, you do not disable the entry field. Instead, you must use the "Skip" setting. Search for Skip in the generated source, and you'll see it several times. For example, in the Clarion/Legacy example program, you'll see the following structure for the Child QUEUE:

```
ChildList:Queue      QUEUE,PRE()
INV6::Itm:PrdNo     LIKE(Itm:PrdNo)
INV6::Prd:Name      LIKE(Prd:Name)
INV6::Itm:Quantity  LIKE(Itm:Quantity)
INV6::Itm:Price     LIKE(Itm:Price)
INV6::Itm:Tax1      LIKE(Itm:Tax1)
INV6::Itm:Tax2      LIKE(Itm:Tax2)
```

```

INV6::Itm:Extended    LIKE(Itm:Extended)
Skips                GROUP
Itm:PrdNo:Skip       BYTE
Itm:Quantity:Skip    BYTE
Itm:Price:Skip       BYTE
                    END!GROUP
Skip                 BYTE,DIM(3),OVER(Skips)
INV6::Position       STRING(1024)
                    END!QUEUE
    
```

If we wanted to skip the Itm:Quantity field, we would do this:

```
ChildList:Queue.Skips.Itm:Quantity:Skip = True
```

This value is stored for each queue record, so that you can conditionally set the Skip attribute differently for each row.

Also, if a column is to be hidden/skipped for the duration of a session, then set the column width to zero after the window is opened, but before the InitWindow routine is called. For example:

```
?ChildList{PROPLIST:Width, INV6::Column:Itm:Price} = 0
```

After setting the template options, don't forget the remaining steps:

3. Populate the regular Entry fields for In-Place data entry
4. Add any desired Formulae
5. Add any desired code to the Embeds

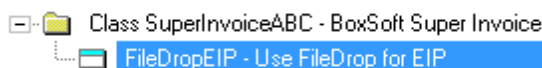
2.2.5 Adding the Regular Controls for "In-Place" Entry

If you're using the Clarion/Legacy chain, then you must create an entry control for each EIP column.

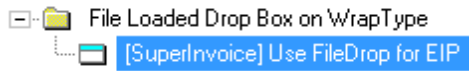
If you're using the ABC chain, then the template will automatically create controls for data entry at runtime. There are a few situations where you might want to create your own entry controls, though. For example, you may want to use the FileDrop or FileDropCombo. Or you may want to use a checkbox, but you want to use a different description from the automatic usage of the column header. Or you may want to use a picture that's different from the one defined in the dictionary. Or you might want to use a special font.

For some of these (like the picture, color, checkbox text, font, etc.), you can use property syntax assignments in the CreateControl method. However, it's often easier just to create a regular control, and to tell the template to use that instead of doing it behind the scenes.

If you plan to use a FileDrop or FileDropCombo, you must populate an additional template in the Extensions window. Highlight the FileDrop(Combo), then press [Insert]. Depending on your control type, you'll be able to choose one of two helper templates from the list:



After selecting the template, it will appear below your FileDrop(Combo) in the extensions window:



NOTE: When you're using a FileDropCombo, your USE variable *must* name the corresponding EIP field to be updated. For example, if your EIP field is **Cus:City**, and your FileDropCombo lets them select a record from the **City** file, then the FDC control's USE variable *must* be **Cus:City**, not **Cit:City**.

After adding your regular entry controls, don't forget the remaining steps:

4. Add any desired Formulae
5. Add any desired code to the Embeds

2.2.6 Formula Classes

There are two additional formula classes provided by the InvoiceControl template:

Prime Fields of ... record for new Row - This is the fourth method for priming fields in a new row. The other three are "Dictionary", "Template Prompts", and "Embed".

Format a variable in the ... - This class is used to calculate values to be displayed in a row.

After adding your regular entry controls, don't forget the remaining step:

5. Add any desired code to the Embeds

2.2.7 Embed Points

When it comes to hand-code, there are two very important things to realize about the ABC Edit-In-Place:

1. The controls are not updating the file's fields. Rather, the queue buffer receives all field values until the row is saved. Therefore, it is useless to say:

```
IF Itm:Quantity=0 THEN RETURN Level:Notify.
```

Instead, you should say:

```
IF INV#.Q.Itm:Quantity=0 THEN RETURN Level:Notify.
```

"INV#" represents the name of your Invoice object. If you don't know the name of the object, look in the Classes tab of your Invoice's extension settings. You can change it there, if you wish.

