Everything you DIDN'T know about IntellisenseX

Jim Nelson
The KONG Company
21 Los Vientos Drive
Newbury Park, CA 91320
Voice: (805) 498-9195
Fax: (805) 498-9195
JimRNelson@Gmail.com

IntellisenseX provides drop-down lists for PEMs from objects and field names from tables for a wide range of objects, tables, and data objects not addressed by FoxPro's native Intellisense. The drop-down lists show additional information in extra column(s) and can be filtered using "match-anywhere" (just a $) to reduce the number of items in the list.

We'll start by showing how IntellisenseX is installed, show how it works duplicating native Intellisense for PEMs of objects in the code window and command window, and then into the murky area where it does so for those cases where native Intellisense might be expected to work, but does nothing. Then we will move onto new ground—uses in WITH/ENDWITH blocks, references to variables not yet defined with LOCAL ... AS ..., uses in PRG-based classes, and objects from an object factory.

Our attention will then turn to how it handles fields from tables (something ignored by native Intellisense outside of the command window). We will show how tables can be referenced in code windows, including tables that are not yet open; how tables can be accessed by their aliases when referenced in code (USE .... ALIAS ....); how to see the fields for aliases used in SELECT statements; and how to see fields from a SQL Server database for use in SELECT statements.

Finally, we will delve into the use of the Alias Dictionary, which allows you to create application-wide aliases for both objects and tables. This simple table provides some remarkable and unexpected capabilities.

IntellisenseX is highly customizable and descriptions of the various options and plug-ins used by IntellisenseX will be interwoven into the session. IntellisenseX is delivered by and integrated with Thor. While installation of Thor will not be addressed during the session, there will be help available outside the session for anybody who needs help installing or using it.

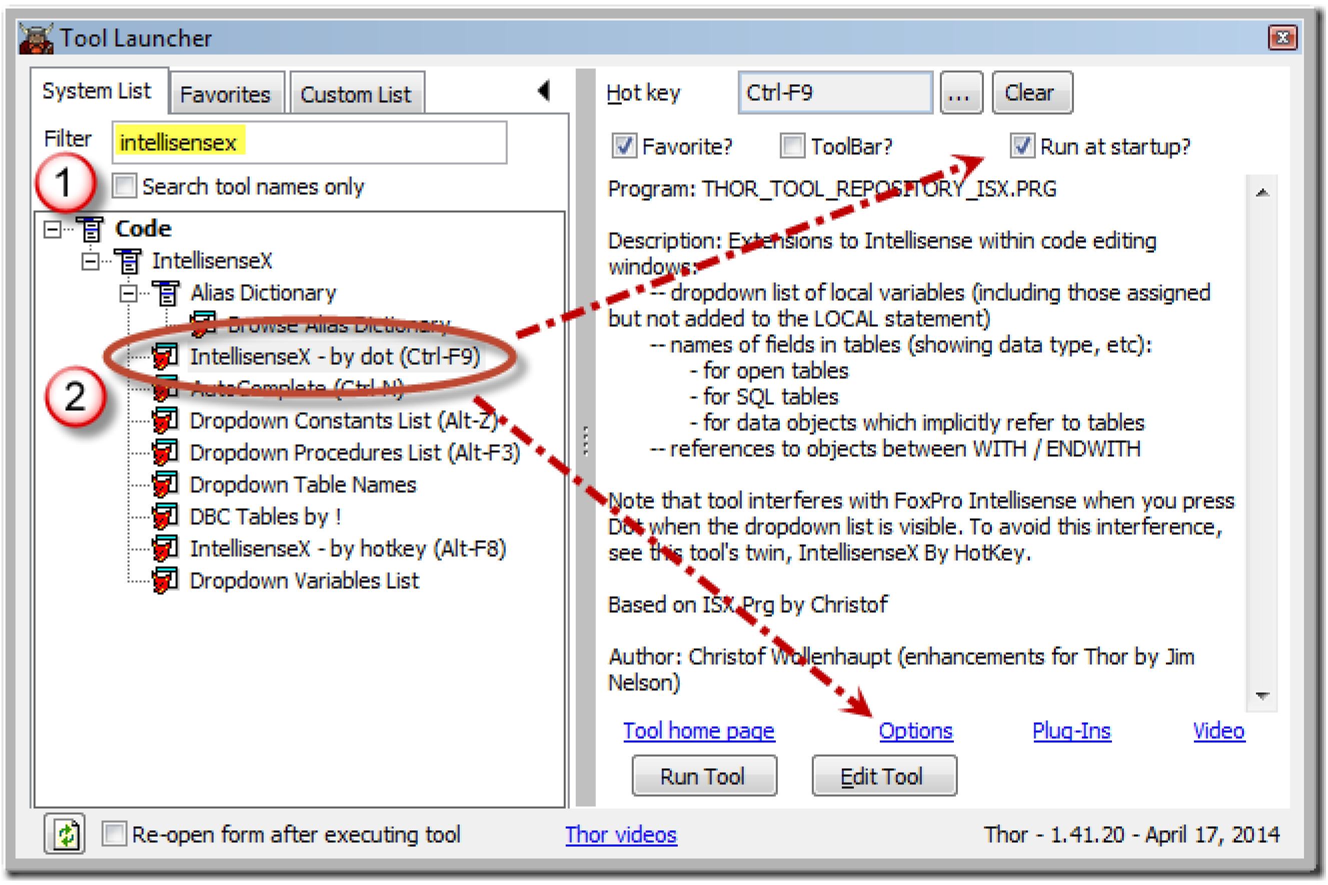
You will learn:

- *How to access PEMs from objects (in SCXs, VCXs, and PRGs) that native Intellisense forgot*
- *How to access lists of fields from native FoxPro tables, in SELECT statements for both native FoxPro tables and SQL Server tables, and from data objects*
- *How to create and use system-wide aliases for objects and tables*
- *How to use other related Thor tools to achieve auto-completion of names used in a procedure and to find #Define compiler constants*

Quick Start Guide

There are only a few steps to getting IntellisenseX up and running to provide a number of extensions to native FoxPro Intellisense. Once set up, you will find that using the dot (whether after object references or table references) produces the helpful dropdown list in a lot of different places that instantly feel natural. There are also some further customizations you can make, most quite easy to implement, that allow IntellisenseX to provide its assistance in a number of quite surprising ways.

To get started, open Tool Launcher, enter “IntellisenseX” in the filter box, and then click on the tool “IntellisenseX – by Dot” in the TreeView on the left.



Filter: intellisensex

Search tool names only

- Code
  - IntellisenseX
    - Alias Dictionary
      - Browse Alias Dictionary
    - IntellisenseX - by dot (Ctrl-F9)
    - AutoComplete (Ctrl-N)
    - Dropdown Constants List (Alt-Z)
    - Dropdown Procedures List (Alt-F3)
    - Dropdown Table Names
    - DBC Tables by !
    - IntellisenseX - by hotkey (Alt-F8)
    - Dropdown Variables List

Hot key: Ctrl-F9 ... Clear

☑ Favorite? ☐ ToolBar? ☑ Run at startup?

Program: THOR_TOOL_REPOSITORY_ISX.PRG

Description: Extensions to Intellisense within code editing windows:
-- dropdown list of local variables (including those assigned but not added to the LOCAL statement)
-- names of fields in tables (showing data type, etc):
- for open tables
- for SQL tables
- for data objects which implicitly refer to tables
-- references to objects between WITH / ENDWITH

Note that tool interferes with FoxPro Intellisense when you press Dot when the dropdown list is visible. To avoid this interference, see this tool's twin, IntellisenseX By HotKey.

Based on ISX.Prg by Christof

Author: Christof Wollenhaupt (enhancements for Thor by Jim Nelson)

☐ Re-open form after executing tool

Thor - 1.41.20 - April 17, 2014

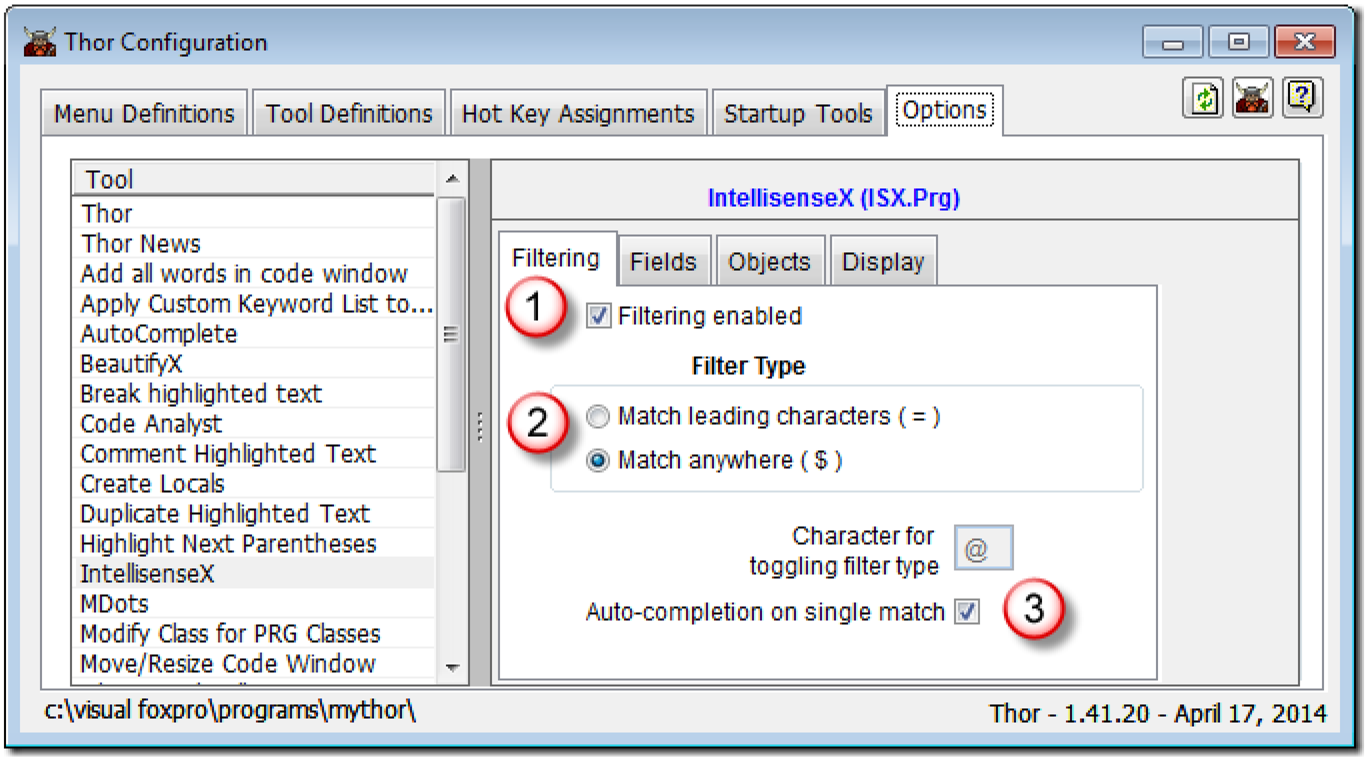
Check off "Run at startup". This will cause IntellisenseX to be enabled each time you start Thor in your IDE. (The tool "IntellisenseX – by Dot" is actually a toggle, enabling/disabling IntellisenseX; however the occasions where it was desirable to have it disabled have been mostly eradicated, so it can be turned on and left on.)

Configuration

Next, click on the Options link to open the IntellisenseX options page in the Thor Configuration form. You can also open the Configuration form directly, should you choose, to navigate to the options page for IntellisenseX.

Filtering

On the "Filtering" page, select the three options indicated below. The first two combine to form one of the sweet enhancements of IntellisenseX – the dropdown list is filtered as you type to show only matching entries and the matches can be anywhere in the each row, not just to the leading characters.


Thor Configuration

Menu Definitions | Tool Definitions | Hot Key Assignments | Startup Tools | Options

Tool
Thor
Thor News
Add all words in code window
Apply Custom Keyword List to...
AutoComplete
BeautifyX
Break highlighted text
Code Analyst
Comment Highlighted Text
Create Locals
Duplicate Highlighted Text
Highlight Next Parentheses
IntellisenseX
MDots
Modify Class for PRG Classes
Move/Resize Code Window

IntellisenseX (ISX.Prg)

Filtering | Fields | Objects | Display

1 [✓] Filtering enabled

Filter Type

2 ( ) Match leading characters (=)
(●) Match anywhere ($)

Character for toggling filter type: @

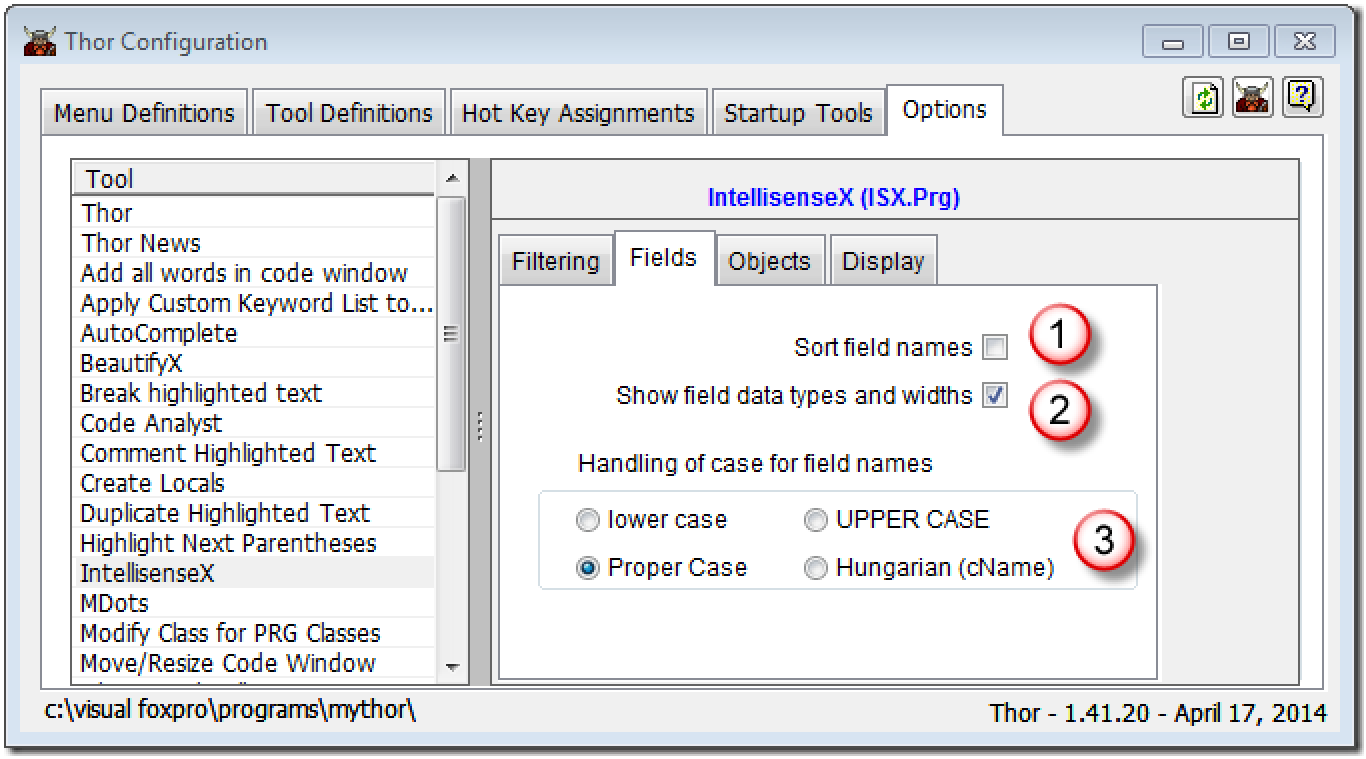
Auto-completion on single match [✓] 3

c:\visual foxpro\programs\mythor\

Thor - 1.41.20 - April 17, 2014

Fields

The “Fields” page controls how field names are displayed in drop down lists. I suggest setting the first two options as shown here and setting the third (“case for field names”) according to your own style. There is further customization available, to be addressed in the section on the Custom Keyword List.



