Formula Editor Guide

Getting Started with Formulas | February 24, 2022

4800 Falls of Neuse Road, Suite 600 | Raleigh, NC 27609

p (800) 848–0143 | f (919) 755–8350 | www.softprocorp.com

Copyright and Licensing Information

Table of Contents

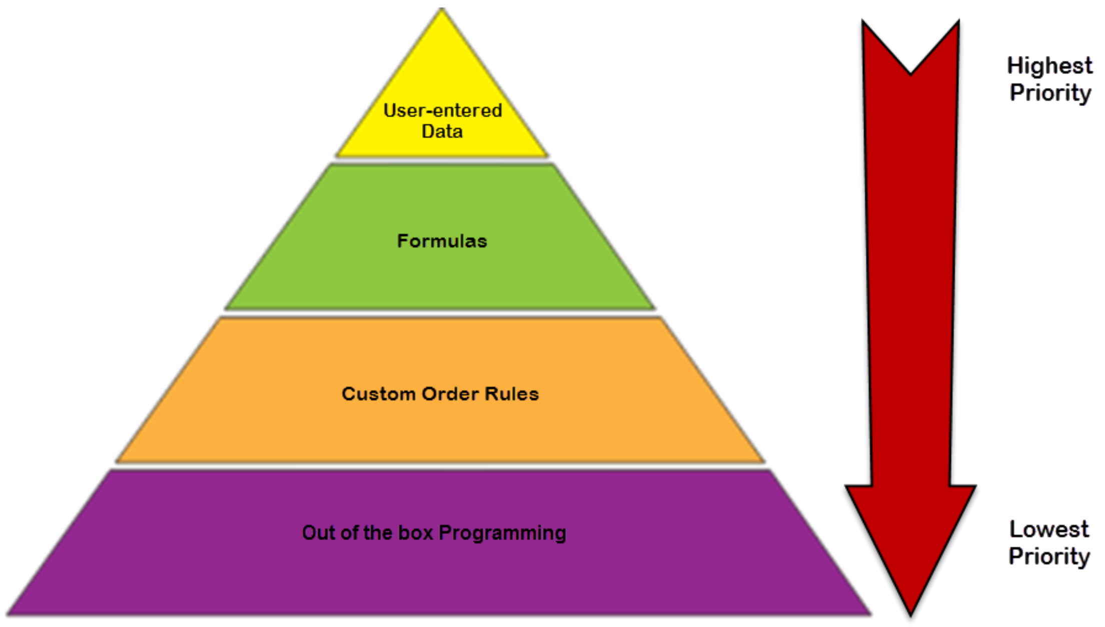
Formula Editor

What is the SoftPro Select Formula Editor?

The SoftPro Select Formula Editor is a tool within the SoftPro Select application where users can add custom business logic to a field or override existing business logic. This functionality allows end users to customize how an individual order acts based on their business processes. For example, if the Commitment Number field should always copy in the Order Number, a formula can be used to automate this data-entry. Formulas can also use logic to make more complicated decisions based on other fields in the Order. Formulas are usually saved in templates so they automatically apply to orders without user having to add them. They can also be stored in Lookup table entries and then retrieved from Lookup tables. They can also be loaded from a database of saved Formulas.

Where do formulas fit into the business logic of the application?

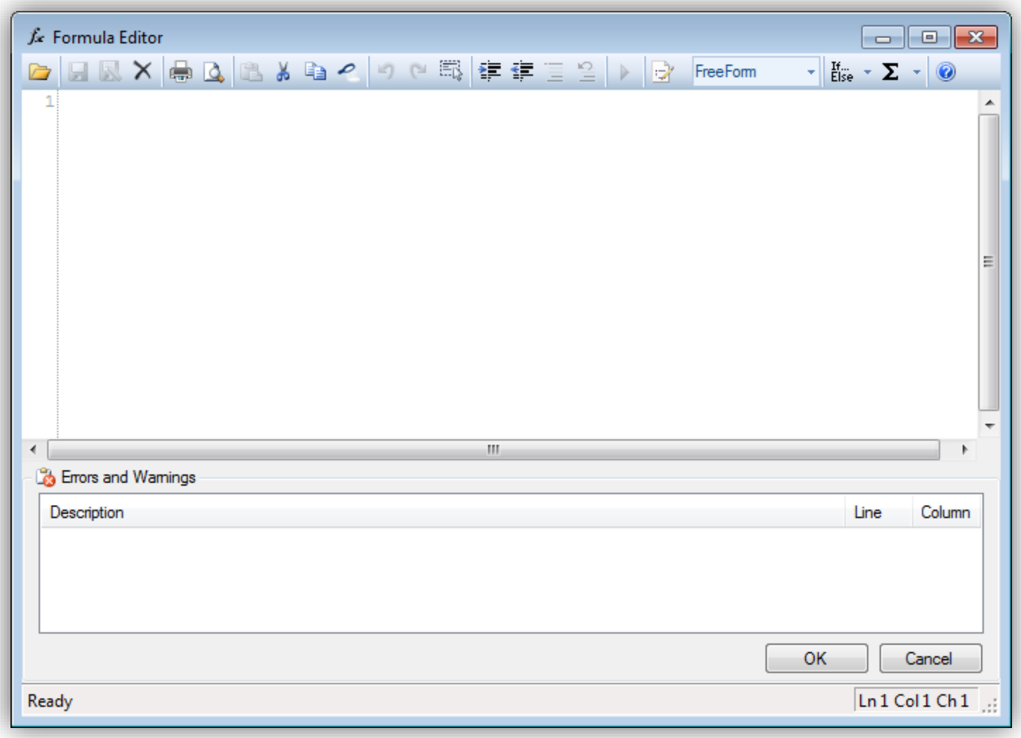
Formulas fit into a hierarchy of business logic used by the application to determine the value for a particular field in SoftPro Select.



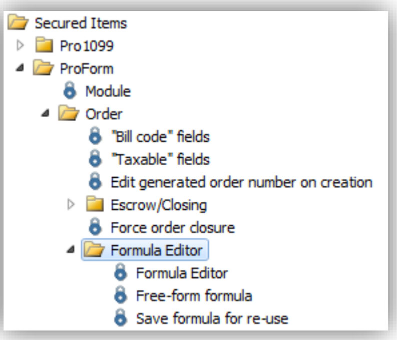
User-entered data is always the highest priority data. It will trump all other data including internal, automatic business logic (Out of the box functionality OR Custom Business Logic that populates a field; i.e. the Disbursement date populates based on the Settlement date). If no user-entered data is present, then SoftPro Select will look for any formulas for the field. If a formula exists, then its value is used before any other. Otherwise, the internal business logic or Custom Order Rules are used to determine the value for the field. If there is no business logic for the field, then the field will receive no value.

Accessing Formulas

Press the Function Key F8 in any field to open the Formula Editor Interface:



Access to the Formula Editor can be restricted with permissions



Permission	Right	Description
Formula Editor	Execute	Allows you to access the formula editor by pressing the F8 key.
Free-form Formula	Edit	Allows you to create a formula as a coded, free-form formula instead of the pre-defined formulas
Save formula for re-use	Execute	Allows you to save and delete formulas from the database.