This makes it difficult for normal templates, as they expect to work with regular fields. We made our template explicitly handle Clarion's FileDrop and FileDropCombo templates.

Supporting each additional template will require special treatment.

2. EIP is controlled by the AskRecord method of the SINV::InvoiceClass. It uses its own ACCEPT loop, and it passes very few events through to your own code. Normally you should use the TakeSelected and TakeAccepted methods to handle your validation code. When you want to stay in a field (e.g. because of validation failure), then you must return LEVEL:Notify from TakeAccepted. For example:

```
IF INV3.Q.Itm:Type = 'X' !If it's a bad value
  RETURN Level:Notify !Stay in the same field
END
```

If, instead, you're using TakeEvent, then you must return EditAction:None to stay in the same field.

NEW in Super Invoice 6.01 - You can check the value of the Request property to determine if you are Inserting, Changing or Deleting a row. Compare it against the standard equates: InsertRecord, ChangeRecord and DeleteRecord. Depending on your scope, you'll use SELF.Request or INV#.Request (e.g. INV3.Request). This feature is not available in the Clarion/Legacy chain, because the system is basically always in "Change" mode during the EIP process.

NEW in Super Invoice 6.03 - You can check the value of the AcceptAll property to determine if you are in "all columns in the row" validation mode. The value will be True or False. Depending on your scope, you'll use SELF.AcceptAll or INV#.AcceptAll (e.g. INV3.AcceptAll). For the Clarion/Legacy chain, this is called INV#:AcceptAll. You can also set this value to False during the validation, to halt the process.

The easiest way to access the Super Invoice embeds is through the individual column settings in the extension. You can also get them through the regular Procedure Embeds button. Here are the methods that you are most likely to use:

Invoice Manager Methods

BeforePrimingChild, BeforePrimingChildR, BeforeAddingChild, BeforeAddingChildR, BeforeDeletingChild, BeforeDeletingChildR, NotUpdatingChild - These methods are useful if you are doing things like maintaining inventory levels in another file. These methods are called when inserting and deleting records from the child file. For example, BeforeAddingChild could have:

```
Prd:No = Itm:PrdNo
Access:Product.Fetch(Prd:NoKey)
Prd:OnHand -= Itm:Quantity
Access:Product.Update
```

The code in BeforeDeletingChild would be the same, except that it would be "+=" instead of "-=". In this situation you should also include the Product file as an "Additional LOGOUT File".

The methods with "R" suffix can return LEVEL:Notify or LEVEL:Benign to notify the system that the save operation should be aborted.

NotUpdatingChild is called for those rows that haven't changed. It may still be necessary for you to do something in those situations.

CalcTotals - If you want to modify how the totals are calculated, this is the place to go. All total calculation code is generated directly in your source module.

IsEmptyRow - This method returns True if the row is considered to be "Empty" and is a candidate for automatic deletion from the bottom row of the list box. The expression that you place into the "Bottom Row is Empty" Condition is generated into this method, or you can place the code into this method yourself.

SetQueueRecord - This method is used to assign the values from the file fields into the queue fields. If you have any extra formulae, like calculating extended totals, then you should place these here. This is one situation where you don't use the INV#.Q.Pre:Field notation. Instead, use the field names themselves.

EqualOriginalRow - This virtual method is called by the invoice class to determine if a particular row has changed. You may want to override it if there are special circumstances where the row would appear to be the unchanged, when it really has changed.

InsertEnabled, ChangeEnabled, DeleteEnabled - These three virtual methods are called by the invoice class to determine whether the update operations are acceptable for the current row. Usually you'll use the template settings to accommodate this, but you may want to override these directly.

LoadOriginal - This is usually called after the initial records are loaded, as a "snapshot" of the starting point. When it comes time to save the invoice, the class compares the current records in the queue with those saved by LoadOriginal. If they are different, then the invoice is updated on disk.