Sort field names [ ]

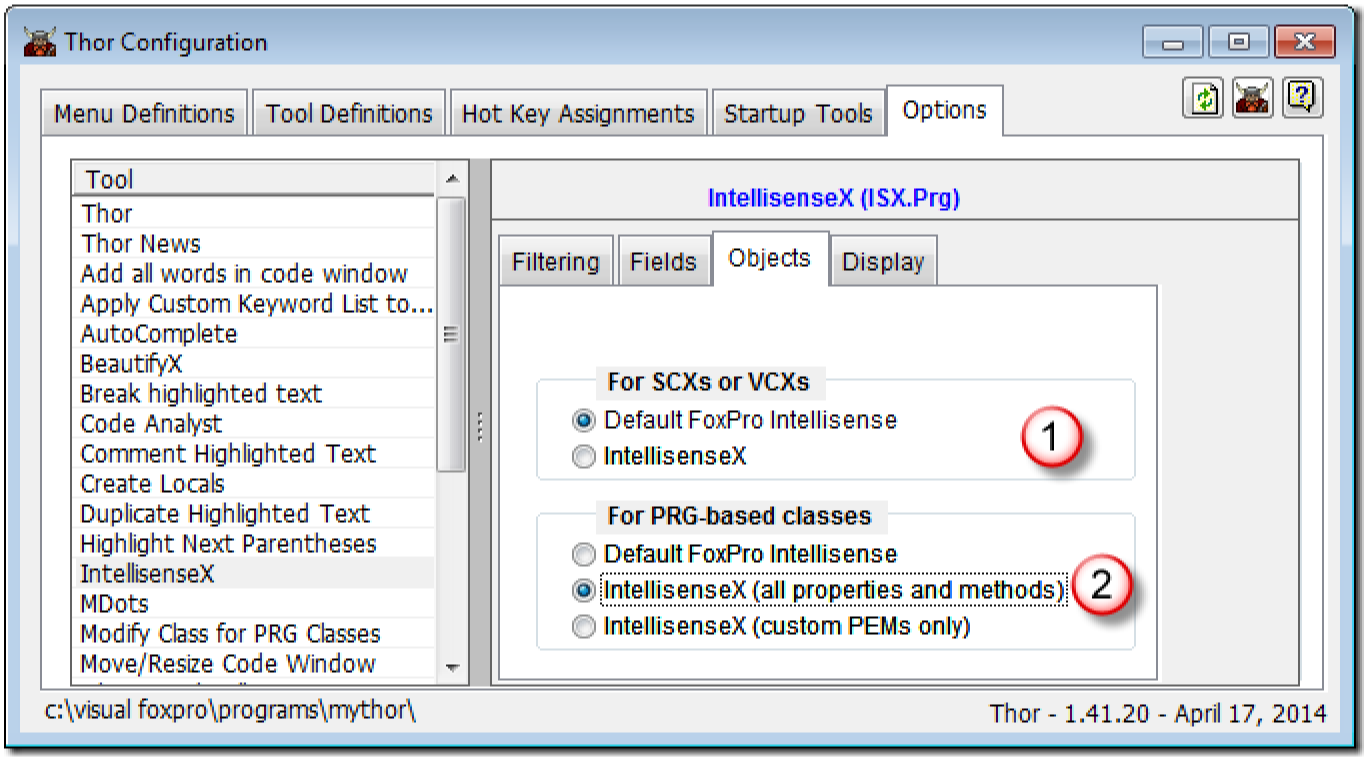
Show field data types and widths [✓]

Handling of case for field names

lower case ○ UPPER CASE ○ [✓ Proper Case] Hungarian (cName) ○

Objects

The third page ("**Objects**") determines when IntellisenseX displays the drop down lists for objects in a visual class (THISFORM or THIS) or PRG-based class. Originally, you may want to use the default FoxPro Intellisense for SCXs and VCXs. Without some additional configuration, IntellisenseX is sometimes quite slow for objects in SCXs and VCXs. However, once you have created your Custom Keyword List, you will want to select the second option (IntellisenseX).



For SCXs or VCXs
- [✓ Default FoxPro Intellisense]
- IntellisenseX

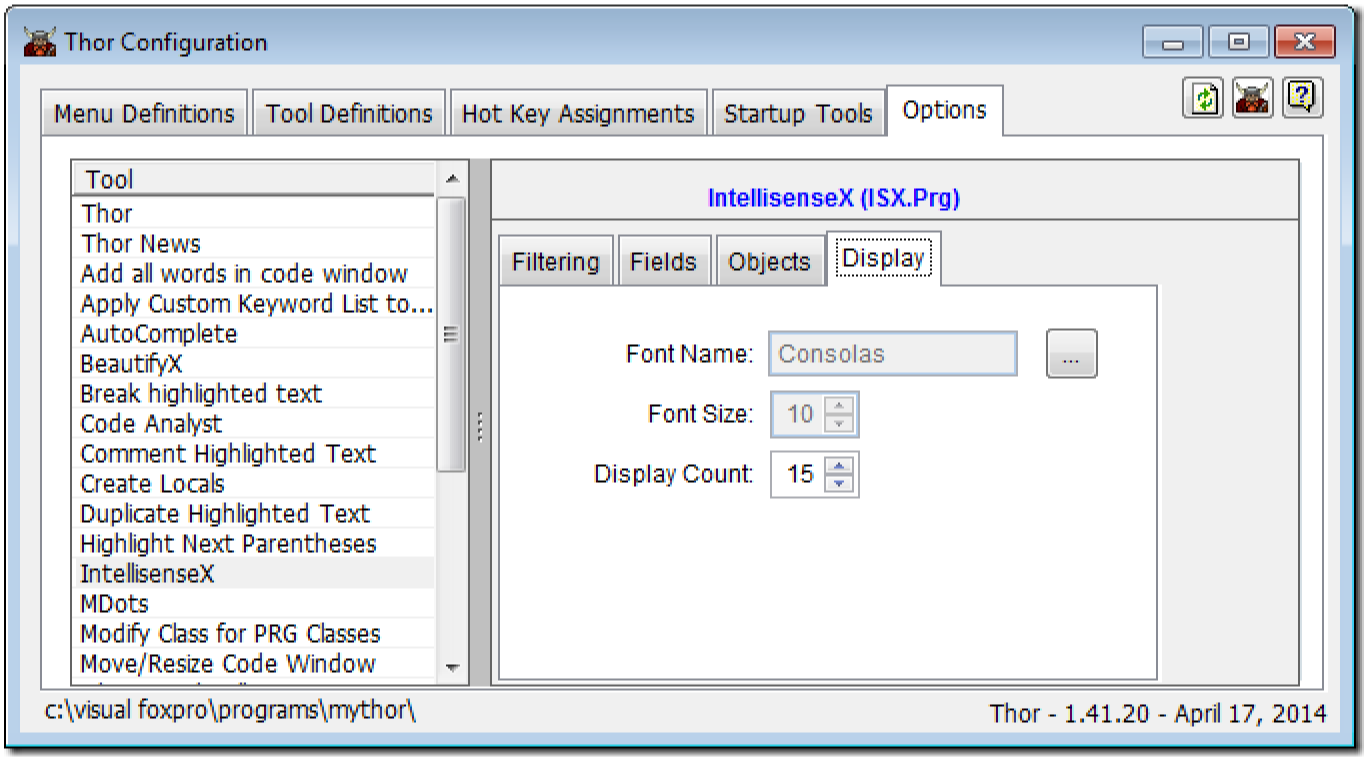
For PRG-based classes
- Default FoxPro Intellisense
- [✓ IntellisenseX (all properties and methods)]
- IntellisenseX (custom PEMs only)

c:\visual foxpro\programs\mythor\

Thor - 1.41.20 - April 17, 2014

Display

The “Display” page controls the display of the dropdown list used in IntellinsenseX. Be careful not to set the Display Count too large as you can get undesirable behavior if the dropdown list can’t fit either above or below the current screen position. This setting should not matter too much, however, as you become familiar with IntellisenseX, since the filtering (set on the first page) will rapidly reduce your list to only a few entries.



Font Name: Consolas

Font Size: 10

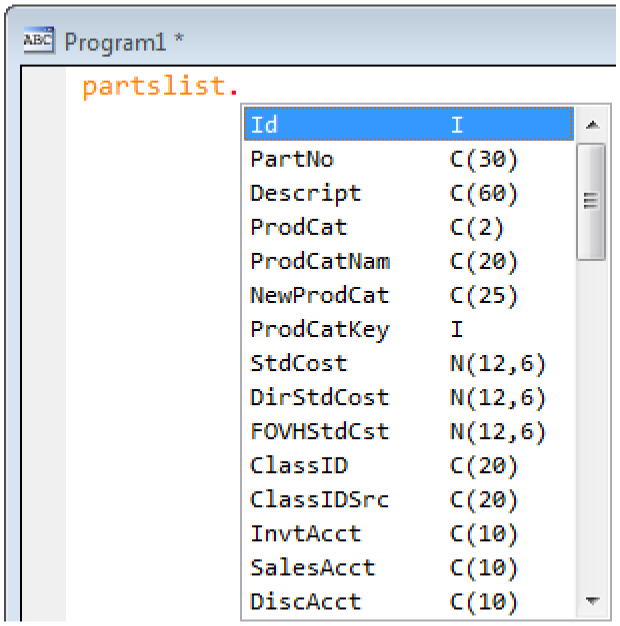
Display Count: 15

c:\visual foxpro\programs\mythor\

Thor - 1.41.20 - April 17, 2014

Default Behavior

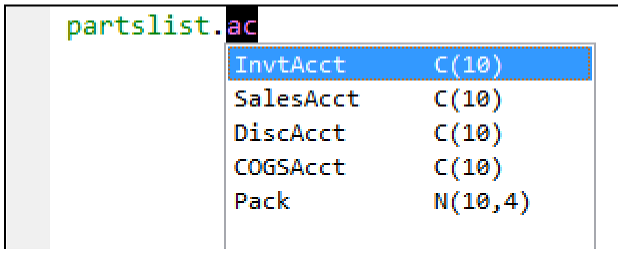
The behavior for IntellisenseX mimics, as much as possible, the behavior of FoxPro Intellisense when the press a dot; you are presented with a dropdown list of choices. The dropdown list will look somewhat different, though, as there will usually be extra columns in the display.



```
partslist.
Id          I
PartNo      C(30)
Descript    C(60)
ProdCat     C(2)
ProdCatNam  C(20)
NewProdCat  C(25)
ProdCatKey  I
StdCost     N(12,6)
DirStdCost  N(12,6)
FOVHStdCst  N(12,6)
ClassID     C(20)
ClassIDSrc  C(20)
InvtAcct    C(10)
SalesAcct   C(10)
DiscAcct    C(10)
```

You can select items from the list as you are accustomed, whether using the mouse to find an item or entering the beginning characters of an item, terminating your entry by pressing [Enter] or any non-alphabetic character.

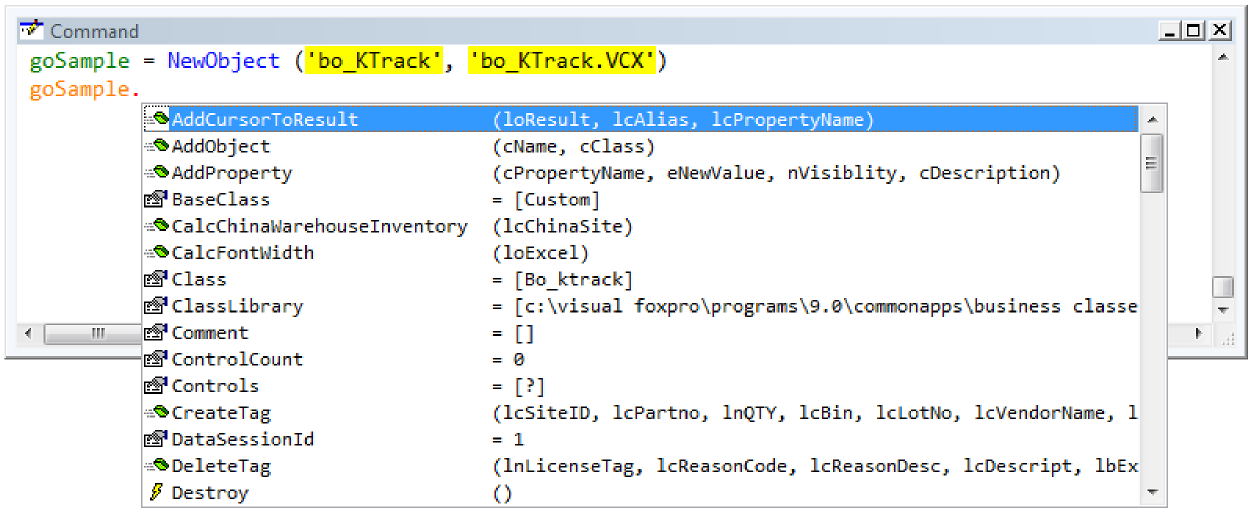
An extremely useful feature of IntellisenseX is that the characters you enter will match character strings anywhere in each item in your list, rather than just the leading characters. See Filtering. *Author's note: Take special note of this feature; it will seem completely natural and desirable in no time.*



```
partslist.ac
          InvtAcct     C(10)
          SalesAcct    C(10)
          DiscAcct     C(10)
          COGSAcct     C(10)
          Pack         N(10,4)
```

Objects and their PEMs

The dropdown list for objects shows the list of all properties and methods, as expected, along with a second column that shows the value for each properties and the parameter list for each method.



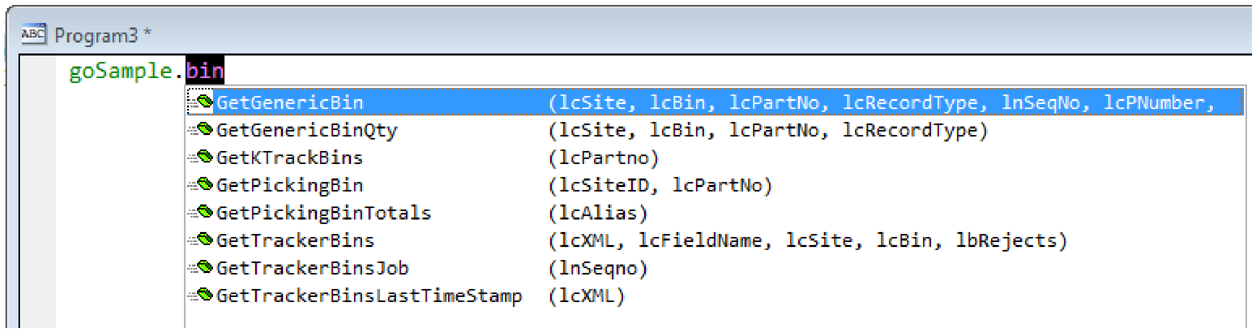
```
goSample = NewObject ('bo_KTrack', 'bo_KTrack.VCX')
goSample.
    AddCursorToResult             (loResult, lcAlias, lcPropertyName)
    AddObject                     (cName, cClass)
    AddProperty                   (cPropertyName, eNewValue, nVisiblity, cDescription)
    BaseClass                     = [Custom]
    CalcChinaWarehouseInventory   (lcChinaSite)
    CalcFontWidth                 (loExcel)
    Class                         = [Bo_ktrack]
    ClassLibrary                  = [c:\visual foxpro\programs\9.0\commonapps\business classe
    Comment                       = []
    ControlCount                  = 0
    Controls                      = [?]
    CreateTag                     (lcSiteID, lcPartno, lnQTY, lcBin, lcLotNo, lcVendorName, l
    DataSessionId                 = 1
    DeleteTag                     (lnLicenseTag, lcReasonCode, lcReasonDesc, lcDescript, lbEx
    Destroy                       ()
```

The parameter list for methods is not provided merely for informational purposes. If you press [Ctrl-Enter] to select a method from the list, the parameter list is pasted into your code window after the method name. You don't need this, of course, if FoxPro provides you with the parameter list when you enter a left-paren after a method name, but that only happens if FoxPro would have provided a dropdown in the first place. For all those places where IntellisenseX provides a dropdown where FoxPro would not have, the only way to get the list of parameters is to use [Ctrl-Enter]

IntellisenseX provides dropdown lists for all cases where FoxPro would provide a dropdown list, but there are a number of new cases worth pointing out.