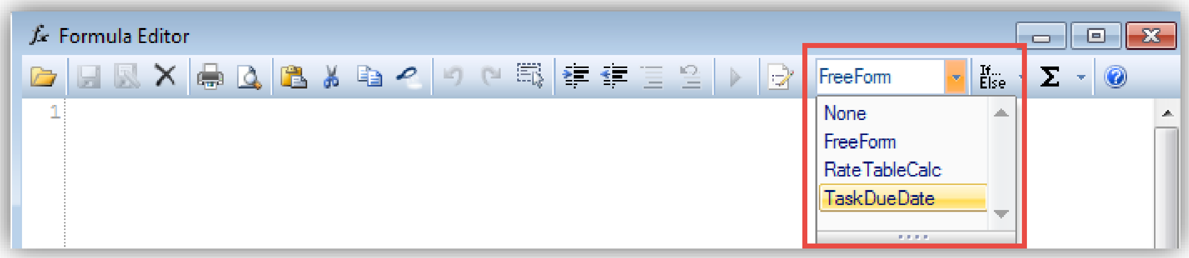
Free-Form vs. Pre-defined Formulas

A free-form formula is coded using the coding language. This is the most flexible way of creating a formula. However, this can be daunting. Pre-defined formulas exist for common scenarios. When a pre-defined formula is selected, the text entry area is replaced with a user-interface that is specific to the pre-defined formula.

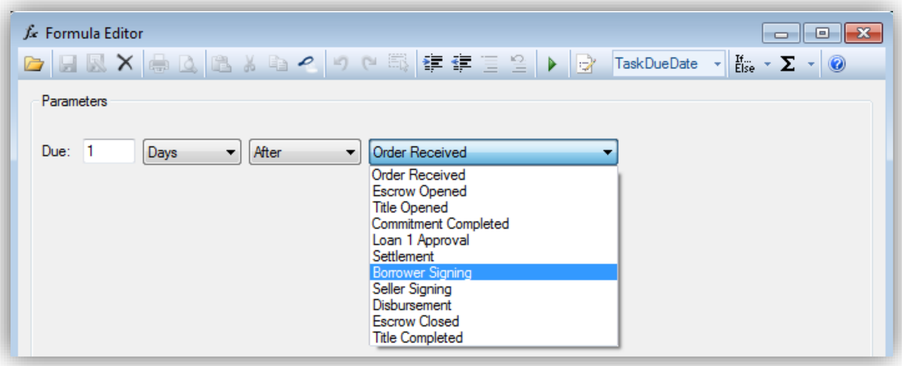
Using Pre-defined Formulas

1. The **TaskDueDate** function can be used in any Date/time field. It is most often for Checklist and Requested Task Due date fields.

   a. Press F8 in the Due field of a task or other date field.
   b. Change the drop-down option from **FreeForm** to **TaskDueDate:**



   c. Use the Pre-defined interface to set a Due Date for tasks or other auto-calculations for date/time fields.



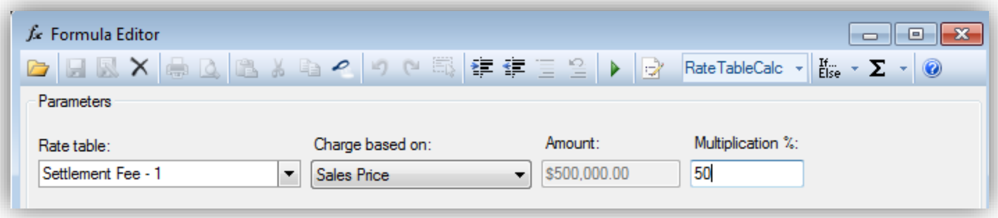
2. The **RateTableCalc** function allows the use of Premium rates that have been created in SPAdmin.
   This is most commonly used to calculate a fee that is based on a sliding scale, such as a Settlement or Escrow fee based on the Sales Price or Loan Amount that goes up in dollar amount increments.

   a. Press F8 in the field where you want to calculate an amount using a Rate table.
   b. Change the drop-down option from **FreeForm** to **RateTableCalc**:

c. Find the applicable Rate table in the **Rate table** drop-down.

d. Base the Charge on the **Sales Price** or select **Other** to enable the **Amount** field.

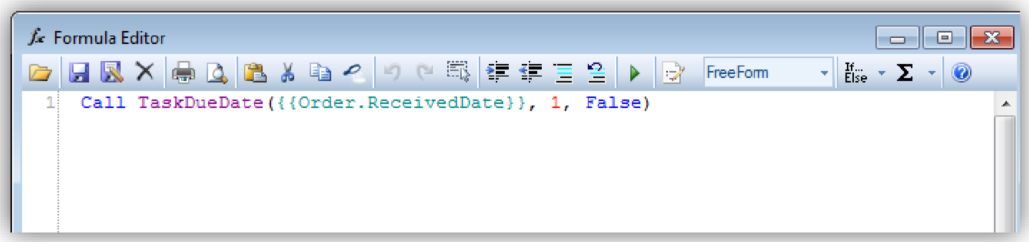
e. The **Multiplication %** defaults to 100.00% but can be changed to adjust the calculation.



Rate table: Settlement Fee - 1
Charge based on: Sales Price
Amount: $500,000.00
Multiplication %: 50

3. Viewing code for Preconfigured Formulas

   a. With a pre-defined formula entered, change back to **FreeForm**. The coding is now available to you and could be edited to include more logic.



```
Call TaskDueDate({{Order.ReceivedDate}}, 1, False)
```

Scripting Free Form Formulas

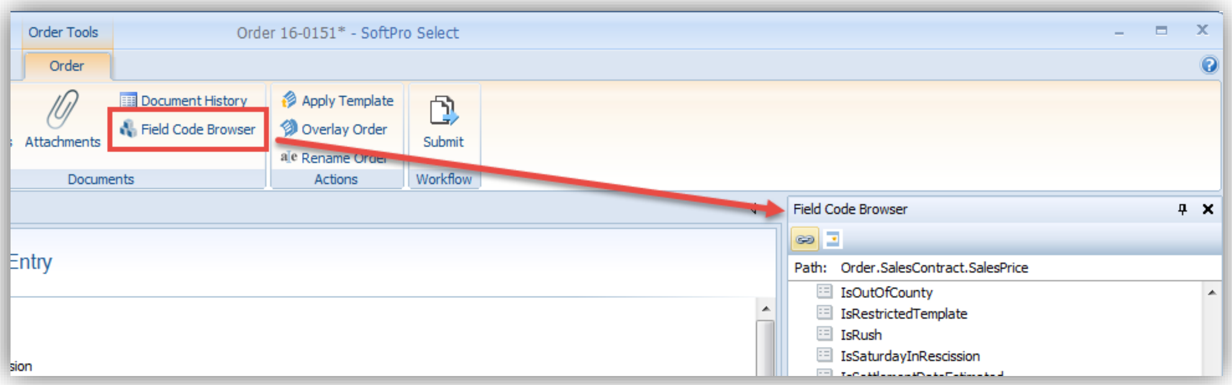
Scripting formulas is very much like programming. When you create a formula you are providing a series of instructions to SoftPro Select on how to evaluate data in order to create a value (result) for a field. The system will evaluate the data and populate the field with that value. Additionally, as input values change (i.e. field values that the formula depends on), the formula will re-run and update with the new data.