In some situations you may want to pre-load default records when an invoice is first added. In this situation, LoadOriginal will show no records, which doesn't correspond to the defaults that you've pre-loaded. In that situation, call LoadOriginal after you pre-load the records, and it won't be confused by your defaults.

InsertQueue, DeleteQueue - These virtual methods are responsible for adding and deleting records in the queue. If you need to do something each time a queue entry is added and/or deleted, then use these derived methods.

TakeAskRecordEvent - This virtual method is called whenever the AskRecord method processing an event. It gives you the ability to hook into the activity of the other hidden EIP ACCEPT loop. You can return the following values:

LEVEL:Benign	The event should be processed normally (the default from the base virtual method).
LEVEL:Program	The event was consumed, but EIP should continue.
LEVEL:Notify	EIP should save the row and quit.
LEVEL:Fatal	EIP should quit without saving the row.

NotUpdatingChild - This virtual method is called when a row has not changed, so does not need to be re-written to the file. Use the derived method to perform any special operations in this situation.

EditClass Methods

CreateControl - This method is called when the EIP is initiating. Even if you are using your own entry

control, it is still called. If you want to tweak the appearance of your control, you should do it here.

TakeAccepted - This method is called each time you exit a field, and for all fields on the row when you are leaving the row. If there is a problem with the field's value, then return Level:Notify. Be sure to use the INV#.Q.Pre:Field notation when checking fields' values.

TakeSelected - This is similar to TakeAccepted, except that it is called when you enter into the field, not when you exit it. It is *not* called when leaving the row.

TakeEvent - This method is called before TakeAccepted and TakeSelected. It actually directs traffic to the appropriate method. You could use this if you wanted to add additional hot keys to a field.

SetupAlertKeys - If you are adding your own hotkeys to a column, you must assign them using runtime property syntax in this method. For example:

```
SELF.Feq{PROP:Alt,10} = Ctrl1T
```

Of course, it is your responsibility to trap that keycode in TakeEvent with something like this:

```
IF E=EVENT:AlertKey AND KEYCODE()=Ctrl1T
    CHANGE(SELF.Feq, TODAY())
END
```

Column Numbers

There are situations when you will need to refer to column numbers. Rather than hard code these as actual numbers, you can reference them via automatically generated properties. For example:

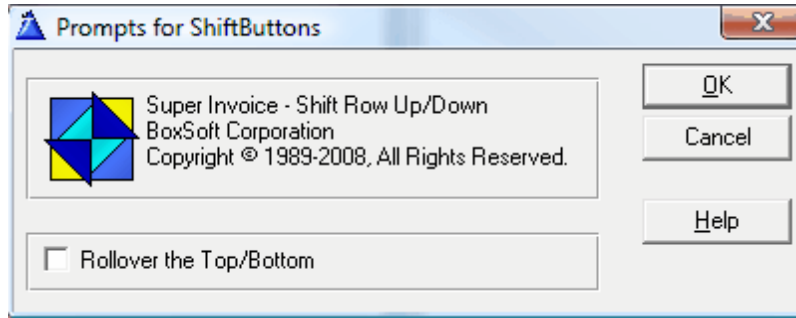
```
SELF.ColumnNumber.Prd:Name
SELF.ColumnNumber.Itm:Quantity
SELF.ColumnNumber.Itm:Price
SELF.ColumnNumber.Itm:Tax1
SELF.ColumnNumber.Itm:Tax2
SELF.ColumnNumber.Itm:Extended
SELF.ColumnNumber.Wrpt:Name
SELF.ColumnNumber.Itm:Notes
SELF.ColumnNumber.Itm:Hold
```

Depending on where you are in the code, you may need to use the invoice object's name rather than SELF (e.g. INV6.ColumnNumber.Prd:Name).

2.3 Shift Buttons

(Procedure Control Template)

This control template provides buttons that can shift individual rows up and down. The window must contain an Invoice control template before you can populate it.



Rollover the Top/Bottom - When a row reaches the top/bottom of the list, do you want it to flip to the other end?