Referencing an object that already exists.

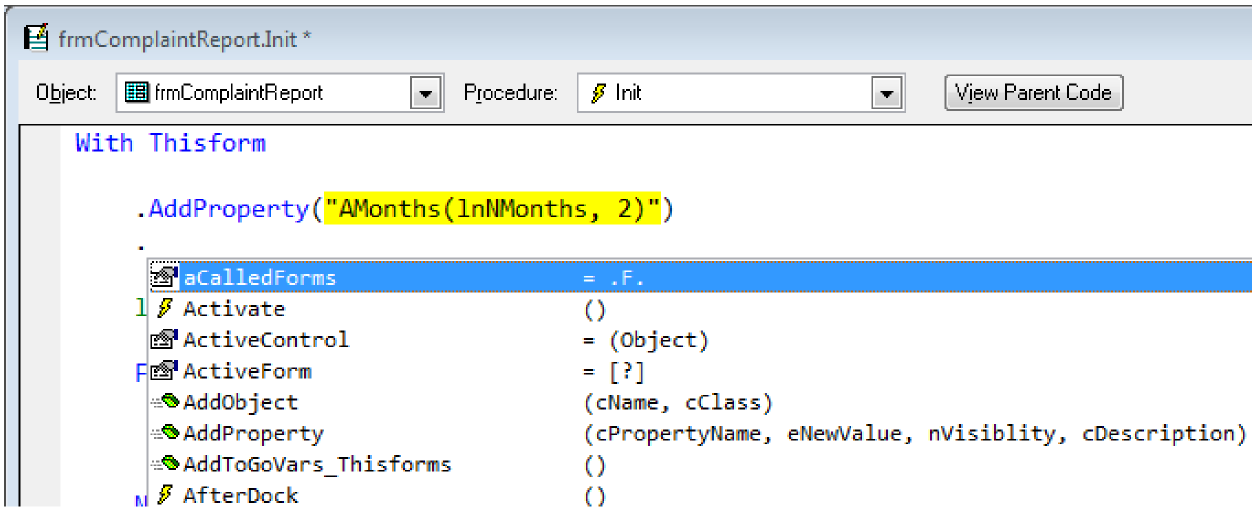
If an object already exists, it can be referenced in a PRG or method as well.


Program3 *

```
goSample.bin
GetGenericBin              (lcSite, lcBin, lcPartNo, lcRecordType, lnSeqNo, lcPNumber,
GetGenericBinQty           (lcSite, lcBin, lcPartNo, lcRecordType)
GetKTrackBins              (lcPartno)
GetPickingBin              (lcSiteID, lcPartNo)
GetPickingBinTotals        (lcAlias)
GetTrackerBins             (lcXML, lcFieldName, lcSite, lcBin, lbRejects)
GetTrackerBinsJob          (lnSeqno)
GetTrackerBinsLastTimeStamp (lcXML)
```

WITH ThisForm

Curiously, for FoxPro Intellisense, “With Thisform” works when editing code for objects contained in a form, but not when working on the form itself.


frmComplaintReport.Init *

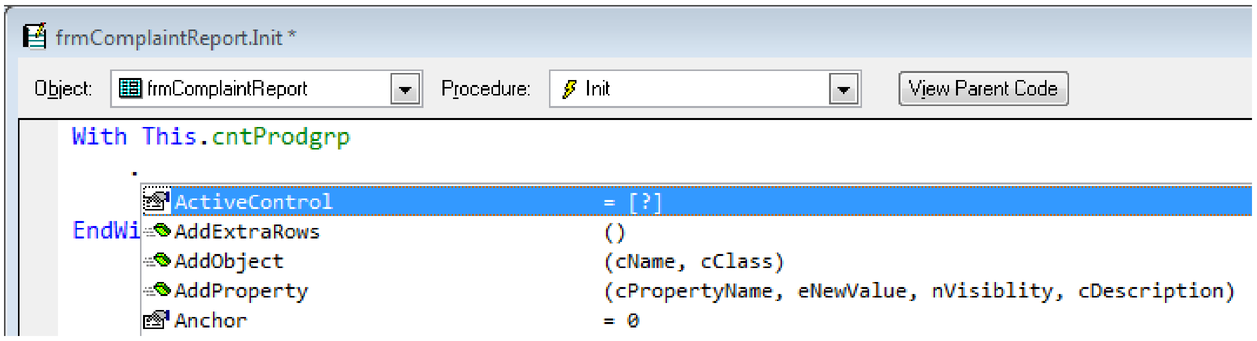
Object: frmComplaintReport   Procedure: Init   View Parent Code

```
With Thisform

    .AddProperty("AMonths(lnNMonths, 2)")
    .
    aCalledForms           = .F.
    Activate               ()
    ActiveControl          = (Object)
    ActiveForm             = [?]
    AddObject              (cName, cClass)
    AddProperty            (cPropertyName, eNewValue, nVisiblity, cDescription)
    AddToGoVars_Thisforms  ()
    AfterDock              ()
```

WITH This.Anything

FoxPro will only provide dropdowns if you have used “With Thisform” or “With This”; IntellisenseX allows references to any objects in the form or class.

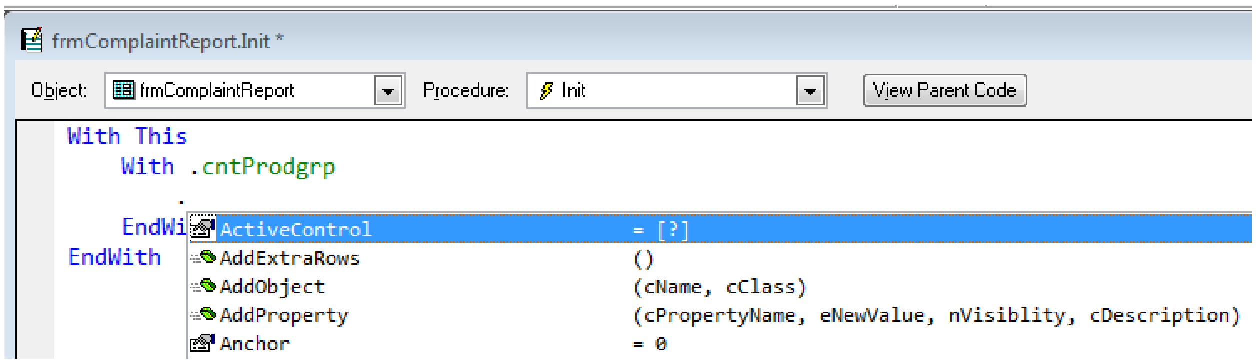

frmComplaintReport.Init *

Object: frmComplaintReport   Procedure: Init   View Parent Code

```
With This.cntProdgrp
    .
    ActiveControl          = [?]
EndWi AddExtraRows         ()
    AddObject              (cName, cClass)
    AddProperty            (cPropertyName, eNewValue, nVisiblity, cDescription)
    Anchor                 = 0
```

Embedded WITHs

IntellisenseX recognizes references inside embedded WITH statements.



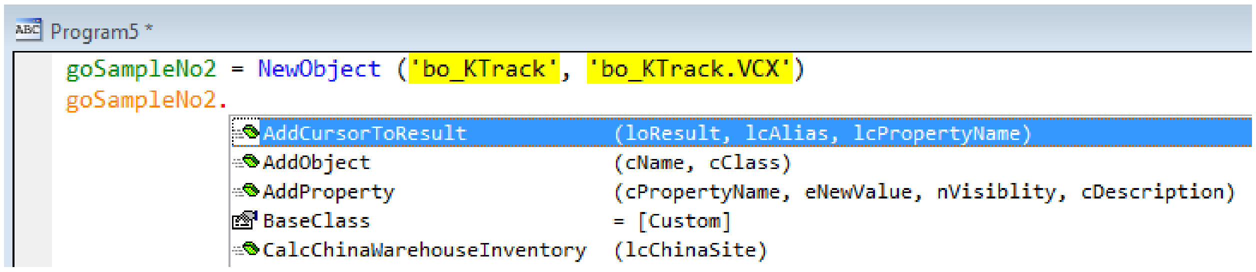
```
With This
    With .cntProdgrp
        .
    EndWi
EndWith
```

```
ActiveControl                = [?]
AddExtraRows                 ()
AddObject                    (cName, cClass)
AddProperty                  (cPropertyName, eNewValue, nVisiblity, cDescription)
Anchor                       = 0
```

Variables created by NewObject or CreateObject

IntellisenseX recognizes variables created by NewObject or CreateObject even if they are not yet defined in a LOCAL statement. See also the "New Object" Plug-In,

Author's note: This is useful for those like me who create their local variables late, using a Thor tool (BeautifyX) to do so when I save.

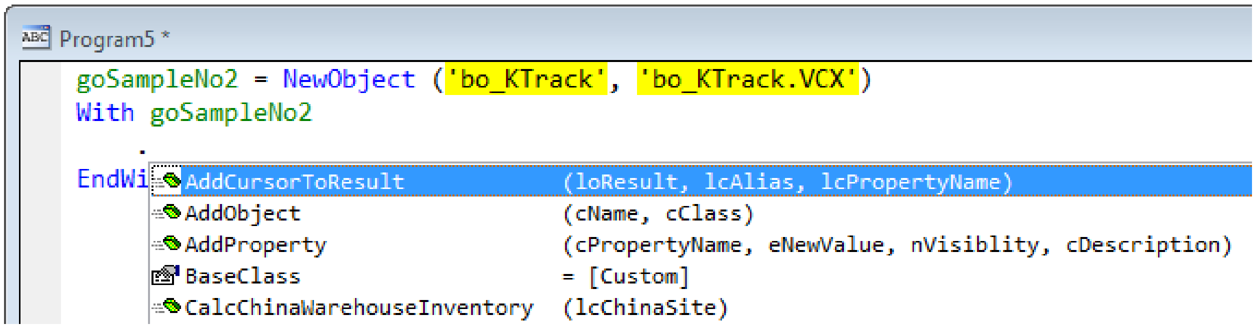


```
goSampleNo2 = NewObject ('bo_KTrack', 'bo_KTrack.VCX')
goSampleNo2.
```

```
AddCursorToResult             (loResult, lcAlias, lcPropertyName)
AddObject                     (cName, cClass)
AddProperty                   (cPropertyName, eNewValue, nVisiblity, cDescription)
BaseClass                     = [Custom]
CalcChinaWarehouseInventory   (lcChinaSite)
```

WITH a Variable created by NewObject or CreateObject

You can also use variables created by NewObject of CreateObject using WITH.

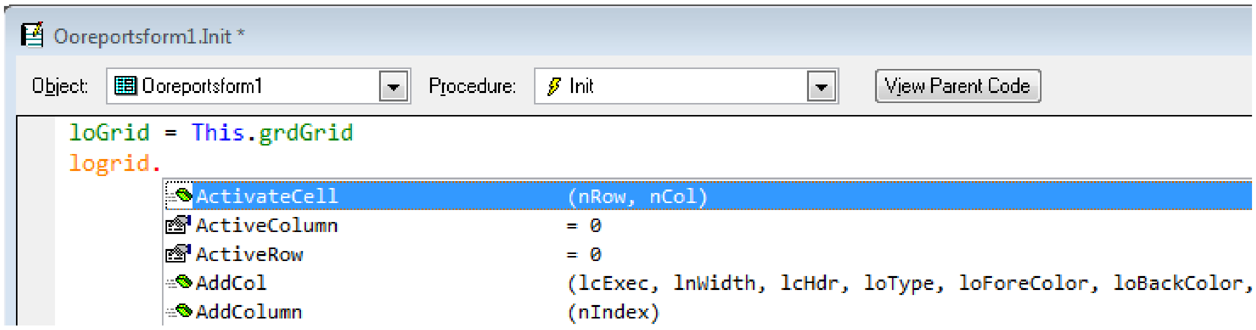


```
goSampleNo2 = NewObject ('bo_KTrack', 'bo_KTrack.VCX')
With goSampleNo2
    .
EndWi
```

```
AddCursorToResult             (loResult, lcAlias, lcPropertyName)
AddObject                     (cName, cClass)
AddProperty                   (cPropertyName, eNewValue, nVisiblity, cDescription)
BaseClass                     = [Custom]
CalcChinaWarehouseInventory   (lcChinaSite)
```

Indirect Object References

IntellisenseX recognizes the use of variables that are assigned as objects in code, as shown below. The objects referenced may be either objects in the form or class,



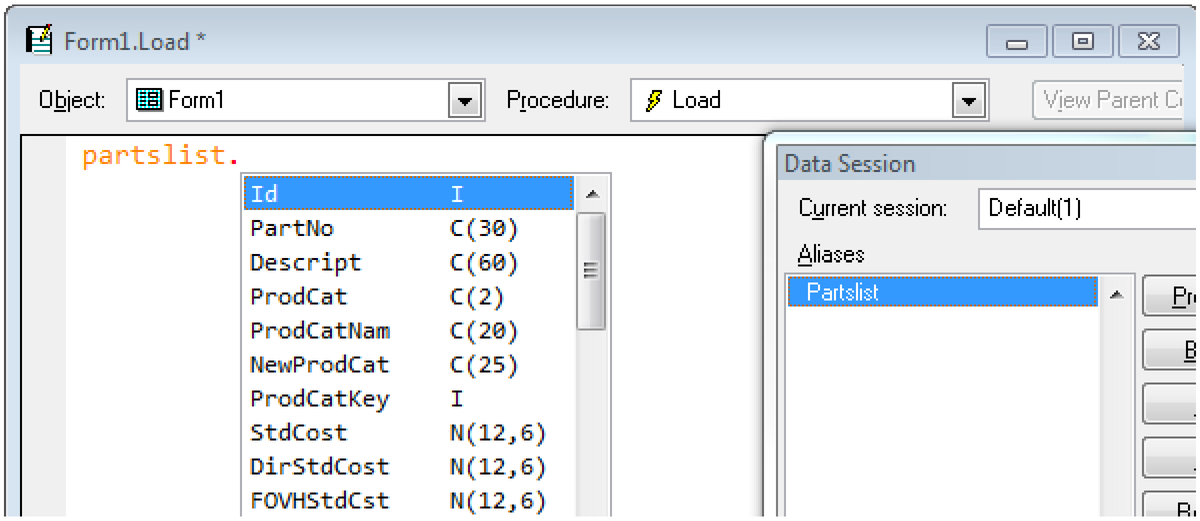
```
loGrid = This.grdGrid
logrid.
       ActivateCell        (nRow, nCol)
       ActiveColumn        = 0
       ActiveRow           = 0
       AddCol              (lcExec, lnWidth, lcHdr, loType, loForeColor, loBackColor,
       AddColumn           (nIndex)
```

Field Names

FoxPro does not provide dropdowns for fields when a table is referenced in a PRG or method code. IntellisenseX does so in a variety of cases, applicable not only to FoxPro tables but also to SQL Server tables, including SQL statements within TEXT/ENDTEXT blocks.

Tables already open

IntellisenseX provides a dropdown for any table open in the default data session.