Using Field Codes

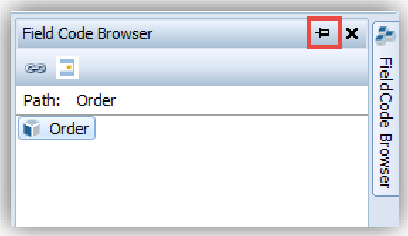
You may want to use other pieces of data in the order to determine the desired result for your formula. Knowing the field code names of these pieces of data will be necessary.

Field Code Browser

Use the **field code browser** to assist in locating field codes. A Field Code is the path from the order to the end field, indicating which objects to pass through. Paths can be relative or absolute. Field codes are represented in the "double-curly bracket" notation (i.e. {{Order.Title.Office}}). Open the browser with the **Field Code Browser** button on the Order toolbar:



- When using it frequently, Pin it to the screen to prevent the Browser from auto-hiding.

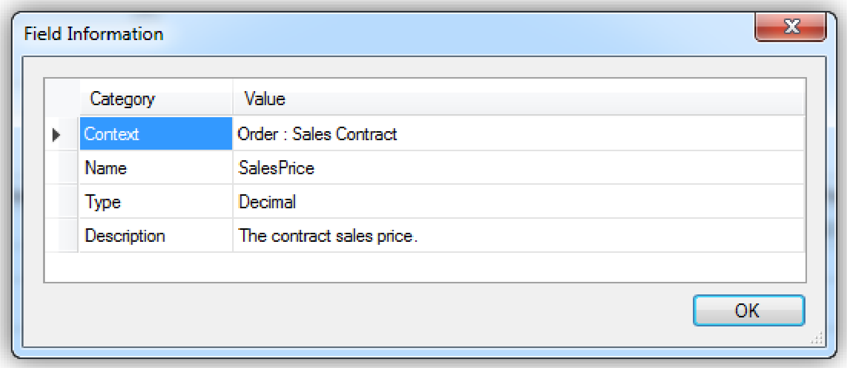


- Click the Track Current Field icon to display the field code for selected ProForm field.
- Click the Display custom fields icon if needed.
- When your cursor is in a field, the field code displays at the top of the Field Code Browser pane in the "Path:" field. You may copy and paste this text.

- Further information (Context, Type, & a brief description) regarding that field is shown in a pane below the field code tree.

Field Information

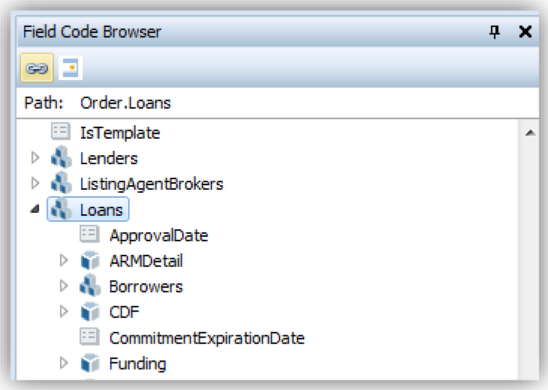
- Pressing **Alt** and **F6** opens the Field Information screen that provides the Context, Name of the field, Type of field and a Description.



Category	Value
Context	Order : Sales Contract
Name	SalesPrice
Type	Decimal
Description	The contract sales price.

Understanding Field Codes

- A **Business Object** is a unit of logic within an order. An example of a business object is loan. Business objects can contain fields, other business objects, and collections of business objects.



- **Absolute Field Codes** represent a full path to a piece of data. For example, this absolute path is is the field code for the order's Sales Price field:

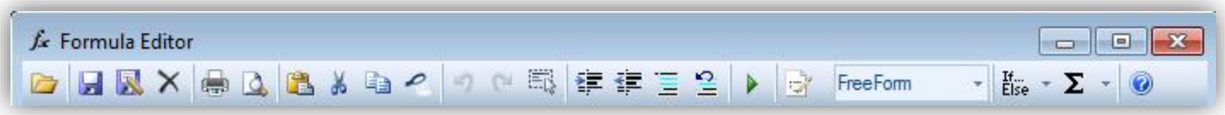
```
{{Order.SalesContract.SalesPrice}}
```

An absolute path might point to a field that involves multiple pieces of data. An order might contain multiple loans, properties, commitments, etc. If you are trying to point to one specific item on a field that contains multiples, you must include an item number by using the indexer syntax []:

- This is simply a pair of square brackets that contain a number.
- This number selects a specific item that the field code is referring to.
- Numbering begins with one. So:
  - [1] = first item
  - [2] = second item
  - [3] = third item
  - Etc.
- Two examples of absolute field codes that demonstrate this indexer syntax are as follows:
  `{{Order.Loans[1].Funding.LoanAmount}}`
  This refers to the *Principal amount of loan* field on the first loan in the order.
  `{{Order.Properties[2].County}}`
  This refers to the *County* field on the second property in the order.

- **Relative Field Codes** - Like absolute field codes, relative field codes also represent a **path** to a piece or collection of data. For example, when used on a field with the Context of Property this relative path points to an order's county field:
  - `{{.County}}`
  - Note carefully that the path Order.Properties and the indexer(`[]`) discussed above are **NOT included**.
  - In order to properly utilize a relative field code, the field code should be understood in respect to its context. Essentially:
    - Relative paths are structured based upon *where you are* in your order. That is the context.
    - Absolute paths start at the very top level, or root, of your order and build from there. In the ProForm order, "Order" is the top level that all objects are on.
    - Comparing Relative and Absolute field codes is similar to driving directions; relative paths are a "shorthand" of sorts, where you input your current location to get directions to a destination.
  - The relative code `{{.County}}` can be used in the Brief Escrow Legal field because both fields are on the Property object. When used on the first property, it will return the first property's county; when used on the second property, it will return the second property's county. If `{{.County}}` is used in a formula in a field with a context other than Property, and error will occur in the Formula Editor. To reference the first property's County field on a Context other than property, we would use `{{Order.Properties[1].County}}`