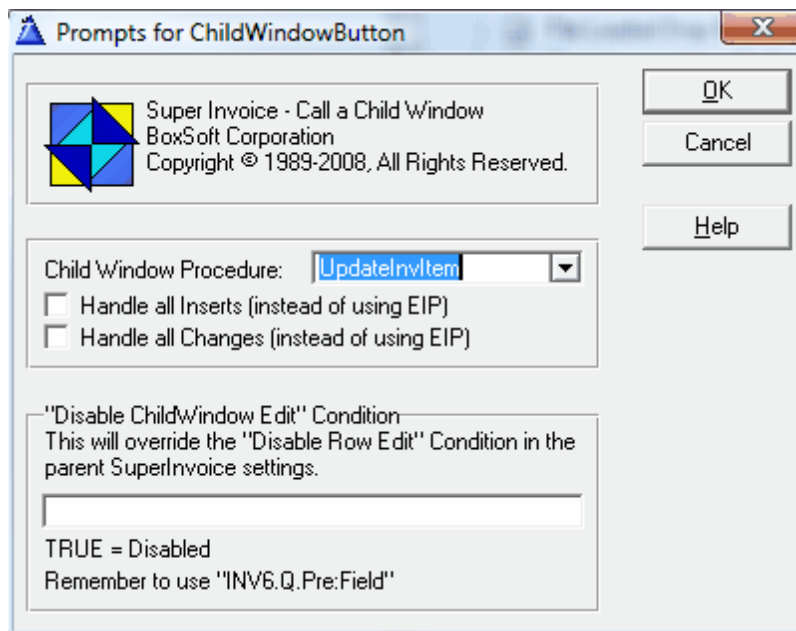
2.4 Child Window Button

(Procedure Control Template)

If you wish to call a separate window to enter child information, use this control template to call it. It automatically displays the changes upon return from the window. There are two things to remember:

1. If you are editing fields that aren't displayed in the listbox, then you must add those fields as "Hot Fields" in the invoice.
2. The ChildWindow must not be a Form with a SaveButton. Rather, it should be a Window with an OkButton template from the Super Invoice templates. For more information, see Child Window OK Button.

You cannot populate this control until you have a Super Invoice control in place.



Child Window Procedure - This is the window procedure with the Super Invoice OkButton template.

Handle all Inserts (instead of using EIP) - Normally, all inserts are handled by EIP. If you want these to be intercepted and rerouted to the ChildWindow, then turn this on.

Handle all Changes (instead of using EIP) - This intercepts the regular Change button. It's the functionally similar to deleting the record Change button, and replacing it with this ChildWindow button control.

NOTE: If you turn on both of these "Handle" options, then EIP will never be used; ALL edits will be processed by the child window. This is useful if your editing works better with a regular window rather than EIP, but you still want all of the other benefits of Super Invoice (transaction processing, etc.) In this situation, you may want to HIDE the child

window button when the Super Invoice window opens, so that the user sees only the regular Insert, Change and Delete buttons.

2.5 Child Window OK Button

(Procedure Control Template)

This control template is similar to the "SaveButton" control template that comes with the stock templates. The difference is that it doesn't save the record. Instead, it just scans the entire window to verify that the data is correct. Place this on a Window, in much the same way as you would find a "SaveButton" template on a Form.

This procedure must be called from a Super Invoice window using the Child Window Button control template.



Messages and Titles (*NEW in Super Invoice 6.01*) - Specify how to display the current editing status.

3 Appendices

3.1 Example Programs

The Super Invoice templates include one example program for Clarion/Legacy (INVOICE1), and one for ABC (INVOICE1). They demonstrate a simple invoice program using Customers, Products, Invoices, and InvItems. The two screens of consequence here are UpdateInvoice and UpdateInvItem.

UpdateInvoice contains a Super Invoice control with the corresponding Insert, Change and Delete buttons. It also contains a ChildWindowButton control to call UpdateInvItem, and the Shift buttons.

UpdateInvItem is a full window to update the child record. This is especially useful if you have fields that won't fit on the scrolling area (like TEXT fields), or too many editable fields to fit comfortably in a list box.