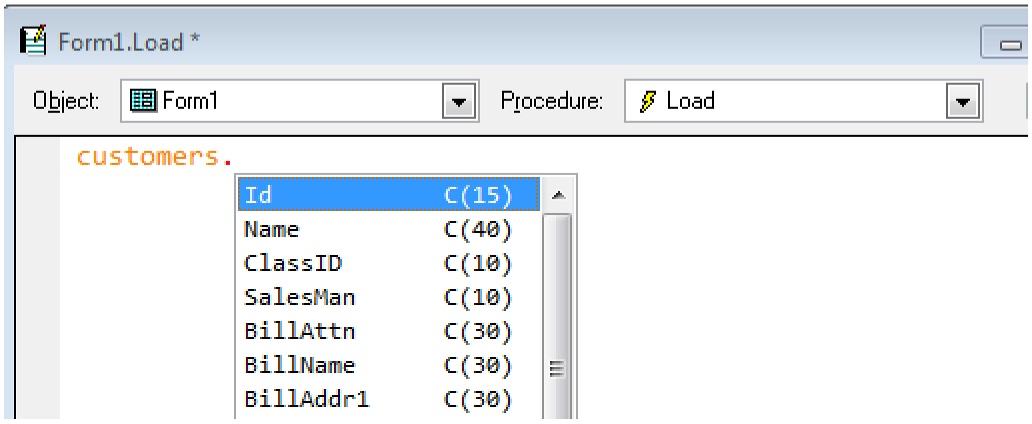
```
partslist.
    Id            I
    PartNo        C(30)
    Descript      C(60)
    ProdCat       C(2)
    ProdCatNam    C(20)
    NewProdCat    C(25)
    ProdCatKey    I
    StdCost       N(12,6)
    DirStdCost    N(12,6)
    FOVHStdCst    N(12,6)
```

Tables easily opened

IntellisenseX also provides a dropdown for a table, even if it is not already open, by looking for it in "all the usual places":

- in the current DBC, if any
- in the current folder or path
- in the data environment of the current form being edited
- in the MRU list for tables

Note: any table opened this way is closed after the dropdown disappears.

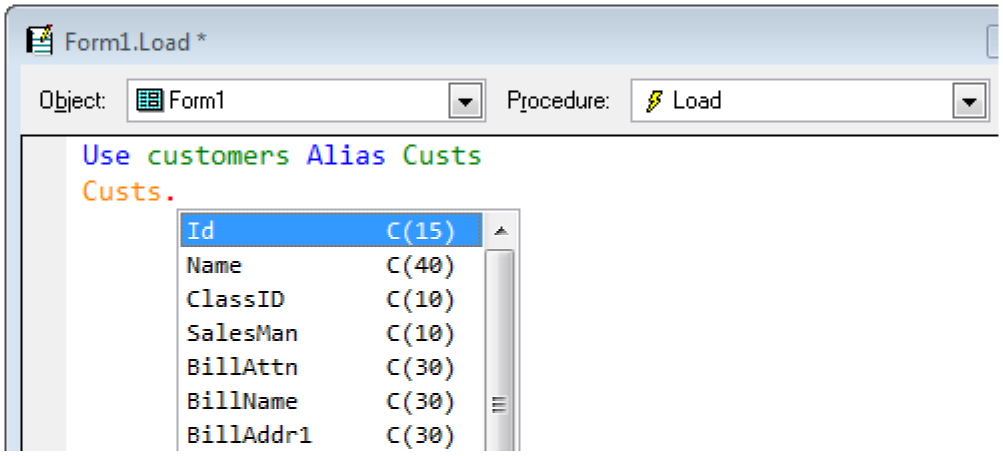


```
customers.
```

```
Id          C(15)
Name        C(40)
ClassID     C(10)
SalesMan    C(10)
BillAttn    C(30)
BillName    C(30)
BillAddr1   C(30)
```

Table Aliases

IntellisenseX recognizes references to tables opened with USE … ALIAS when the references is in the same method or procedure.

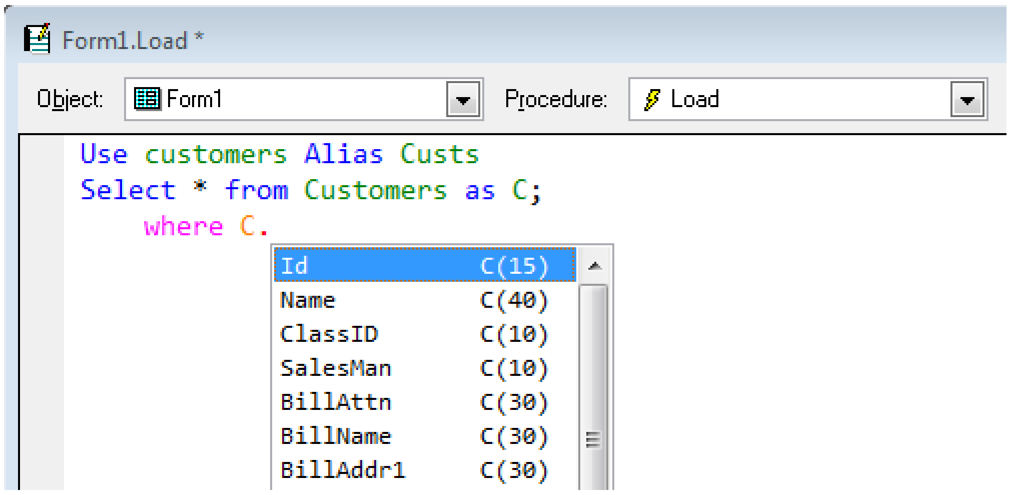


```
Use customers Alias Custs
Custs.
```

```
Id          C(15)
Name        C(40)
ClassID     C(10)
SalesMan    C(10)
BillAttn    C(30)
BillName    C(30)
BillAddr1   C(30)
```

Table Aliases in SQL-SELECT statements

IntellisenseX recognizes local aliases uses in SQL-Select Statements

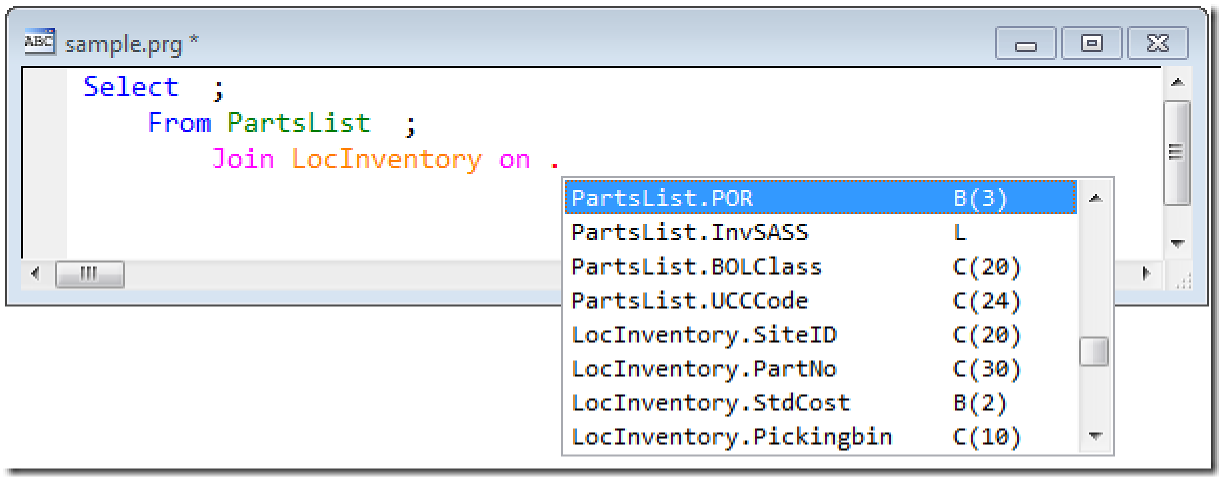


```
Use customers Alias Custs
Select * from Customers as C;
    where C.
```

```
Id          C(15)
Name        C(40)
ClassID     C(10)
SalesMan    C(10)
BillAttn    C(30)
BillName    C(30)
BillAddr1   C(30)
```

Fields from tables referenced in SQL-SELECT statements

IntellisenseX provides a dropdown list of all fields referenced in tables defined in an SQL Select statement; just press a dot with nothing before it.



```
Select  ;
    From PartsList  ;
        Join LocInventory on .
```

```
PartsList.POR               B(3)
PartsList.InvSASS           L
PartsList.BOLClass          C(20)
PartsList.UCCCode           C(24)
LocInventory.SiteID         C(20)
LocInventory.PartNo         C(30)
LocInventory.StdCost        B(2)
LocInventory.Pickingbin     C(10)
```

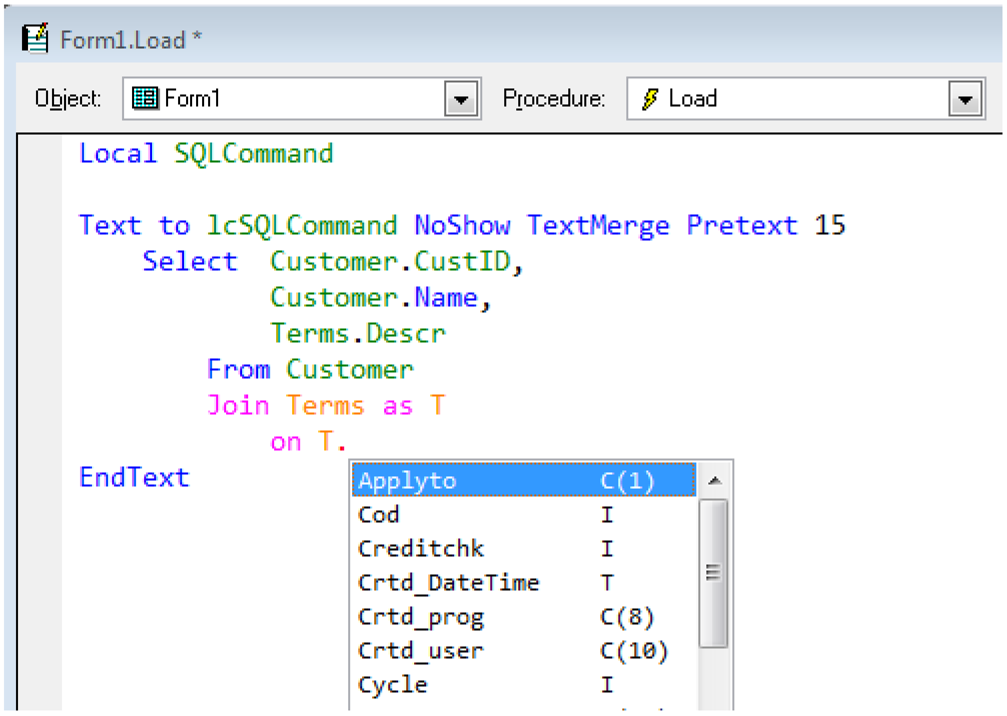
Writing inside-out SQL-SELECT statements

The preceding two examples present an interesting dilemma if you want to use them when creating the list of fields; if you are entering the statement in normal direct manner, you won't be able to use any of the local aliases or the bare dot, since you have not yet written the code yet that identifies the tables to be used.

You can get around this by writing your statements inside-out, in what will seem to be an unnatural order: enter "SELECT", then all the "FROM" and "JOIN" statements, before returning to identify the fields being selected.

SQL Server Tables

IntellisenseX now supplies field names from your SQL Server tables, as shown in the following example, which demonstrates a few of the features shown previously:



```
Local SQLCommand

Text to lcSQLCommand NoShow TextMerge Pretext 15
    Select  Customer.CustID,
            Customer.Name,
            Terms.Descr
        From Customer
        Join Terms as T
            on T.
EndText
```

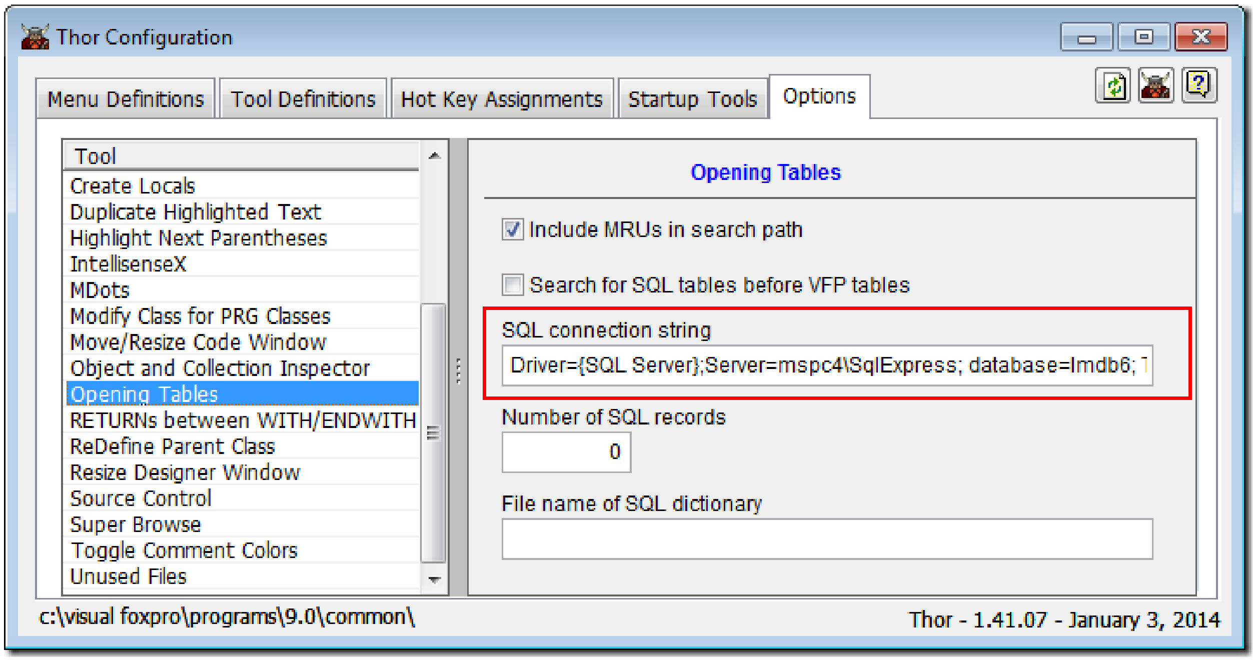
```
Applyto        C(1)
Cod            I
Creditchk      I
Crtd_DateTime  T
Crtd_prog      C(8)
Crtd_user      C(10)
Cycle          I
```

What you are seeing is a dropdown list of field names from an SQL table named “Terms”. It may occur to you that this looks exactly like the list from a VFP table. You would be completely correct, as IntellisenseX treats both FoxPro tables and SQL Tables the same way.