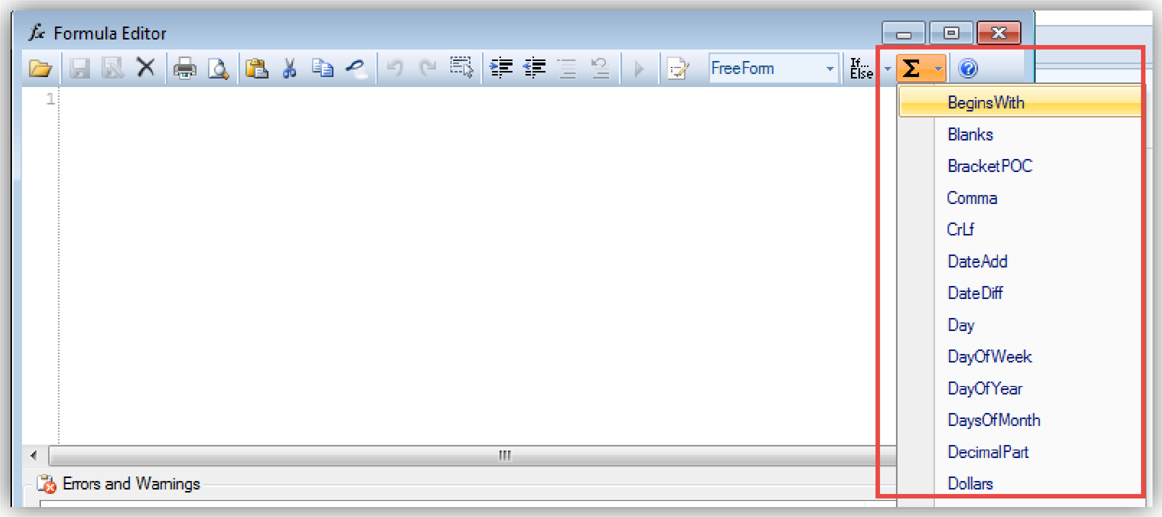
Toolbar



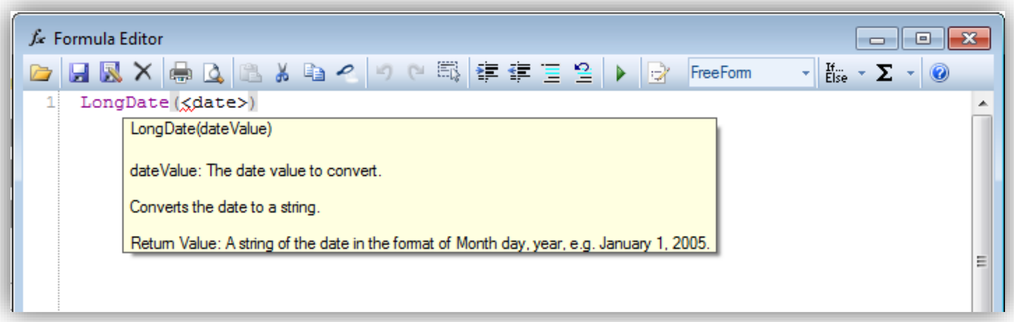
- **Load**: Opens a list of formulas saved to the database. Choose one from this list and it will appear in the Editor window.
- **Save**: Saves the current formula with the same name with which it was loaded (the formula must be loaded first).
- **Save As:** Saves the current formula as a new entry in the database.
- **Comment** and **Uncomment:** Highlight lines to Comment them "out" – Comments are ignored and can be used to explain the code below but can also be helpful when testing. Uncomment a section when you're ready to activate it again.
- **Command:** Displays a sub-menu of all available command constructs (such as If-EndIf, Case, and Iterate) that may be used in a free-form formula. These are helpful in giving you the structure to start a formula.
- **Function:** Displays a sub-menu of all available system functions that may be used in a free-form formula.

Formula Editor Functions

A list of available Functions is accessible from the Tool bar under the Σ icon.



Insert a Function and then hover over the purple function name to display more information about what each piece does, how to code the function and what the function will return:



```
LongDate (<date>)
```

LongDate(dateValue)

dateValue: The date value to convert.

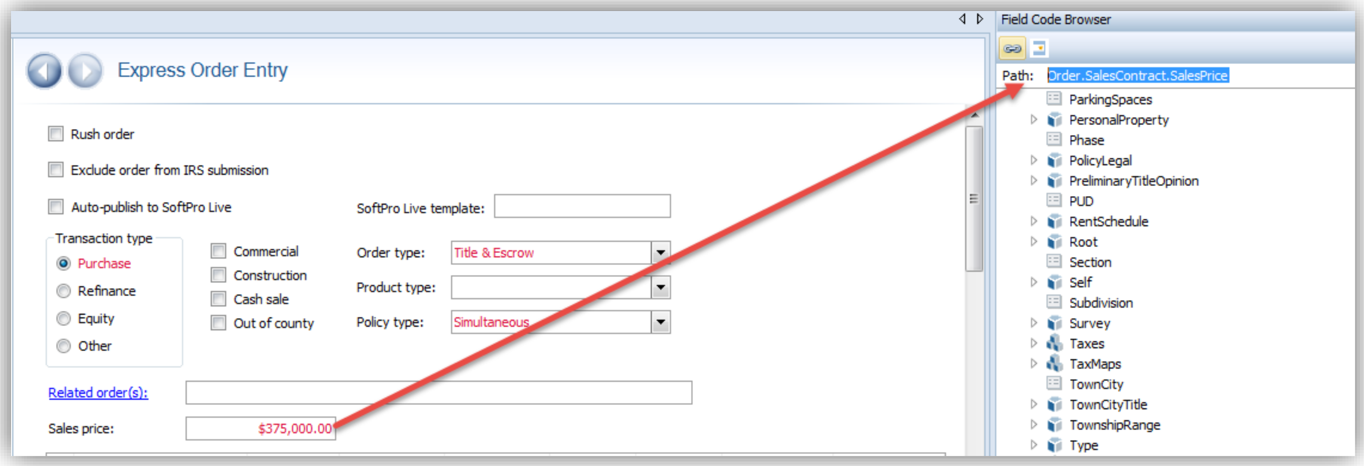
Converts the date to a string.

Return Value: A string of the date in the format of Month day, year, e.g. January 1, 2005.

Writing Free Form Formulas

The most basic way to use a formula is to reference a field code in the formula editor in another field to copy that data into the destination field. For example, if you wanted the Consideration Amount on the Deed screen to be the Sales Price we would place the field code `{{Order.SalesContract.SalesPrice}}` in the Formula Editor Interface on the Consideration Amount field.

1. To find the field code for the Sales Price field, place your cursor in the Sales Price field and then copy the field code path from the Field Code Browser:



Express Order Entry

- [ ] Rush order
- [ ] Exclude order from IRS submission
- [ ] Auto-publish to SoftPro Live

SoftPro Live template:

Transaction type: [✓ Purchase] Refinance Equity Other

- [ ] Commercial
- [ ] Construction
- [ ] Cash sale
- [ ] Out of county

Order type: Title & Escrow

Product type:

Policy type: Simultaneous

Related order(s):

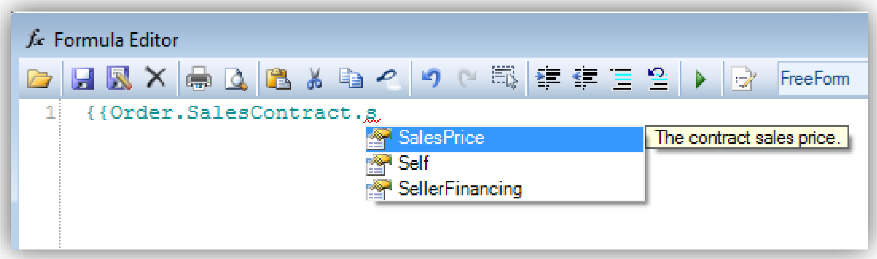
Sales price: $375,000.00

Field Code Browser

Path: Order.SalesContract.SalesPrice

ParkingSpaces
PersonalProperty
Phase
PolicyLegal
PreliminaryTitleOpinion
PUD
RentSchedule
Root
Section
Self
Subdivision
Survey
Taxes
TaxMaps
TownCity
TownCityTitle
TownshipRange
Type

Note: Another way to find a field code is to use the **Intellisense** function within the Formula editor module. When you begin to enter a field code, you will type {{. As you type, the most closely matching entry will come into view. If you select an object and press the period, the fields available to that new object are displayed. In this fashion, Intellisense guides you through the path to your intended value.