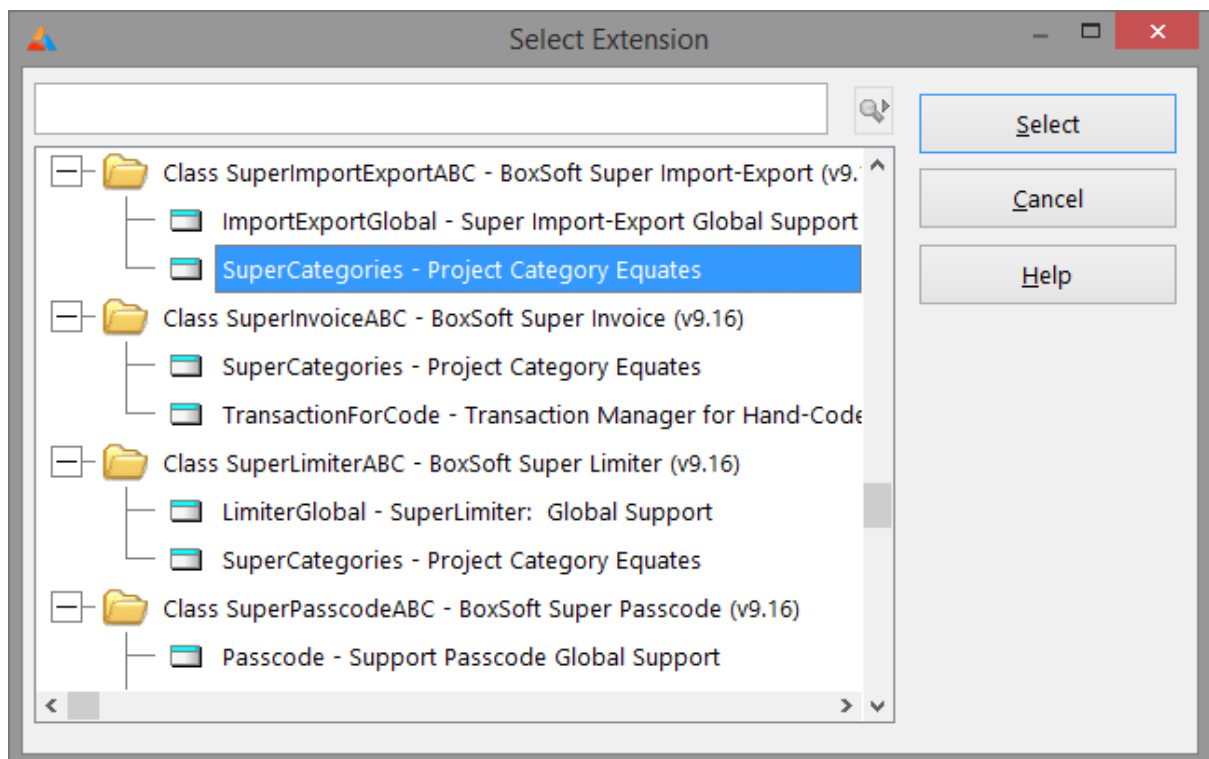
New in Super Invoice 6.70 - The ABC version contains examples of both FileDrop and FileDropCombo templates, along with checkbox and TEXT columns.