You need follow only a few steps to activate this:

- Open the Thor Configuration form
- Go to the Options page
- Click on “Opening Tables” on the left
- Enter your connection string on the right

See also Thor TWEeT #10: IntellisenseX: Field Names from SQL Server Tables



Tool
Create Locals
Duplicate Highlighted Text
Highlight Next Parentheses
IntellisenseX
MDots
Modify Class for PRG Classes
Move/Resize Code Window
Object and Collection Inspector
Opening Tables
RETURNs between WITH/ENDWITH
ReDefine Parent Class
Resize Designer Window
Source Control
Super Browse
Toggle Comment Colors
Unused Files

Opening Tables

[✓] Include MRUs in search path

[ ] Search for SQL tables before VFP tables

SQL connection string: Driver={SQL Server};Server=mspc4\SqlExpress; database=lmdb6;

Number of SQL records: 0

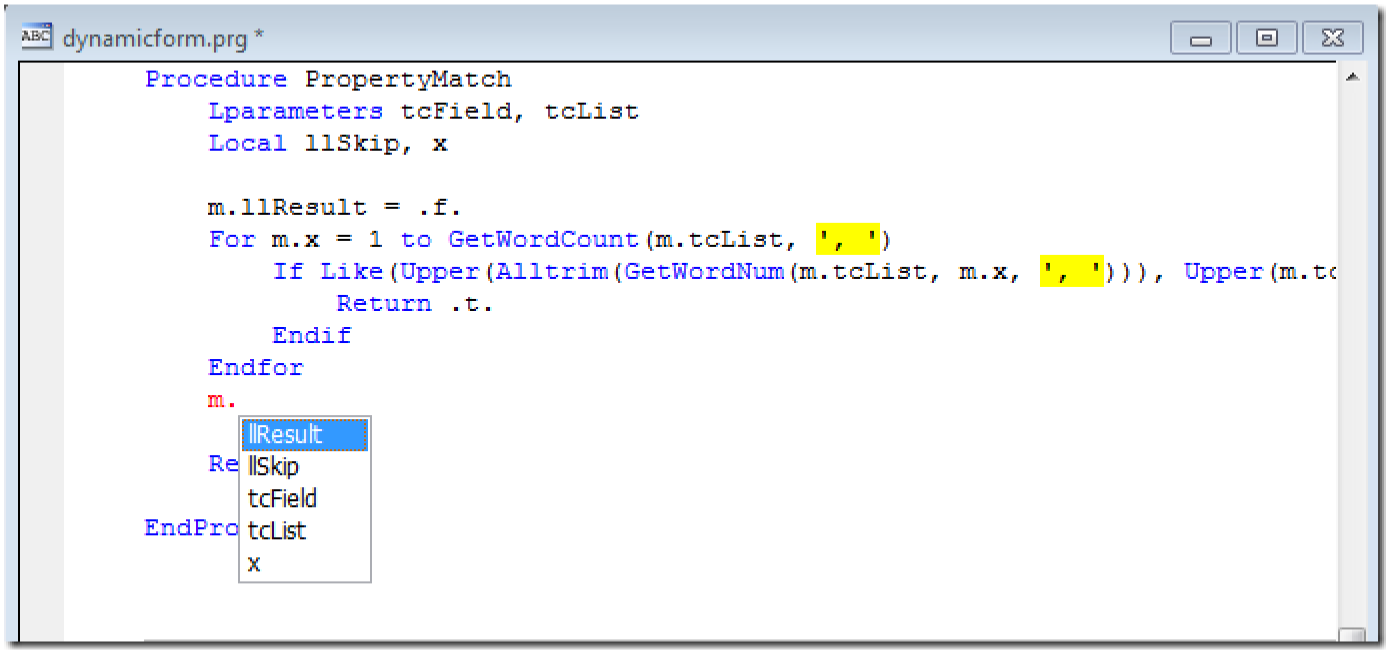
File name of SQL dictionary:

c:\visual foxpro\programs\9.0\common\

Thor - 1.41.07 - January 3, 2014

M-Dots

Typing "m" and then a dot (period) drops down a list of all variables in a procedure or method that are local, private, public, or are assigned values anywhere in the procedure , even after the current line of code. See also AutoComplete.



```
Procedure PropertyMatch
    Lparameters tcField, tcList
    Local llSkip, x

    m.llResult = .f.
    For m.x = 1 to GetWordCount(m.tcList, ', ')
        If Like(Upper(Alltrim(GetWordNum(m.tcList, m.x, ', '))), Upper(m.tc
            Return .t.
        Endif
    Endfor
    m.
       llResult
    Re llSkip
       tcField
EndPro tcList
       x
```

Customization

One of the measures of success in a project such as this is the number of suggestions for enhancements that it generates. Many of the features already described above were implemented based on suggestions from the community, but there have been many others as well. These suggestions were usually prefaced by phrases like "It might be a really crazy idea, but what would really work for me would be …"

- "I have a number of aliases I refer to all the time in my apps, but IntellisenseX is not showing dropdowns for them. How can I tell it that when I enter `Customer` I mean the table `D:\Data\Customer`?"
- "How do I get the dropdown of the properties and methods in my application object oApp?"
- "Would it be possible that I could tell it that `loJob` always refers to my business object for the Job table?"
- "And if you can do that, how about `loJob.oData`, which refers to the data object that contains all the fields from the Job table as properties?
- "I have a similar situation with the business object for my Vendor table, except that I typically reference it as a property of another object. Can you give me a dropdowns for the business object `This.oVendor` and its data object `This.oVendor.oData`?
- "Can I get dropdowns for chained objects like `This.oJob.oCustomer.oData`?

All of these questions can be answered by using either of the two types of customization available for IntellisenseX: aliases, which are static and are relatively easy to implement, and plug-ins, which are dynamic and provide much more power but also require more work to implement.

Aliases

The concept of generalizing the definitions of aliases came in response to a feature described earlier, Table Aliases and USE/ALIAS. The problem with this feature is that it only applies to the procedure that it is defined in, a use that is too limited in most cases.

IntellisenseX uses an unusually broad interpretation of an alias, which is normally understood to mean that "A" is to be substituted every time "B" is encountered. Essentially, any part of what you entered before a dot can be interpreted as an alias and what is to be substituted in its place can be either an object or a table.

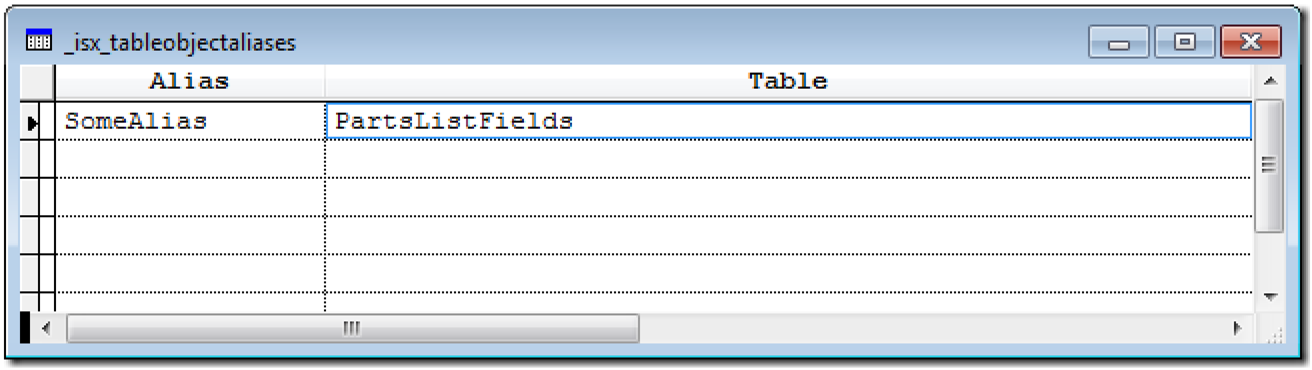
There are three different types of Aliases recognized by IntellisenseX:

- Global aliases (applicable everywhere)
- Form and object-related aliases (applicable only to the current form or class)
- Local aliases (applicable only to the current procedure.)

Global Aliases (the Alias Dictionary)

The *Alias Dictionary* is a table (maintained by Thor) that contains global definitions of tables and objects. These definitions are not relative to the current form, class, folder, project, application, or anything else.

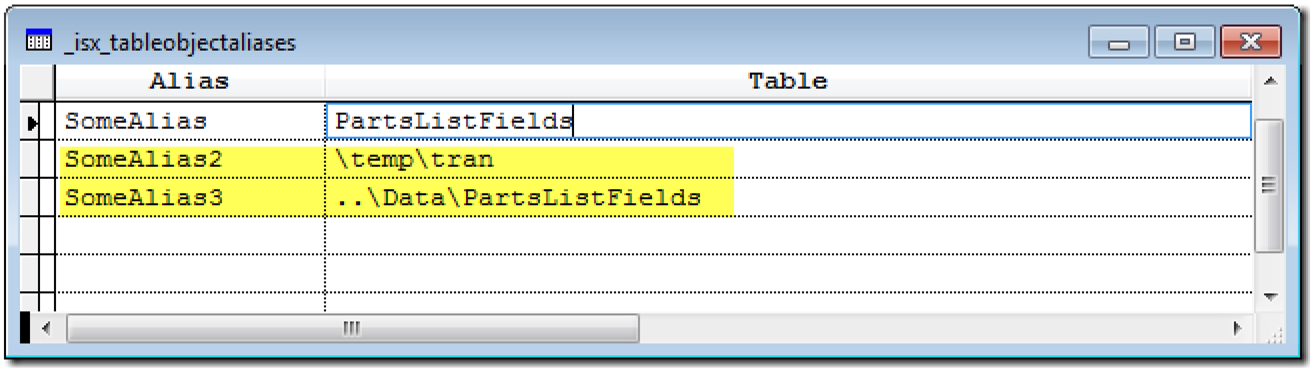
You can add records to this table by executing Thor tool *Browse Table/Object Alias List*, which does exactly that – it opens a browse window where you can add records or edit existing records, like this:



Alias	Table
SomeAlias	PartsListFields

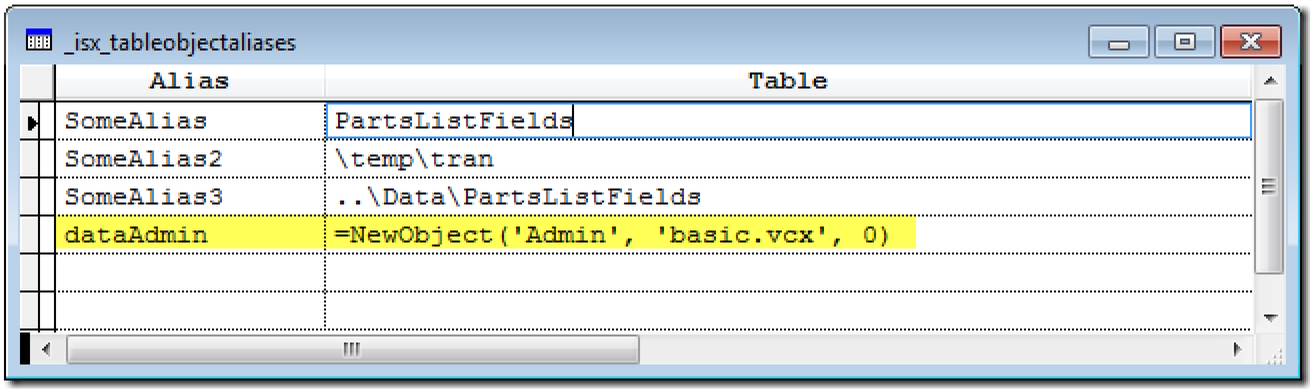
Henceforward, whenever you enter "SomeAlias" and invoke IntellisenseX as you always do, by pressing the dot, you will get the dropdown list for the table PartsListFields.

You can enter full path or relative paths for the table name.



Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields

As noted earlier, you can also add entries to this table that correspond to global objects. To do this, you can enter "=" followed by an executable expression in the Table field.



_isx_tableobjectaliases

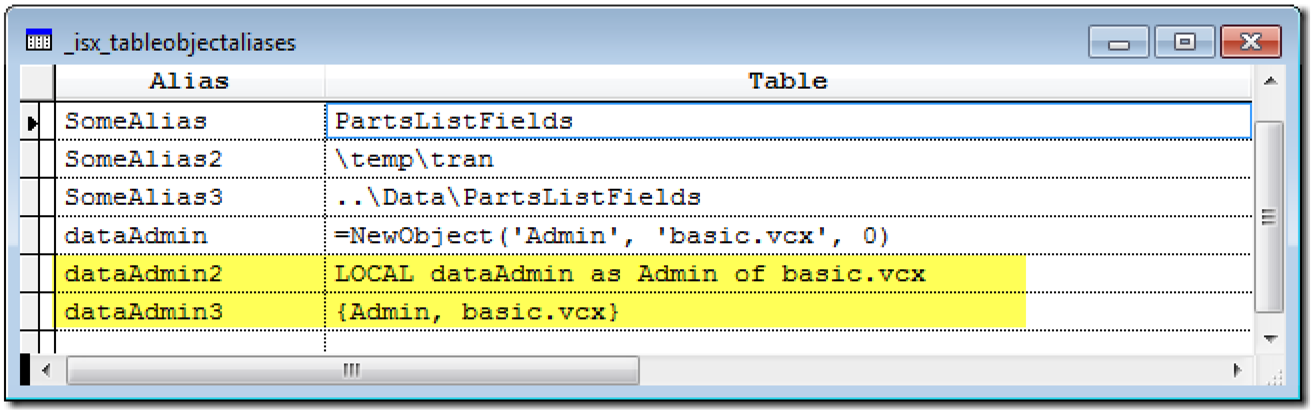
Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)

There are a few points of interest worth noting:

- NewObject() is used in this example, instead of CreateObject(), because NewObject() accepts a third parameter of zero so that the INIT method does not fire.
- If CreateObject() would in fact return the correct object, you can supply an empty (‘’) second parameter to NewObject().

There are two alternatives that can be used instead of calling NewObject():

- You can enclose the class name and class library name in curly braces, such as **{ClassName,ClassLib.VCX}.** The class library name is optional if CreateObject() would in fact return the correct object.
- You can copy a LOCAL statement and paste it directly into the table.



_isx_tableobjectaliases

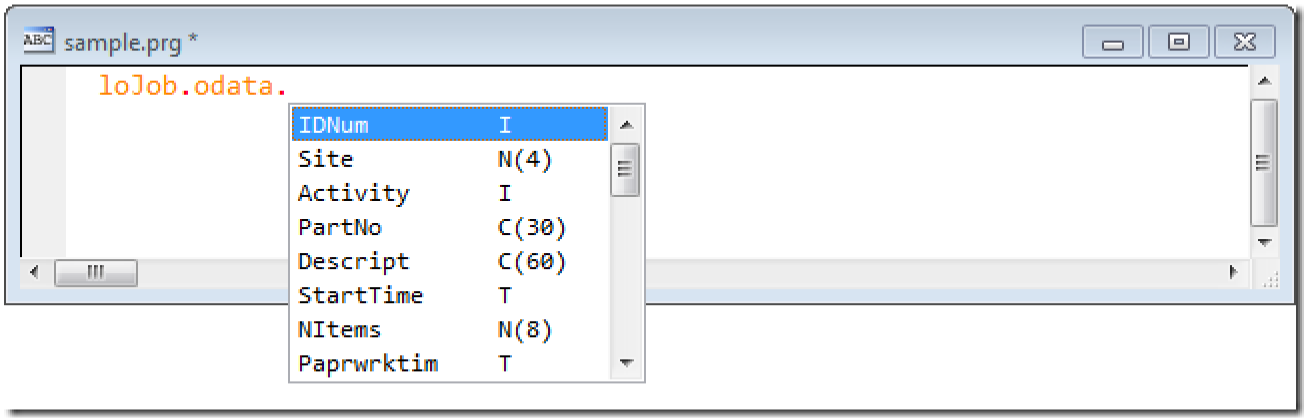
Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}

Actually, any executable expression (preceded by '”=”) will work, as long as the result is one of these:

- an object
- the alias of an open table
- the alias of a table that Thor can open

- an object containing an array named 'aList'. This array may have up to three columns and the contents of the array will populate the drop down list.