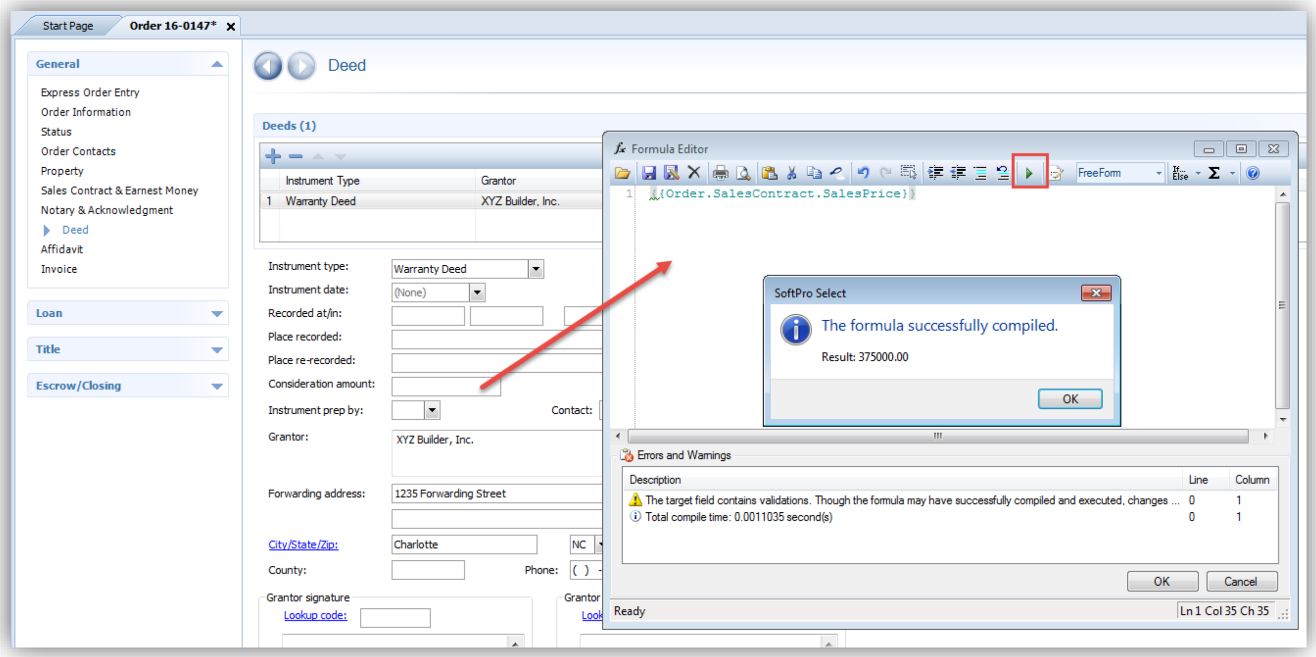
Formula Editor

FreeForm

```
{{Order.SalesContract.s
```

SalesPrice — The contract sales price.
Self
SellerFinancing

2. From the Consideration Amount field on the Deed Screen, press F8 to open the Formula Editor.



	Instrument Type	Grantor
1	Warranty Deed	XYZ Builder, Inc.

Instrument type: Warranty Deed
Instrument date: (None)
Recorded at/in:
Place recorded:
Place re-recorded:
Consideration amount:
Instrument prep by:
Contact:
Grantor: XYZ Builder, Inc.
Forwarding address: 1235 Forwarding Street
City/State/Zip: Charlotte NC
County:
Phone: ( )
Grantor signature
Lookup code:

```
{{Order.SalesContract.SalesPrice}}
```

SoftPro Select

The formula successfully compiled.

Result: 375000.00

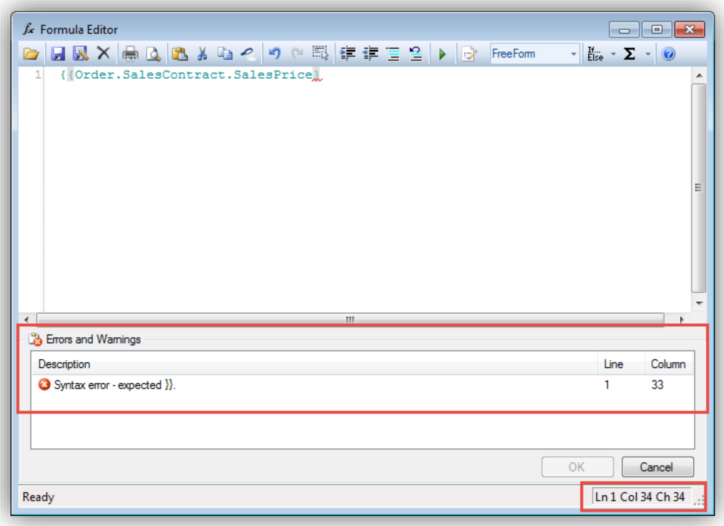
Errors and Warnings

Description	Line	Column
The target field contains validations. Though the formula may have successfully compiled and executed, changes ...	0	1
Total compile time: 0.0011035 second(s)	0	1

Ready
Ln 1 Col 35 Ch 35

3. The **Execute** button (or F5) will run the formula and a message will appear indicating if the formula successfully compiled (ran) or if there are any errors. The Errors will typically display before hitting the Execute button but testing the formula can be useful to see what result is given. The Execute button is helpful for testing more complex formulas.

4. When Errors are present, the Errors and Warnings panel will show you the Line and Column where errors were found. In the below formula, the final brace } is missing on Line 1, Column 33 as the Error indicates. The line and space where the cursor is located can be found in the bottom right-hand corner of the window:



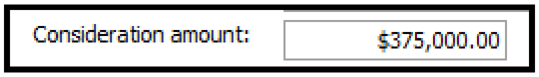
```
{{Order.SalesContract.SalesPrice}
```

Errors and Warnings

Description	Line	Column
Syntax error - expected }}.	1	33

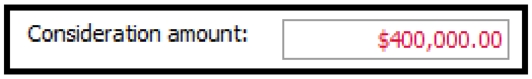
Ready   Ln 1 Col 34 Ch 34

5. Once the formula is working, click the OK button to exit the Formula Editor Interface. Data from a formula shows in black text; if overwritten by user-entered data, the user's change will show in red text.
   - Formula result shows in black text:



Consideration amount: $375,000.00

   - User-enter data has been entered over the formula and shows in red:



Consideration amount: $400,000.00

Hitting F2 will remove user-entered data and the formula result will reappear.

Basic String Formula

Stringing together Text and Field Codes. To hard code text, include it inside of quotation marks

- Include any spaces that should appear in the returned value inside the quotation marks
- Use the Plus sign + to add on to a String (for Numeric values, this is an Addition operator - see Appendix A for all Arithmetic Operators).

Project Name Formula Example

```
{{Order.Properties[1].Condo}} + ", Phase: " + {{Order.Properties[1].Phase}}
```

- Where "Aspen Pond" is entered in the 1st Property Condo field and "II" is entered in the 1st Property Phase field, the above formula results in an Email Subject Line of:
  Aspen Pond, Phase: II

Value

Value is the only variable that is available. It allows you to hold onto calculated and/or retrieved values, manipulate those values, and return the value. Value should be initialized. If this is not done, the results may be erratic and unexpected. It should be noted that Value is not required. The following formulas are equivalent:

```
Value = {{Order.SalesContract.SalesPrice}}
```

Or

```
{{Order.SalesContract.SalesPrice}}
```

If Value is not present, the system will return the last value that was pushed onto its internal evaluation stack. However, once Value is used, the value of Value will be returned regardless. Thus, assume the sales price is 100,000:

```
{{Order.SalesContract.SalesPrice}}
Value = 1
```

will return 1.