3.2 Project Defines

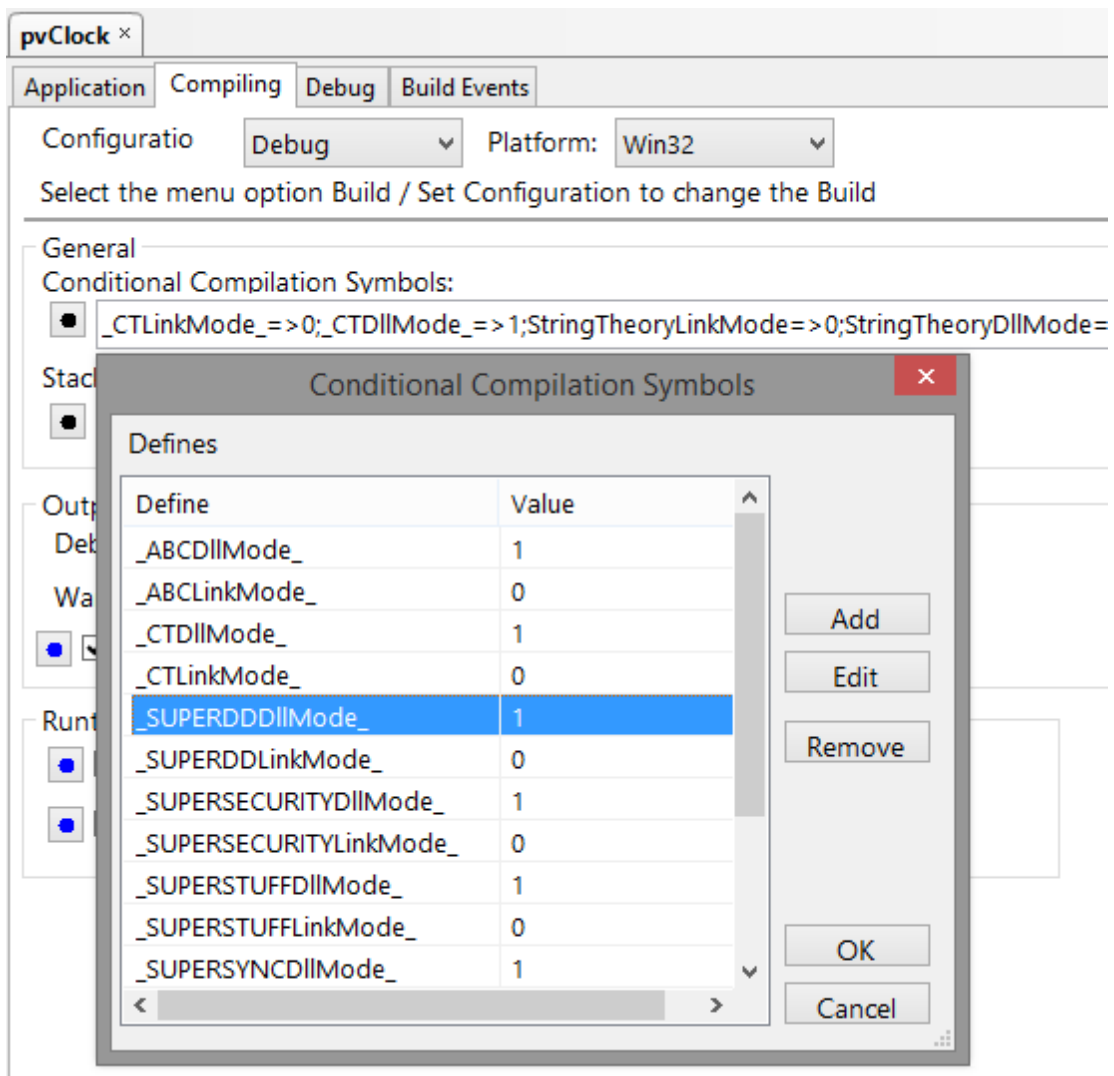
Prior to our 7.0 template versions, we were utilizing the same LINK and DLL Project Defines as the ABC libraries: `_ABCLinkMode_` and `_ABCDIIMode_`. This caused all of our libraries to be included in your base/dictionary DLL, even if you weren't using them in a particular development project.

To make this inclusion more selective, as of our 7.0 versions we changed to use various other switches. Some of these are product related (e.g. Super QBE uses `_SuperQBELinkMode_` and `_SuperQBEDIIMode_`), while others are associated with one of our shared base classes (e.g. Drag & Drop uses `_SuperDDLLinkMode_` and `_SuperDDDIIMode_`). Usually the templates (especially the global ones) automatically add the necessary entries to your Project Defines. If you happen to use the templates in your APPs in the wrong combination, these can be inadvertently omitted.

For APP-based systems, you can force the switches to be included by using the Super Categories global extension template. Every one of the Super Templates has this extension to apply its own switches, so if you're using multiple templates in a particular APP, you may have to add this extension for each of the products. (As was mentioned above, if there's already a global extension populated for a given Super Template, then you don't have to add this extension for that product.) Even if it's not needed, there's no problem with adding the SuperCategories global extension.