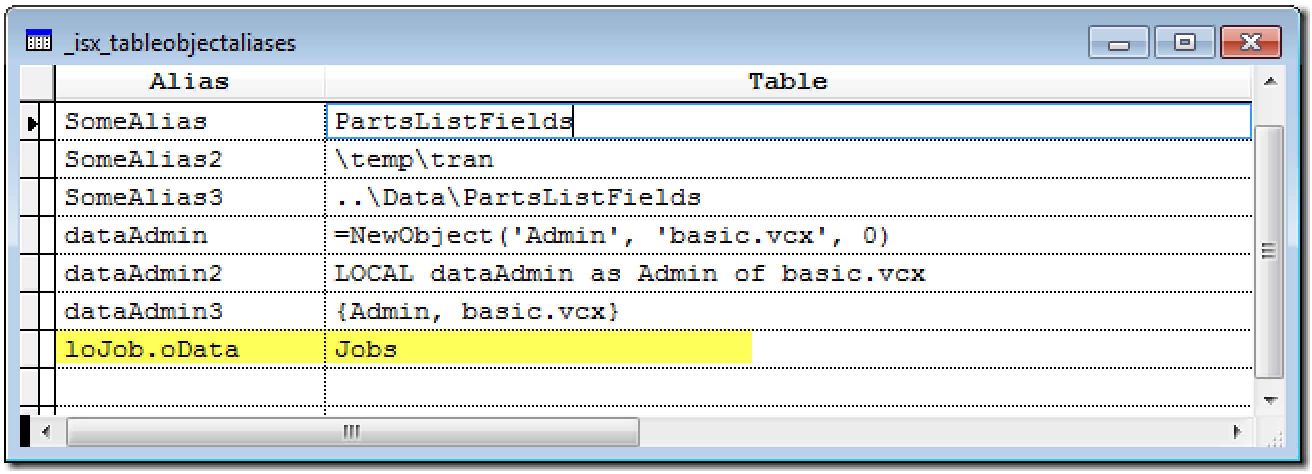
The alias field may also contain references to nested objects. Consider, for instance, the sample below, where oData is actually understood to be populated (at run time) by a Scatter from the Jobs table.



```
loJob.odata.
```

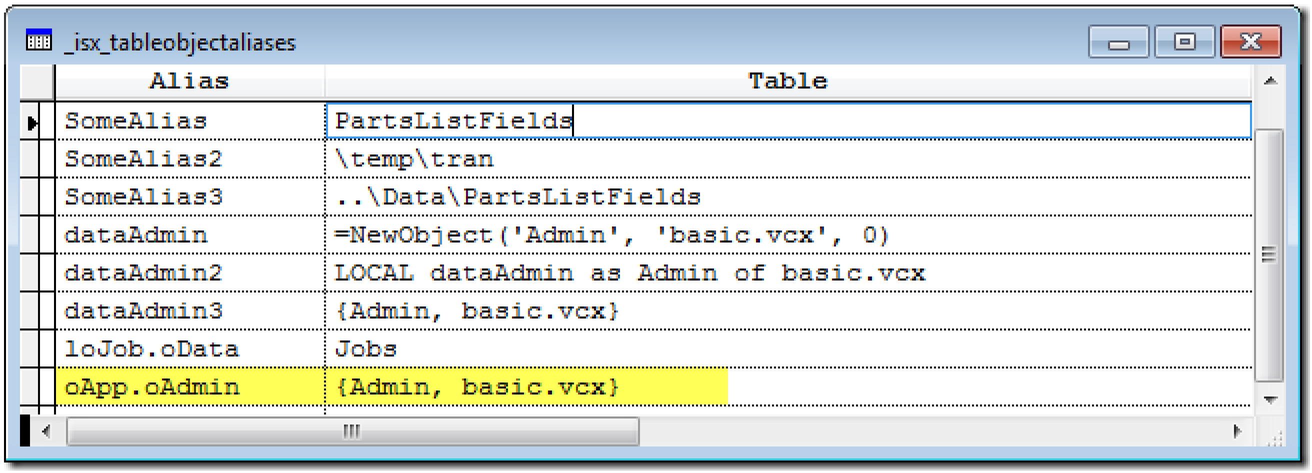
```
IDNum        I
Site         N(4)
Activity     I
PartNo       C(30)
Descript     C(60)
StartTime    T
NItems       N(8)
Paprwrktim   T
```

This is achieved by the entry shown below.



Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}
loJob.oData	Jobs

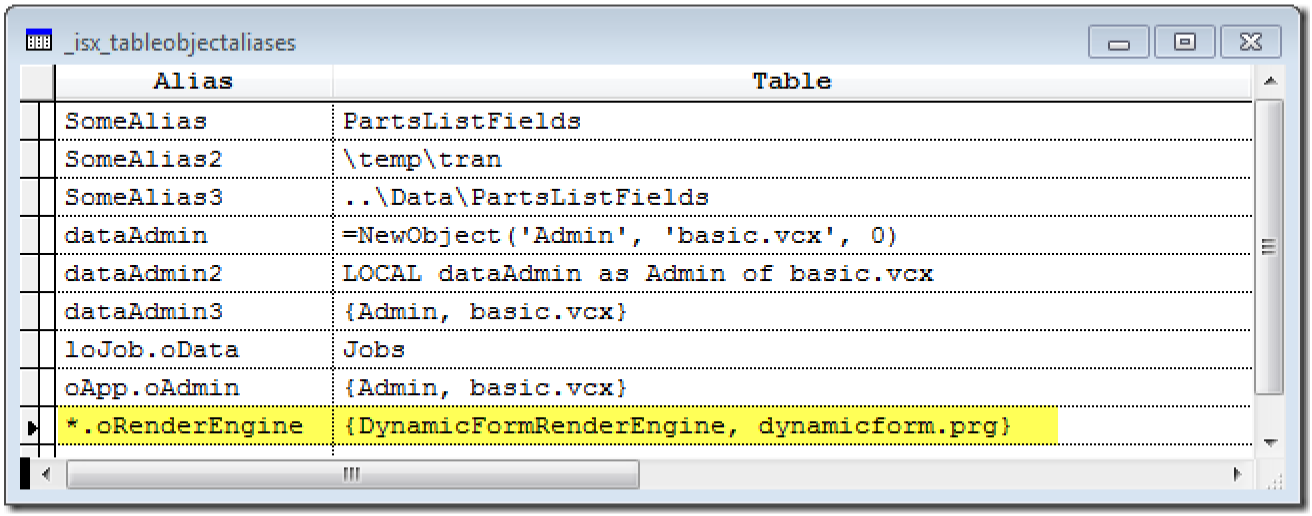
All of these combinations that can be used in the Table field apply equally well when using a nested object in the alias field. Below we have the definition of "oApp.oAdmin" so that it will present the list of properties from the Admin object defined in the Table field. (Note that any of the values in the Table field for "dataAdmin", "dataAdmin2", or "dataAdmin3" would have worked just as well.)



_isx_tableobjectaliases

Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}
loJob.oData	Jobs
oApp.oAdmin	{Admin, basic.vcx}

You can also use a single "*" as a wildcard in the Alias field.

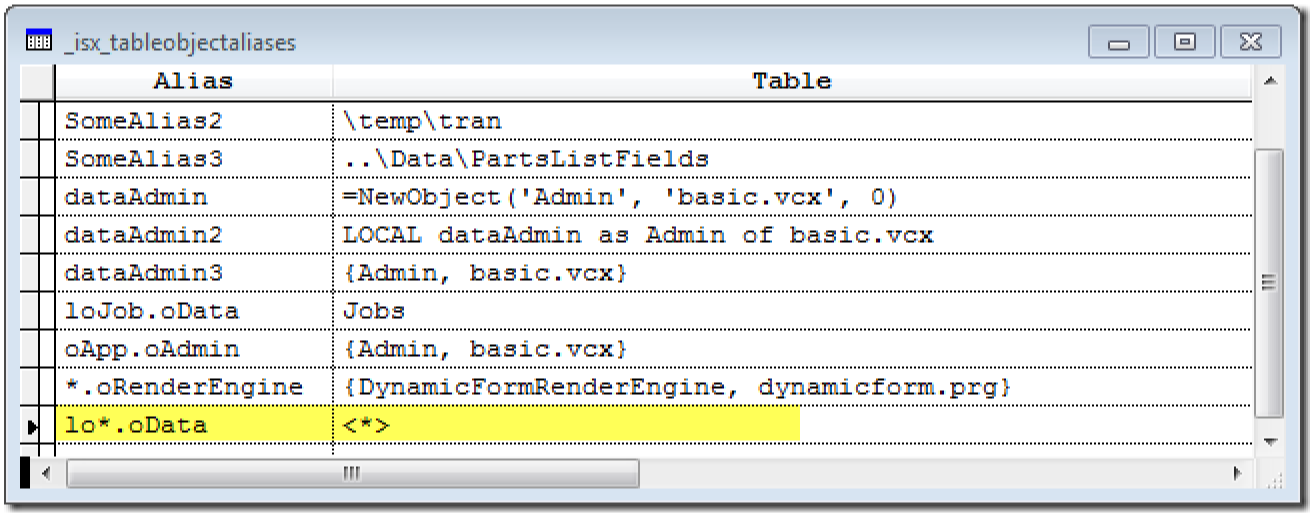


_isx_tableobjectaliases

Alias	Table
SomeAlias	PartsListFields
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}
loJob.oData	Jobs
oApp.oAdmin	{Admin, basic.vcx}
*.oRenderEngine	{DynamicFormRenderEngine, dynamicform.prg}

This allows you to see the PEMs for this particular object any time that it is referenced, regardless of the parent object that owns it.

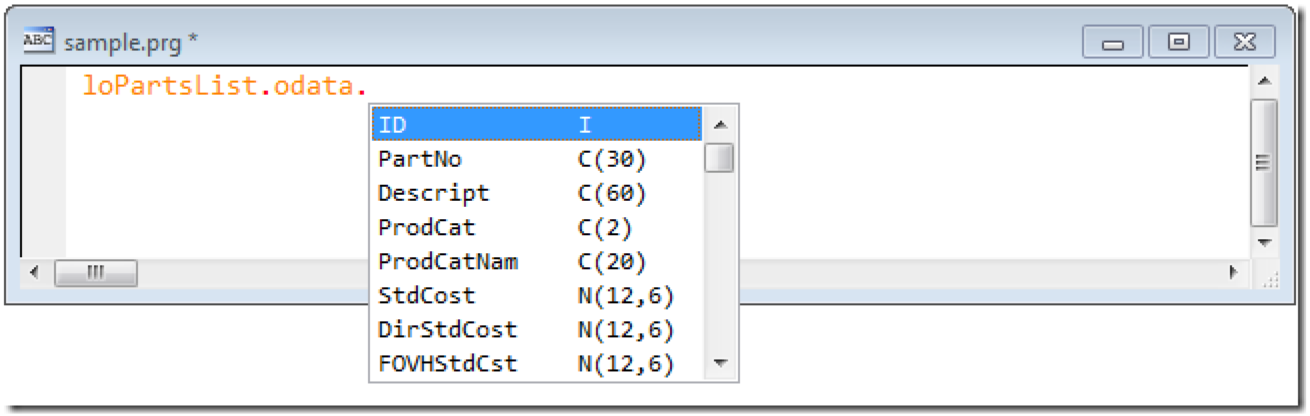
Finally, the characters that the wildcard represents can be used in the Table field; simply use <*> (as shown below)



_isx_tableobjectaliases

Alias	Table
SomeAlias2	\temp\tran
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}
loJob.oData	Jobs
oApp.oAdmin	{Admin, basic.vcx}
*.oRenderEngine	{DynamicFormRenderEngine, dynamicform.prg}
lo*.oData	<*>

Thus, in this example, "*" replaces the text "PartsList", so the fields from the PartsList table are displayed.

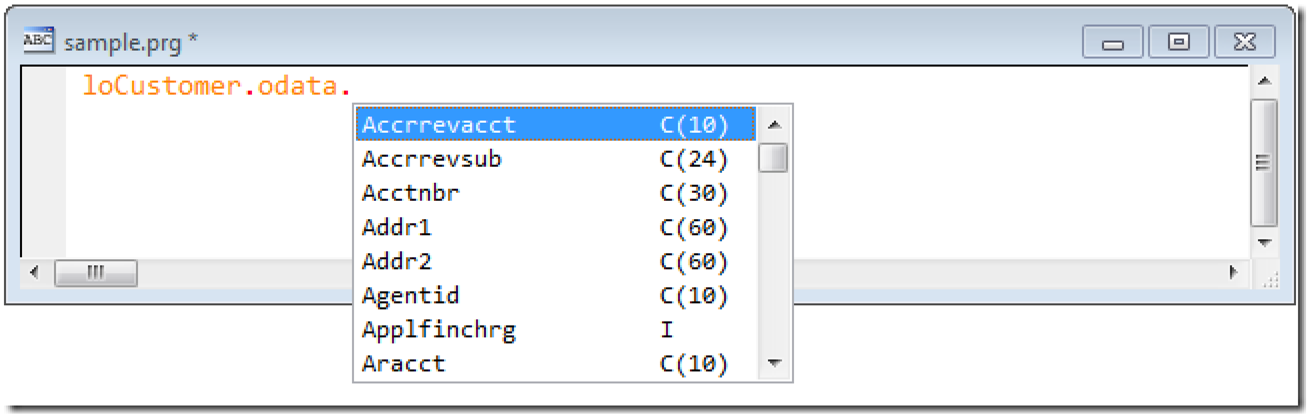


sample.prg *

```
loPartsList.odata.
```

```
ID          I
PartNo      C(30)
Descript    C(60)
ProdCat     C(2)
ProdCatNam  C(20)
StdCost     N(12,6)
DirStdCost  N(12,6)
FOVHStdCst  N(12,6)
```

And similarly for the Customer table ….

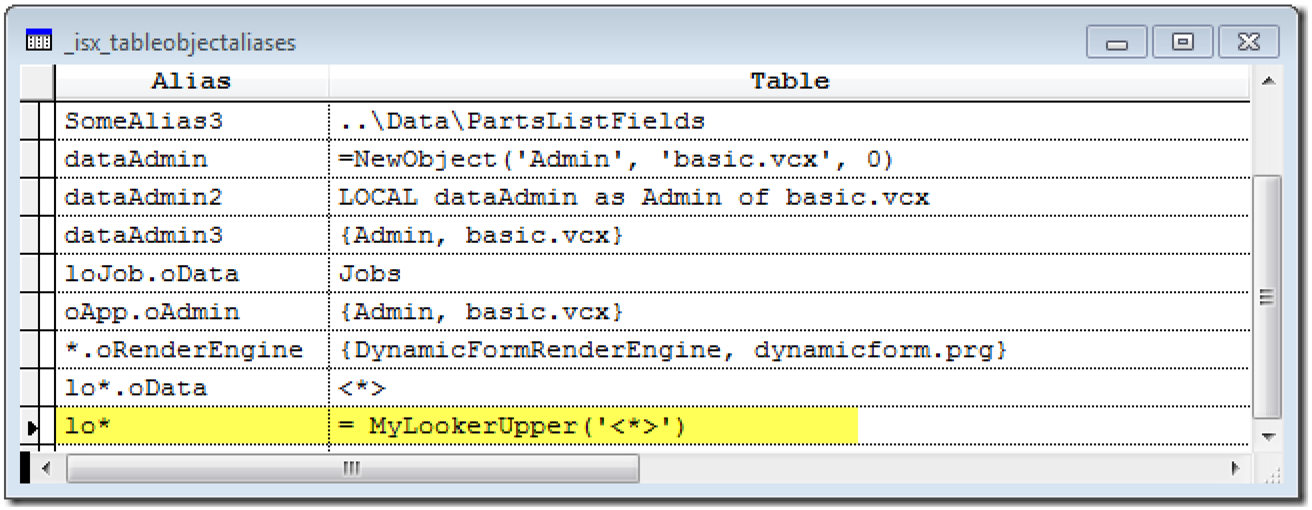


sample.prg *

```
loCustomer.odata.
```

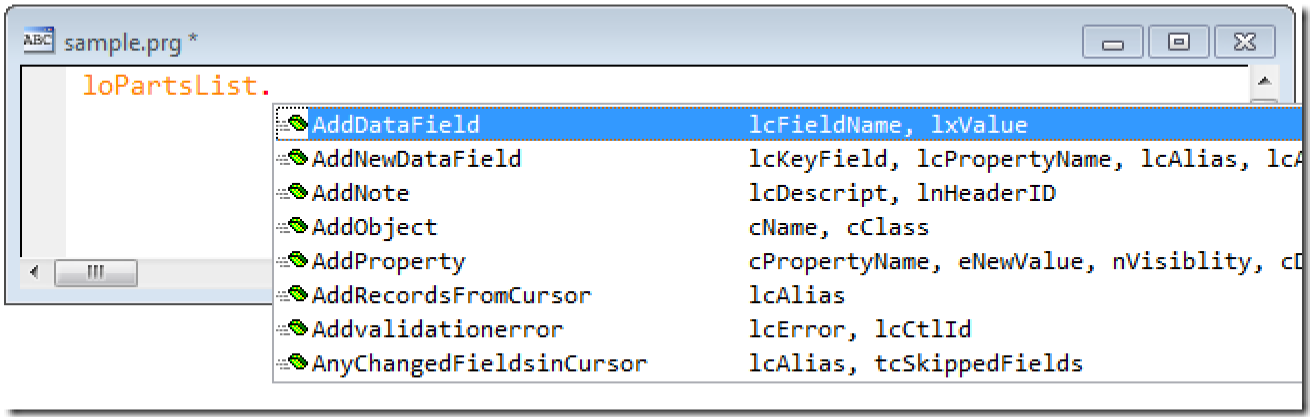
```
Accrrevacct     C(10)
Accrrevsub      C(24)
Acctnbr         C(30)
Addr1           C(60)
Addr2           C(60)
Agentid         C(10)
Applfinchrg     I
Aracct          C(10)
```

If you happen to name your objects consistently, you can pass the extracted text as a parameter to your own UDF, which should return the object as a result. (Note the necessary use of quote marks around <*>.)