- Initialize Value or state what it will be at the beginning of the formula. This tells the system what your end results will be – usually either text or a number.
- Ways to Initialize the Variable:
  - `Value = ""`
    Indicates the result will be a string of text.

- `Value = 0.00`
  Indicates the result will be a number. If no conditions are true then the formula will return zero.
- `Value = "This is the default value the formula should return if no other conditions are met."`
- `Value = "This is the beginning of the text that will always be the same and we can add on to later in the formula"`

<u>The If-EndIf Statement</u>

Within the parentheses is an expression that must evaluate to either `True` or `False`. If the condition is `True`, all code statements between the `If` and the `EndIf` will be executed – otherwise, they'll be skipped. The conditional expression can be of any complexity as long as the resultant value is either `True` or `False`. Any number of statements can be placed within the confines of the `If-EndIf` statement, including other control statements. The following example demonstrates the use of the `If-EndIf` statement:

Endorsement Charge Formula Example

```
//Endorsement Calculation
Value = 0.00

//Charge $100 for Commercial Properties
If ( {{Order.IsCommercial}} ) Then
      Value = 100.00
EndIf
```

In this formula, the value is set to zero. If the Commercial checkbox is checked, the formula will return $100.00.

The If-Else-EndIf Statement

The `If-Else-EndIf` statement is very similar to the `If-EndIf` statement above, the difference is the Else block. This code is executed if the conditional expression evaluates to False. As before, any number or type of statements can be included in the Else block. The following example demonstrates the `If-Else-EndIf` statement:

Requirement Formula Example

```
Value = "Clear Title must be found for the current owner, whose name(s) are/is "

//If the Transaction Type is Purchase, use the Seller's name(s)
If ({{Order.TransactionType}} = "Purchase") Then
       Value = Value + {{Order.SellerData.NamesWithCommas}} + "."
//Otherwise, use the Buyer's name(s)
Else
       Value = Value + {{Order.BuyerData.NamesWithCommas}} + "."
EndIf

Value = Value + CrLf() + CrLf() + "Any outstanding Liens, Mortgages or other Debts
must be cleared prior to issuing the Final Policy."
```

- In this example, the first line of the desired result is always the same so the Value is initialized using the beginning of the text, "Clear Title must be found for the current owner, whose name are/is ".
- An `If-Else-EndIf` Command is used to determine whether to use the Seller's names (on a Purchase) or the Buyer's names (on all other Transaction types).
- "Value = Value +" is used to add on to the existing Value with additional data.
- The Crlf() Function inserts a carriage return/line feed in the string. The result would be:

Clear Title must be found for the current owner, whose name(s) are/is Robert T. Smith and Kathleen L. Smith.

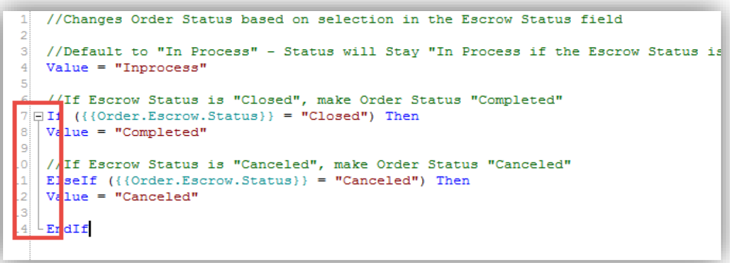
Any outstanding Liens, Mortgages or other Debts must be cleared prior to issuing the Final Policy.

Comments and White Space

Order Status Formula Example

```
//Changes Order Status based on selection in the Escrow Status field

//Default to "In Process" - Status will stay "In Process" if the Escrow Status is
blank, "Hold" or "In Process"
Value = "InProcess"

//If Escrow Status is "Closed", make Order Status "Completed"
If ({{Order.Escrow.Status}} = "Closed") Then
        Value = "Completed"

//If Escrow Status is "Canceled", make Order Status "Canceled"
ElseIf ({{Order.Escrow.Status}} = "Canceled") Then
        Value = "Canceled"

EndIf
```

- Two forward slashes “//” are used as a **Comment**. Lines beginning with the // are ignored by the system and are used for explaining the code.
- Use tabs for lining things up and making formulas easier to read
- Notice that **drop-down** options such as “In Process” are stored in the database without spaces. To evaluate them, type them without spaces ; i.e. “InProcess” (not case sensitive). TIP: put the field code in the formula editor and hit the Execute (or F5) to see how the system is storing a drop-down.
- In the Formula Editor Module, Commands such as If must have a closing EndIf. When used properly, this can be seen by the ⊟ indicator:



```
//Changes Order Status based on selection in the Escrow Status field

//Default to "In Process" - Status will Stay "In Process if the Escrow Status is
Value = "Inprocess"

//If Escrow Status is "Closed", make Order Status "Completed"
If ({{Order.Escrow.Status}} = "Closed") Then
Value = "Completed"

//If Escrow Status is "Canceled", make Order Status "Canceled"
ElseIf ({{Order.Escrow.Status}} = "Canceled") Then
Value = "Canceled"

EndIf
```

InString Example

Policy Transaction Code Example

```
// Transaction Code for Policy
Value = ""

// 3000-If the Policy Lookup Code contains the word "Enhanced" return Transaction code
1000
If (InString({{.Parent{TitleInsuranceCalculation}.PolicyLookupCode}}, "Enhanced") > 0 )
Then
      Value = "3000"

// 0225-If the Policy Lookup Code contains the word "Construction" return Transaction
code 0225
ElseIf (InString({{.Parent{TitleInsuranceCalculation}.PolicyLookupCode}},
"Construction") > 0 ) Then
      Value = "0225"

//1000-If neither of those words are found in the Policy code, use the Standard Loan
Policy Transaction code 1000
Else
      Value = "1000"
EndIf
```

- The **InString** function returns the space number where the text evaluated for begins. It will return -1 if the text is not found. We can test for text in the Policy code by seeing if the InString function returns something greater than 0.
- If the values returned by a formula do not exist in a drop-down field, they will not set the field's value. If the above formula is used in the Transaction Code field in a database that does not have these Transaction Codes, the formula will not load the values into the Transaction Code field.

Nesting functions

There may be multiple items that need to be evaluated in a formula. You may nest functions inside of other functions to evaluate multiple items.

Title Charge Example

```
// Fee based on Transaction type
If ({{Order.TransactionType}} = "Purchase") Then
       // If Cash Sale, charge $400
       If ({{Order.IsCashSale}}) Then
              Value = 400.00
       // Otherwise, $500 for Purchases
       Else
              Value = 500.00
       EndIf
// For Refinance & Equity charge $350.00
ElseIf ({{Order.TransactionType}} = "Refinance" or {{Order.TransactionType}} =
"Equity") Then
       Value = 350.00
// Otherwise (Other type), charge $250.00
Else
       Value = 250.00
EndIf
```

- This formula first evaluates if the Transaction Type is Purchase and if so, it looks to the nested `If-Else-EndIf` statement to further evaluate if the order has the Cash Sale boxed checked. If the box is checked (Cash Sale evaluates to True) the formula will return a value of $400.00. If it's a Purchase transaction but the Cash Sale box is unchecked, the Else condition will be returned and $500.00 is the formula result.
- The next `ElseIf` condition checks for transaction types of Refinance and Equity return $350.00.
- The final `Else` will return $250.00 when the Transaction Type is Other, or if SoftPro were to add any additional Transaction types in the future.
- Note the indented `If-Else-EndIf` statement to evaluate if Cash Sale is checked or not does not have to be tabbed in, but this is helpful in understanding the logic and adjusting the code later if/when things change.