For hand-coded PRJ-based systems, you must add the switches manually. Take note of their names in the INC files, and then add them to the project settings like this:



3.3 Troubleshooting

Problem:

I'm placing validity checks into my TakeAccepted method, and it doesn't seem to be responding properly.

Solution:

Remember that TakeAccepted uses the queue variables, not the fields themselves. For example, your field is called Itm:Quantity. In the TakeAccepted, TakeSelected and TakeEvent methods you would refer to it as INV#.Q.Itm:Quantity. (The "#" will vary depending on your template settings. Look in the "Classes" tab for the actual name of the object.)

Problem:

My TakeAccepted event was called, even though I was not in that field.

Solution:

If you are leaving a row, the Invoice class automatically fires the TakeAccepted methods for all editable columns. This is similar to the "AcceptAll" mode that is used when you press the [OK] button on an update form. You must anticipate this behavior, and write your TakeAccepted methods accordingly.

Problem:

I'm trying to use a FileDropCombo with BrowseEntry (Edit-In-Place), but the value that appears in the FDC doesn't match the current value of my EIP field, and values chosen with the FDC are not saved to the EIP field.

Solution:

When you're using a FileDropCombo, your USE variable must name the corresponding EIP field to be updated. For example, if your EIP field is Cus:City, and your FileDropCombo lets them select a record from the City file, then the FDC control's USE variable must be Cus:City, not Cit:City.

3.4 Contacting Technical Support

If you have any troubles with this product, then please contact:

Mitten Software
2354 West Wayzata Blvd
Second Floor, Suite H
Long Lake, MN 55356

Voice: (952) 745-4941
Fax: (952) 745-4944

Internet: www.mittensoftware.com
answers@mittensoftware.com
www.boxsoft.net
www.boxsoft.net/contact.htm

3.5 License Agreement

One License per Developer

This Super Template product is comprised of the templates, default applications, libraries, source code, documentation, and help files provided with the package. You must have a separate registered copy for each developer using it.

Redistribution

You are allowed to use the product for any programs that you create, and you are permitted to distribute the generated source code. You may not, however, distribute any portion of the product in its original or modified form without the prior written consent from BoxSoft Development.

One exception to this is the example programs provided with this installation or separately from BoxSoft or its agents: these may be distributed without penalty, in either their original or a modified state.

Disclaimer

BoxSoft Development does not warranty this software for any use. Any expenses or lost time due to errors in this product are not the responsibility of BoxSoft Development. We will attempt to fix any errors that are brought to our attention, but we are not legally liable for any lack of correctness of the product.

Index

- A -

ABC 10
 Accepted 10
 Adding Rows 12, 26
 Adding SuperInvoice to you Applications 11
 Adding the Regular Controls for "In-Place" Entry 29

- B -

Buttons 34

- C -

CalcTotals 30
 Child File 12, 26
 Child Window 37
 Child Window Button 35
 Child Window OK Button 37
 Class Libraries 6, 39
 Classes 12, 26
 Columns 12, 26
 Contacting Technical Support 42
 CreateControl 29

- D -

Defines 39
 Directories 6
 Disappear 39
 DLLMode 39

- E -

EditClass 10
 Embed Points 30
 Entry Controls 29
 Example Program 38

- F -

FileDrop Template 29

Formula Classes 30
 Freeze 39

- G -

GPF 39

- I -

Include Files 6
 Installation 6
 Introduction 3
 Invoice Control 30

- L -

License Agreement 43
 LinkMode 39
 LOGOUT 12, 26

- M -

Methods 30

- N -

Navigation 12, 26

- O -

OK Button 37

- P -

Populating an InvoiceControl Template 12
 Project Defines 39
 Property Syntax 29

- R -

Redirection File 6
 Registering the Template 6
 Rollover 34
 RTFM Warning 5

- S -

Selected 10

Shift 34

Support 42

- T -

TakeAccepted 30

TakeEvent 10, 30

TakeSelected 30

Tech Support 42

Template Registry 6

Template Settings 12, 26

Troubleshooting 41

- U -

Upgrading From Earlier Versions 10