Alias	Table
SomeAlias3	..\Data\PartsListFields
dataAdmin	=NewObject('Admin', 'basic.vcx', 0)
dataAdmin2	LOCAL dataAdmin as Admin of basic.vcx
dataAdmin3	{Admin, basic.vcx}
loJob.oData	Jobs
oApp.oAdmin	{Admin, basic.vcx}
*.oRenderEngine	{DynamicFormRenderEngine, dynamicform.prg}
lo*.oData	<*>
lo*	= MyLookerUpper('<*>')

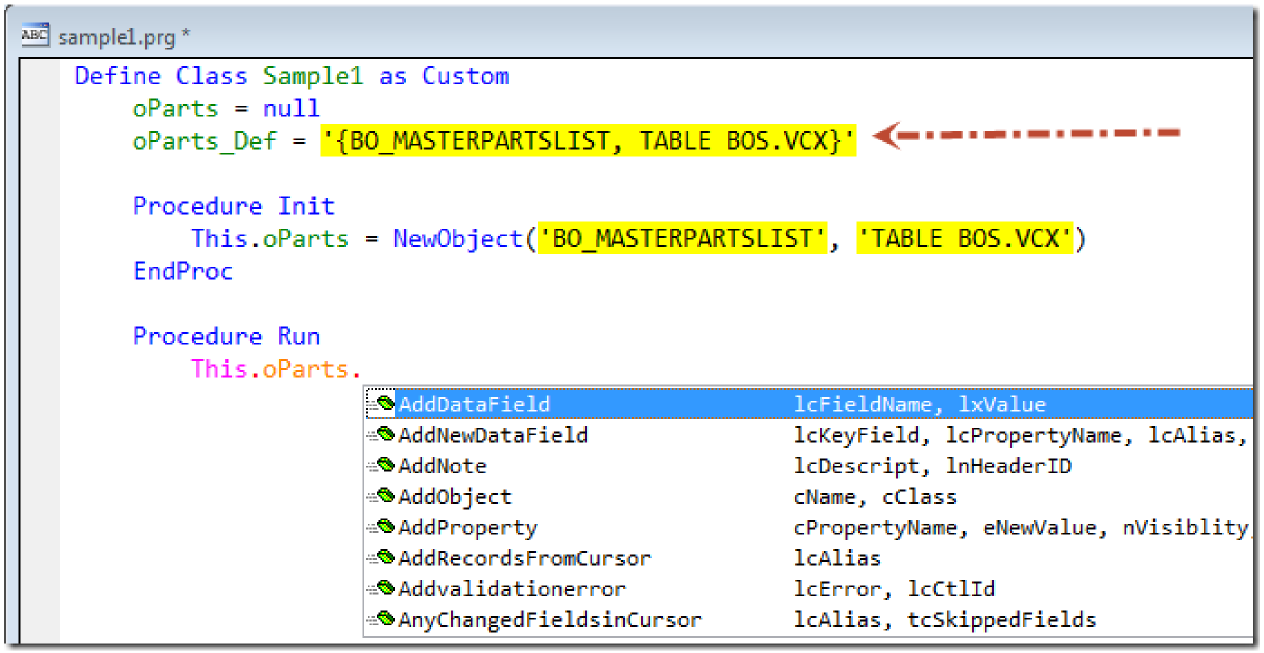
The following shows how 'PartsList' is passed as a parameter to my UDF "MyLookerUpper", which returns the object for 'PartsList' as a result. As noted before, UDF "MyLookerUpper" should call NewObject() with a third parameter of zero so that its INIT does not execute.



```
loPartsList.
AddDataField              lcFieldName, lxValue
AddNewDataField           lcKeyField, lcPropertyName, lcAlias, lcA
AddNote                   lcDescript, lnHeaderID
AddObject                 cName, cClass
AddProperty               cPropertyName, eNewValue, nVisiblity, cD
AddRecordsFromCursor      lcAlias
Addvalidationerror        lcError, lcCtlId
AnyChangedFieldsinCursor  lcAlias, tcSkippedFields
```

Form and Object-related Aliases

IntellisenseX provides dropdown lists for nested objects, available throughout an entire form or class, as shown in this example:



sample1.prg *

```
Define Class Sample1 as Custom
    oParts = null
    oParts_Def = '{BO_MASTERPARTSLIST, TABLE BOS.VCX}'

    Procedure Init
        This.oParts = NewObject('BO_MASTERPARTSLIST', 'TABLE BOS.VCX')
    EndProc

    Procedure Run
        This.oParts.
```

```
AddDataField               lcFieldName, lxValue
AddNewDataField            lcKeyField, lcPropertyName, lcAlias,
AddNote                    lcDescript, lnHeaderID
AddObject                  cName, cClass
AddProperty                cPropertyName, eNewValue, nVisiblity
AddRecordsFromCursor       lcAlias
Addvalidationerror         lcError, lcCtlId
AnyChangedFieldsinCursor   lcAlias, tcSkippedFields
```

The key to making this work is to define a property that can used by IntellisenseX to determine the dropdown list.

The name of the property is the name of the nested property (“oParts” in the example above) with “_Def” added as a suffix.

The value of the property is basically the same as used for Global or Local Aliases. It can be any one of a number of things, referring to either tables or objects.

- A table, cursor, or view name
- The full or relative path name to a table
- The name of an SQL table
- A reference to class from a class library; thus, “{class, classlibrary}“ *See note below for Forms and VCX classes.*
- A reference to class using the same syntax as the LOCAL command; thus, “Local loPAL as PAL of BO_PAL.VCX“
- A reference to an object; thus, “Thisform.oParts “
- An executable expression that returns an object or the name of an table, cursor, or view. This executable must start with an “=”; thus, something like “= MyGetObject(‘Parts’) “. *See note below for Forms and VCX classes.*

  Note: An unexpected problem arises when using the Property Sheet or PEM Editor to set the values for the two items noted above – values inside curly braces are converted *to dates, and values beginning with an = sign are saved as expressions. To avoid these issues, prefix the values with && – thus:*

```
   “&& {class, classlibrary}“
   “&& = MyGetObject(‘Parts’)“
```

Nested objects may take any of a number of forms, including:

- ThisForm.oBusObj
- This.oData
- This.oPartsList.oData
- This.oBusObj.oPartsList.oData

Nested objects may appear in forms, VCX-based classes, and PRG-based classes and there can be multiple levels of nesting.

Local Aliases

Local Aliases are like LOCAL statements (“Local name as class of classlib”) in two ways:

- they provide an annotation to your code so that IntellisenseX can provide a drop-down list.
- they apply only to the current method or procedure.

There are three equivalent ways to define Local Aliases:

- **{{** SomeAlias = What-It-Stand-For **}}** ... in any comment
- ***#Alias** SomeAlias = What-It-Stand-For ... at the beginning of a line
- **&&#Alias** SomeAlias = What-It-Stand-For ... at the end of a line

(The bold text above must be entered exactly as is, with no intervening spaces.)

“What-It-Stands-For” can be any one of a number of things, referring to either tables or objects.

- A table, cursor, or view name
- The full or relative path name to a table
- The name of an SQL table
- A reference to class from a class library; thus, “{{ loObject == {class, classlibrary} }} “
- A reference to an object; thus, “{{ loObject == Thisform.oParts}} “

- An executable expression that returns an object or the name of an table, cursor, or view. This executable must start with an "="; thus, something like "{{ loObject = = MyGetObject('Parts')}} ". (Note the double '=' signs.)

One handy use of this occurs with variables created using the SCATTER NAME command.

```
Select MyTable
Scatter Name loObject && {{ loObject = MyTable }}
```

This provides the dropdown list for the table MyTable when the properties of loObject are referenced.

Note also that "SomeAlias" can also refer to compound names, such as This.oMainObj or This.oMainObj.oData; the substitution rules work the same way.

Plug-Ins

There are five Plug-Ins that provide the last type of customization available for IntellisenseX. Even though they may seem quite obscure at first, they each provide quite powerful enhancements, far beyond what you might first expect.

Each of the plug-ins contains comments that explain (or, at least, should explain) the parameters that are passed in and the form of the result. Thus, the explanations here address the general concepts of their usage.

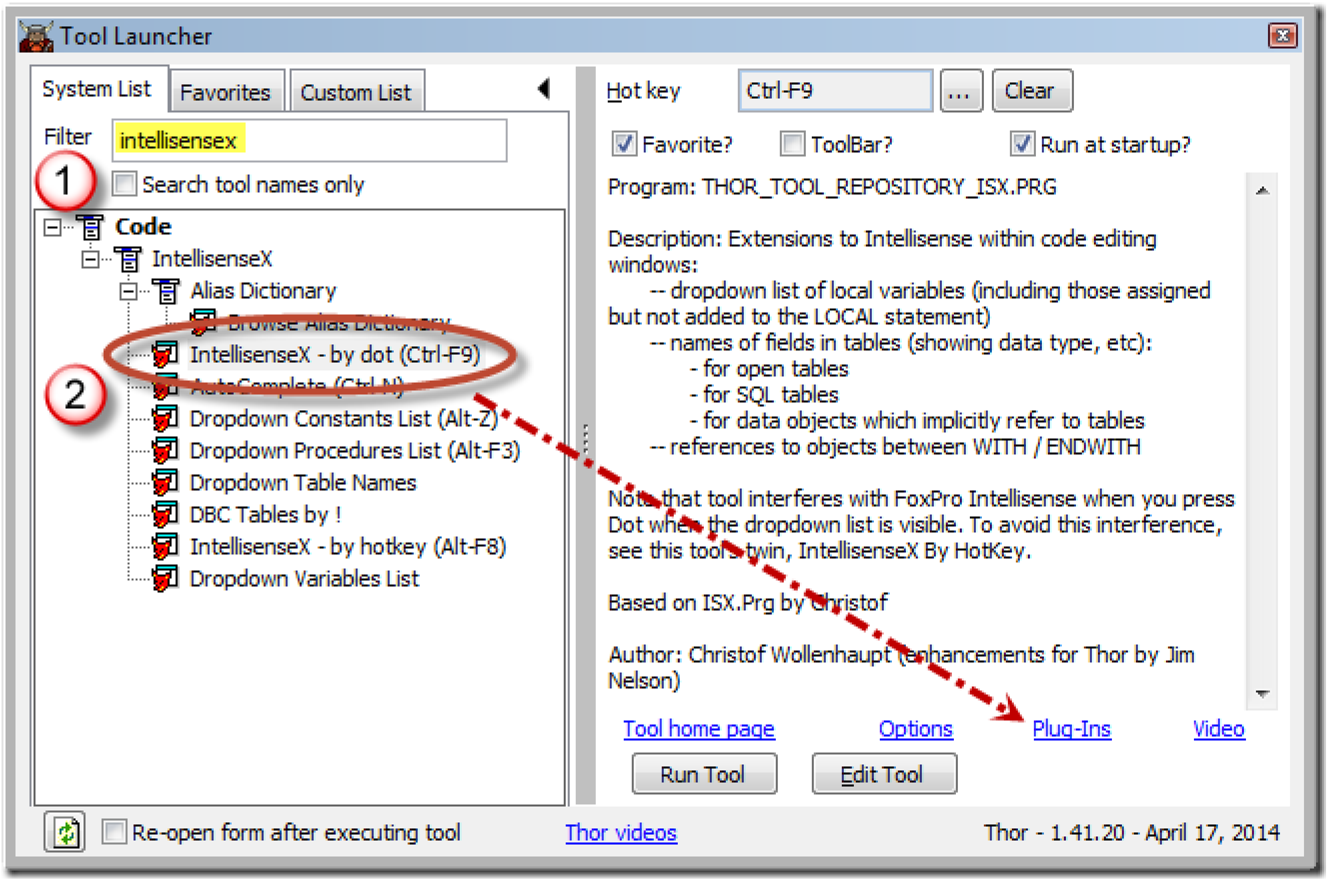
It can be very helpful to think of these plug-ins as event handlers. Until you are quite familiar with the use of IntellisenseX, their relevance may be hard to see, just as the relevance of many event handlers for a FoxPro object may be hard to see at first. Eventually, though, you may come across a situation where IntellisenseX does not provide a dropdown, but you can imagine it would be possible to do so. At that time, search through the list of plug-ins to see which might apply, open it up and dig through the sample code.

You need not be concerned about when a plug-in is called, just as you are not concerned about when an event handler fires. The plug-in will be called when it is appropriate.

There is no natural order to these Plug-Ins (there really is no relationship between them), so they are explored in alphabetical order. Note, however, that for some of them their usage has changed over time, so the names are not necessarily clear explanations of what they do.

To access them, follow these steps:

1. Open Tool Launcher
2. Enter "IntellisenseX" in the filter box
3. Click on the tool "IntellisenseX – by Dot" in the TreeView on the left.
4. Click on the Plug-Ins link to open the Plug-Ins Form.



Filter: intellisensex

☐ Search tool names only

- Code
  - IntellisenseX
    - Alias Dictionary
      - Browse Alias Dictionary
    - IntellisenseX - by dot (Ctrl-F9)
    - AutoComplete (Ctrl-N)
    - Dropdown Constants List (Alt-Z)
    - Dropdown Procedures List (Alt-F3)
    - Dropdown Table Names
    - DBC Tables by !
    - IntellisenseX - by hotkey (Alt-F8)
    - Dropdown Variables List

Hot key: Ctrl-F9

☑ Favorite? ☐ ToolBar? ☑ Run at startup?

Program: THOR_TOOL_REPOSITORY_ISX.PRG

Description: Extensions to Intellisense within code editing windows:
-- dropdown list of local variables (including those assigned but not added to the LOCAL statement)
-- names of fields in tables (showing data type, etc):
- for open tables
- for SQL tables
- for data objects which implicitly refer to tables
-- references to objects between WITH / ENDWITH

Note that tool interferes with FoxPro Intellisense when you press Dot when the dropdown list is visible. To avoid this interference, see this tool's twin, IntellisenseX By HotKey.