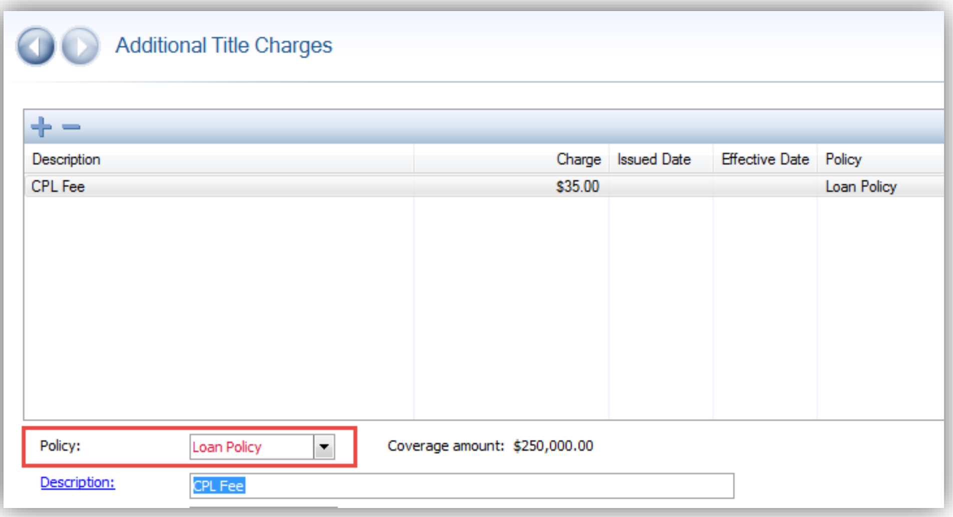
Case-EndCase Statement

If there are several conditions to evaluate, it may be more practical to use the `Case-EndCase` statement. The system will try to match the value with any of the exact values in the `CaseOf` blocks. If it finds a match, it will execute the statements related to that block and then exit the `Case` statement. If no matches are found, then the statements in the `AllOthers` block is executed. The `AllOthers` block is optional. If no matches are found and no `AllOthers` block is present, no code is executed. The following example illustrates the use of the `Case-EndCase` statement.

CPL Fee Formula Example

```
//CPL Fee Looks to the Underwriter selected for the Parent Policy on the
//Title Insurance Premium screen to determine the CPL Fee Charge.
Value = 0.00

//Uses the Underwriter’s Lookup Code
Case ({{.Parent{AdditionalTitleCharge}.Policy.Underwriter.LookupCode}})

//If the Underwriter is Chicago Title (Lookup Code "CT"), Charge $145
  CaseOf "CT":
              Value = 145.00
//If the Underwriter is First American (Lookup Code "FA"), Charge $135
  CaseOf "FA":
              Value = 135.00
//If the Underwriter is Fidelity (Lookup Code "FN"), Charge $135
  CaseOf "FN":
              Value = 135.00
//If the Underwriter is Old Republic (Lookup Code "OR"), Charge $130
  CaseOf "OR":
              Value = 130.00
//If the Underwriter is Stewart Title (Lookup Code "ST"), Charge $140
  CaseOf "ST":
              Value = 140.00
//For any other Underwriter, Charge $125
AllOthers:
              Value = 125.00

EndCase
```

Note in the above formula the Parent field code is used on the Case line. This means the formula will look at the Underwriter selected for the Policy that the CPL Fee is issued for. This requires the Policy drop-down field to be entered:


Additional Title Charges

Description	Charge	Issued Date	Effective Date	Policy
CPL Fee	$35.00			Loan Policy

Policy: Loan Policy
Coverage amount: $250,000.00
Description: CPL Fee

If it's unlikely that the Policy will be selected as shown above, the Case line can be replaced with

```
Case ( {{Order.Title.TitleInsuranceCalculations[1].Underwriter.LookupCode}} )
```

This references the Underwriter Lookup Code selected on the 1st Title Insurance calculation in the order.

Iterate Statement

Using the Iterate Statement is an ideal way to deal with multiples to grab a particular one or get all of the items from a multiple. The code will loop through each item in the **collection** with an index between the lower bound and the upper bound. The item can then be used within the statements to execute. A special operator of @ indicates the currently indexed object. `Iterate-From-To` statements can be nested within other `Iterate-From-To` statements. In instances like this, the @ operator must be appended the appropriate number of levels of nesting to get the desired object. The # operator is useful in this statement for determining the upper bound of the iteration. The lower bound must be 1 or greater but less than the number of items in the collection.

Invoice Screen – Bill to code Formula Example

```
// Loop through each other contact and contact type of other = ERecording
// Once found, break out of the loop. Set Code to that Other contact code
// If none exists - the code will be blank

Iterate {{Order.Others}} From 1 To #{{Order.Others}}
      If  (( @.OtherType) = "ERecording") Then
            Value = @.Code
      Break
      EndIf
Loop
```

- The above formula is looping through the collection of Other Contacts in the Order. We want to look at all possible Other contacts so the Iterate line uses `From 1 To #` to include the 1st Other contact up to the total number in the order. When you are only concerned with a certain number of items in a collection, you can change the # to the highest instance of the object to evaluate, but this is rare. An example would be when we are only concerned with the first 2 Loans in the order and would not consider any more than that. The Iterate line would look like:

  ```
  Iterate {{Order.Loans}} From 1 To 2{{Order.Loans}}
  ```

- Inside of the Iterate statement, use the @ to refer to objects that live on the collection. @.OtherType and @.Code are objects found under the Others context.
- Break is used to stop looping through a multiple. In this formula, we will find the first Other contact indicated with the Other Type “ERecording” and then stop searching through any remaining Other contacts. Placing the Break after we set the Value accomplishes this. Break is not required nor always wanted. See the next Iterate example where we are gathering all items from a collection that meet certain criteria.

Requirement - All Parcel ID's for all Properties Formula Example

```
//Parcels Requirement - Pulls Parcel ID Numbers
//for all Properties, all Parcel ID's - Labels Each Parcel ID
Value = "FOR INFORMATIONAL PURPOSES ONLY" + CrLf() + CrLf()

Iterate {{Order.Properties}} From 1 To #{{Order.Properties}}
      Iterate @.Parcels From 1 To #@.Parcels
            If ( Not IsEmpty(@.Identification) ) Then
                  Value = Value + "Parcel No. : "+ @.Identification + CrLf()
            EndIf
      Loop
Loop

Trim(Value)
```

- This formula contains two Iterate Statements nested within each other. The first is looking through all possible Properties in the Order. The second looks through all possible Parcel ID numbers on each Property. The If Statement ensures that the formula only returns Parcel ID numbers where data has been entered. `Value = Value +` is used so that when the code is read on the 2nd and subsequent loops, the original Value will be added to and not overwritten. The Trim function is used to remove the trailing Returns from the last Parcel ID found.
- In an Order that contains 3 total Parcel ID's the above formula returns:

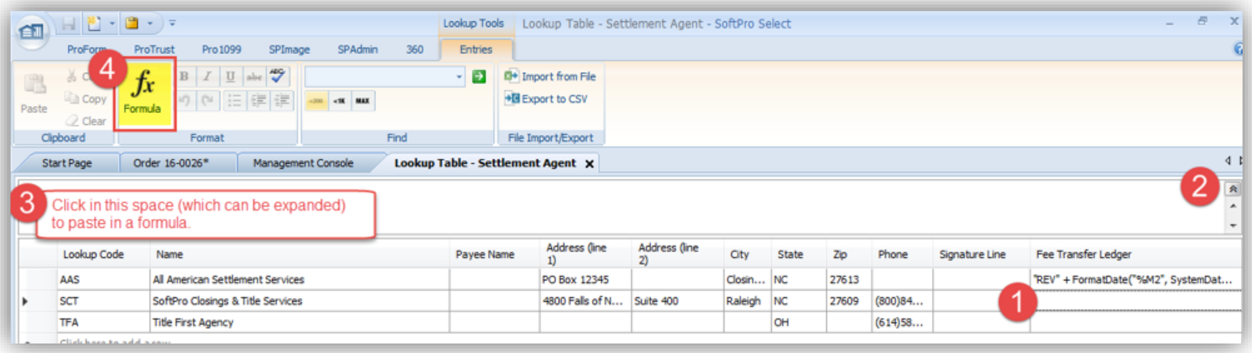
  FOR INFORMATIONAL PURPOSES ONLY

  Parcel No. : 789-456321
  Parcel No. : 235-643425
  Parcel No. : AP-39233

Adding Formulas to Lookup Tables

There are many times when it is beneficial to include formulas in a Lookup table entry. Formulas can be added directly in an Order using the same steps that would be taken to update a Lookup table entry. This is useful when a single or few changes need to be made. You may also find that editing in SPAdmin is most efficient in some cases.

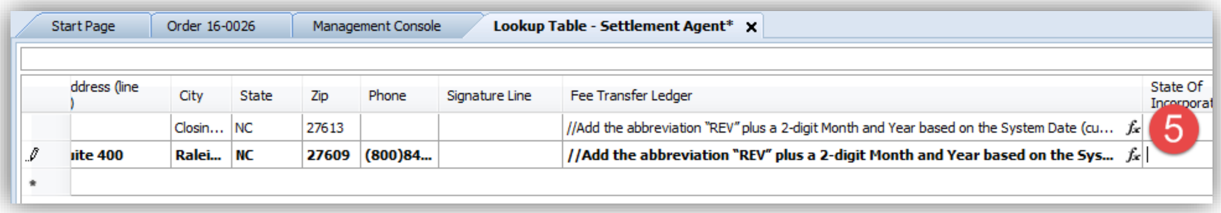
- From an order you can add a formula to the Formula editor, click OK to close the Formula Editor Interface and then Insert or Update the entry in the Lookup table. This will add/update the formula in the Lookup table.
- To add formulas to Lookup table entries from SPAdmin, follow these tips:
  1. Click in the field where you want to place a formula
  2. Expand the expandable Format Editor. It provides easier data entry for large-text fields. You can edit text for any selected cell by moving your cursor to the expandable Format editor. You can expand this field by clicking on the right drop-down arrow or by grabbing the bottom of the field with your cursor and dragging it down.
  3. Click in the white area to paste the formula in and scroll if necessary to ensure the entire formula is there.
  4. Click the Formula button *fx* Formula to make the Lookup table recognize the data as a formula.



Click in this space (which can be expanded) to paste in a formula.

Lookup Code	Name	Payee Name	Address (line 1)	Address (line 2)	City	State	Zip	Phone	Signature Line	Fee Transfer Ledger
AAS	All American Settlement Services		PO Box 12345		Closin...	NC	27613			"REV" + FormatDate("%M2", SystemDat...
SCT	SoftPro Closings & Title Services		4800 Falls of N...	Suite 400	Raleigh	NC	27609	(800)84...		
TFA	Title First Agency					OH		(614)58...		

  5. Click somewhere else in the Lookup table grid or Tab to the next cell so the system can recognize a change is there before saving. The line appears bold when a change has occurred but has yet to be saved. Additionally, the *fx* will show on the right side of the cell when it is recognized as a formula.



ddress (line)	City	State	Zip	Phone	Signature Line	Fee Transfer Ledger	State Of Incorporat
	Closin...	NC	27613			//Add the abbreviation "REV" plus a 2-digit Month and Year based on the System Date (cu... *fx*	
ıite 400	**Ralei...**	**NC**	**27609**	**(800)84...**		**//Add the abbreviation "REV" plus a 2-digit Month and Year based on the Sys...** *fx*	

Limitations

As with all things, there are limitations to what the Formula Editor can do. Known limitations include:

- Formulas only apply to the instance of a business object on which they are defined. For example, if you create a formula on a field of loan 1, that formula will only exist on loan 1. It will not exist on the same field of loan 2. If this behavior is needed, consider coding a Custom Order Rule instead of a Formula.
- There is no indicator displayed to alert you if a formula exists behind a field other than the result appearing in black text.
- Pre-defined formulas can only be cleared by switching to Free Form and pressing F2.
- Formulas are limited to using fields within the Order object to evaluate data. Formulas will not look at the Register or whether Documents have been printed/attached/rendered/emailed, etc.

Unsupported Practices

- Do not use the field where a formula is placed in your coding. For instance, do not reference the Sales Price field in the Sales Price field's formula editor.
- Do not use formulas that create infinite loops. For example, do not place a formula in the Escrow Legal Field copying the Commitment Legal field AND a formula in the Commitment Legal field referencing the Escrow Legal field.

Appendix A – Syntax Operators

Arithmetic Operators

The SoftPro Select Formula Editor contains numerous arithmetic operators that are common to all development environments. These operators are:

Operator	Precedence	Description
+	2	Addition operator. For string values, this operator appends two strings.
-	2	Subtraction operator when used with two operands; it is a negation operator when used with one operand.
*	1	Multiplication operator
/	1	Division operator
\	1	Integer division operator – This will return the integer portion of a division and drop the fractional portion (e.g. 4\3 = 1).
Mod	1	The modulo operator – This operator returns the remainder of a division operation (e.g. 26 Mod 10 = 6).

As in mathematics, the operators have precedence. All precedence 1 operators (multiplication-based operators) are evaluated left-to-right first followed by the precedence 2 operators evaluated left-to-right. For example, 1 + 2 – 3 would result in 0, whereas 1 + 2 * 3 would result in 7. This is because the 2 * 3 would be evaluated before the addition of the 1.

Logical Operators

Logical operators are those operators that result in value of True or False when evaluated. Operators that are familiar are:

Operator	Description
=	Determines if the left side is equal in value to the right side
<>	Determines if the left side is not equal in value to the value right side
>	Determines if the left side is greater in value than the right side
<	Determines if the left side is less in value than the right side
>=	Determines if the left side is greater than or equal in value than the right side
<=	Determines if the left side is less than or equal in value than the right side