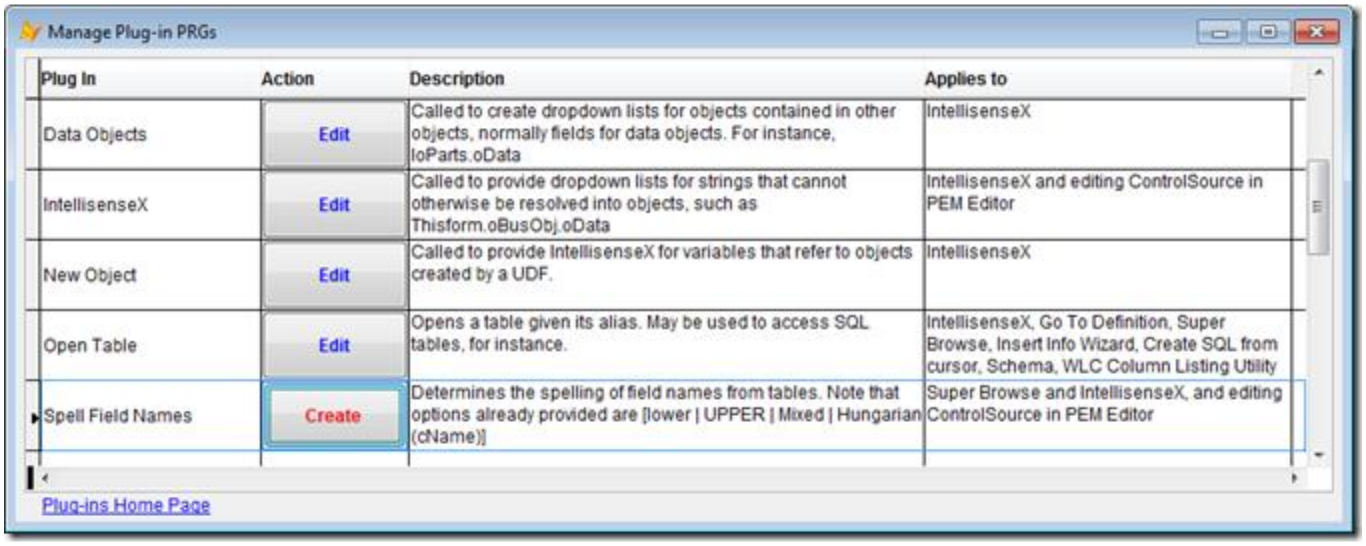
Based on ISX.Prg by Christof

Author: Christof Wollenhaupt (enhancements for Thor by Jim Nelson)

☐ Re-open form after executing tool

Thor - 1.41.20 - April 17, 2014

The Plug-Ins form (when accessed this way) shows only those Plug-Ins that apply to IntellisenseX:



Plug In	Action	Description	Applies to
Data Objects	Edit	Called to create dropdown lists for objects contained in other objects, normally fields for data objects. For instance, loParts.oData	IntellisenseX
IntellisenseX	Edit	Called to provide dropdown lists for strings that cannot otherwise be resolved into objects, such as Thisform.oBusObj.oData	IntellisenseX and editing ControlSource in PEM Editor
New Object	Edit	Called to provide IntellisenseX for variables that refer to objects created by a UDF.	IntellisenseX
Open Table	Edit	Opens a table given its alias. May be used to access SQL tables, for instance.	IntellisenseX, Go To Definition, Super Browse, Insert Info Wizard, Create SQL from cursor, Schema, WLC Column Listing Utility
Spell Field Names	Create	Determines the spelling of field names from tables. Note that options already provided are [lower | UPPER | Mixed | Hungarian (cName)]	Super Browse and IntellisenseX, and editing ControlSource in PEM Editor

"Data Objects" Plug-In

This plug-in allows you to identify an object (or table) referenced in some form of nested usage. It is called when IntellisenseX has been able to resolve part of the name ("Thisform" or "Thisform.oBusObj" e.g.,) into a real object but not the member name ("oData" or "oParts", e.g.)

The plug-in is called with the object (in a unique structure – see the comments) and the member name as parameters. The plug-In returns the appropriate object or table reference, if appropriate.

Some examples:

- This.oCustomers – if a consistent naming convention is used for objects (such as "oCustomers" here), the plug-in could take the characters after the "o" and see if there is a business object to handle that name, returning that object.
- ThisForm.oBusObj – similar to the previous item, but the plug-in could look for an property ("cAlias", e.g.,) that identifies the table for the business object being referenced.
- ThisForm.oBusObj.oData – similar to the previous item, but instead of returning the business object being referenced, returns the alias referred to in the property ("cAlias"), causing the drop down to show the fields from that table.
- Thisform.oJob.oCustomers.oData – the plug-in may be called multiple times to obtain a single dropdown list; in this case, first to resolve "Thisform.oJob", then "Thisform.oJob.oCustomers", and finally "Thisform.oJob.oCustomers.oData". This is transparent to you, however.
- loJob.oData – can provide the dropdown list of fields from the loJob table – if loJob can already have been resolved into an object. That can be done by the next Plug-In, "IntellisenseX".

"IntellisenseX" Plug-In

This plug-in allows you to identify an object (or table) based on the entire text that precedes the dot that you entered. That entire text is passed in as a parameter.

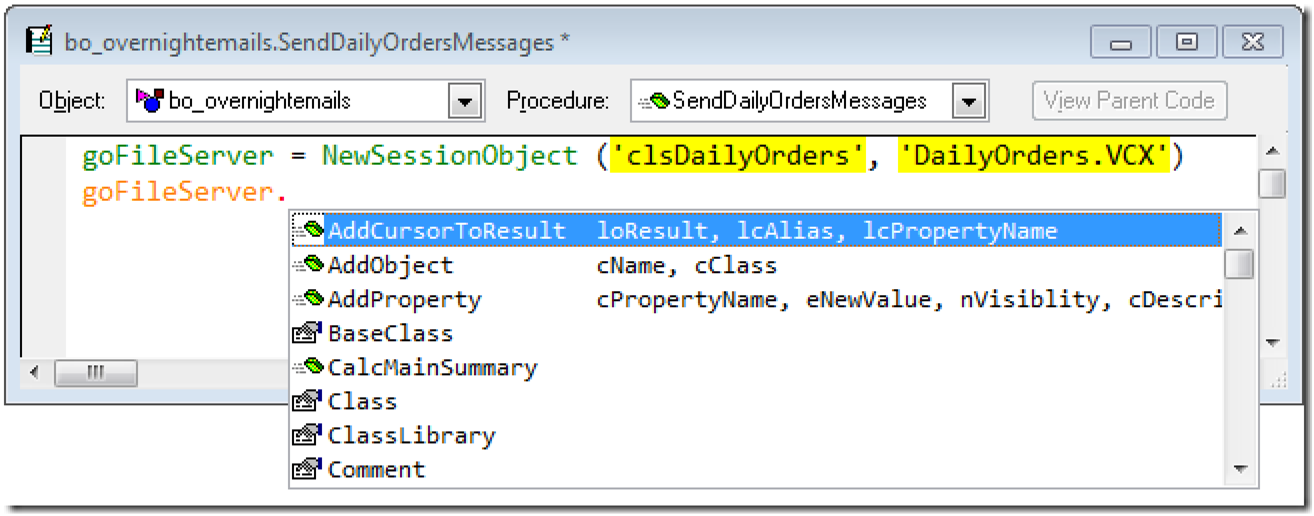
This plug-in is closely related to the "Data Objects" plug-in, and is called in all cases where the "Data Objects" plug-in was unable to return a usable result. The difference between the two is the parameters that are passed in.

But it can also handle getting "loJob" as a parameter (in this case, the "Data Objects" plug-in is not called). If "loJob" can be resolved into an object by this plug-in, there will be a dropdown list for loJob.

This plug-in is also called by PEM Editor when you are setting the values of properties, allowing you to get dropdown lists for tables of data objects when setting a ControlSource.

“New Object” Plug-In

Use this plug-in when you use a UDF to create objects, instead of NEWOBJECT() or CREATEOBJECT(). In the example below, a UDF named NewSessionObject is used to create an object, and the plug-in allows IntellisenseX to display the dropdown list for it (which happens to use the same parameters as NEWOBJECT, but this is not necessary).


bo_overnightemails.SendDailyOrdersMessages *
Object: bo_overnightemails
Procedure: SendDailyOrdersMessages
View Parent Code

```
goFileServer = NewSessionObject ('clsDailyOrders', 'DailyOrders.VCX')
goFileServer.
```

```
AddCursorToResult  loResult, lcAlias, lcPropertyName
AddObject          cName, cClass
AddProperty        cPropertyName, eNewValue, nVisiblity, cDescri
BaseClass
CalcMainSummary
Class
ClassLibrary
Comment
```

“Open Table” Plug-In

When IntellisenseX encounters a single (non-nested) name before the dot, it checks whether the name corresponds to an existing alias, or (optionally) the name of a table in your SQL Server database, or the name of a file in the path, in the Data Environment or an open DBC, or (optionally) in the MRU list. All of this is handled by the default version of this plug-in.

The plug-in is called with the potential table name as its parameter, which the plug-in can open if not already handled by the default behavior. *(Author’s note: In my universe, all table names and the folders they are found in are themselves stored in a table, where the alias is used as a key to the table. These tables are only opened by a single UDF which takes the alias as a parameter. My personal version of this plug-in calls this UDF to open the table.)*

There are examples of other uses in the comments in the code.

“Spell Field Names” Plug-In

This plug-in has been essentially superseded by the Custom Keyword List, but can be used in cases where the Custom Keyword List is not used, is insufficient, or there are other rules for field names (such as project or customer specific rules).

IntellisenseX “by Dot” or “by Hot Key”?

There are two different ways that you can invoke IntellisenseX:

- Thor Tool *IntellisenseX by Dot* is a toggle that turns on/turns off the use of IntellisenseX. When turned on, IntellisenseX is invoked every time you press the dot (the period).
- Thor Tool *IntellisenseX by Hot Key* causes IntellisenseX to be invoked only when you press the hot key you have assigned to it. You press the hot key when you want IntellisenseX (and it supplies the dot for you).

See all this video: “IntellisenseX by Dot” vs “IntellisenseX by Hot Key” (3:29)

IntellisenseX by Dot is the “natural” way to invoke IntellisenseX, as it mimics the way native Intellisense works. There are some minor issues with it, however, including the following:

- The selection box may be slow in coming up, particularly when listing properties and methods for an object. (This is addressed by using the “Custom Keyword List”, described later.)
- It may stumble over itself a little when you enter any of .T., .F., or .NULL. This can be avoided by typing slower, but nobody wants to do that.
- It uses a timer that fires every second, which can cause annoying behavior in the debugger. The timer is absolutely essential since *IntellisenseX by Dot* must be disabled when the focus is not on a code window

All of these issues can be overcome by using *IntellisenseX by Hot Key* instead, each time you want to invoke IntellisenseX. However, besides learning to use a different key than dot to invoke IntellisenseX, there is another drawback as well –there are features available from IntellisenseX not provided by native Intellisense. These will come as a pleasant surprise if using *IntellisenseX by Dot* (the selection box will simply appear when you press the dot) but you may never be aware of them if using *IntellisenseX by Hot Key*.

Other features from IntellisenseX by Hot Key

There are two features are available from *IntellisenseX by Hot Key*:

- If you enter the name of a DBC following by a ‘!’ and then use *IntellisenseX by Hot Key*, the selection box lists all tables and views in that DBC. (If the database is already open, you need not enter the DBC name). This list, curiously, also include files from the Data Environment of a form being edited. This is the same behavior as the tool *DBC Tables by !*, described later.
- If you enter a ‘*’ followed by *IntellisenseX by Hot Key*, the selection box lists all tables in the current folder and the path. This is the same behavior as the tool *Dropdown Table Names*, described later.

Deficiencies in IntellisenseX (and how to avoid them)

All tools have their problems, and IntellisenseX is no different. Most noticeable is that under some conditions it so *slow*, but there are a other issues as well.

It's slow

The most serious problem with IntellisenseX is that it is SO SLOW when getting the list of properties and methods in a VCX or SCX. The process of getting the correct case for names in a VCX/SCX, requiring parsing MemberData in parent class(es), takes more time than anybody would be willing to wait each time you enter THISFORM followed by a dot.

There is a similar problem when working in PRG-based classes where the parent class is anything other than a FoxPro base class. There is no good way to obtain the correct case for inherited names (PRG-based classes don't even use MemberData), so all inherited custom names end up in lower case.

The suggested remedy for these two problems is the same – the Custom Keyword List, described in the next section.

It mucks up the debugger

IntellisenseX uses a timer, which fires every second. This can mess up the debugger.

To avoid this problem, toggle IntellisenseX on and off. *(Author's note: I have assigned a hot key to IntellisenseX by Dot so that I can turn it off when it is getting in the way.)*

"It doesn't work in my command window"

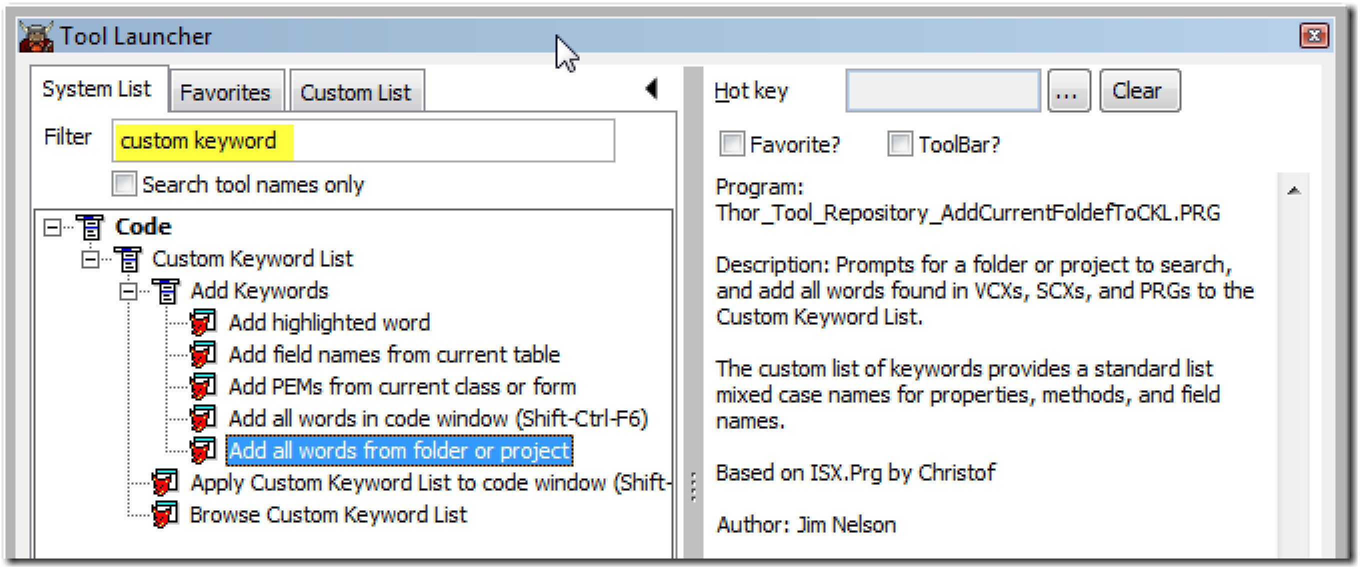
IntellisenseX does not work in the Command Window if it is marked Dockable. The problem is that there is no way (at least that anybody has discovered) to determine where the pop-up window should appear.

The Custom Keyword List

The Custom Keyword List is just a simple table with the list of all words from your programming universe, where each word is saved with the mixture of upper/lower case that you prefer. This table can be created easily, updated automatically, and applied to any block of code. Not only does this eliminate the problems mentioned above, it ensures that your words are always used consistently (same upper/lower case).

Getting Started

First, create the Custom Keyword List by running Thor Tool *Add all words from folder or project.* (You can find all the tools referenced here by filtering on "Custom Keyword" in the Thor Launcher.)



Tool Launcher

Filter: custom keyword

☐ Search tool names only

☐ Favorite? ☐ ToolBar?

- Code
  - Custom Keyword List
    - Add Keywords
      - Add highlighted word
      - Add field names from current table
      - Add PEMs from current class or form
      - Add all words in code window (Shift-Ctrl-F6)
      - Add all words from folder or project
    - Apply Custom Keyword List to code window (Shift-
    - Browse Custom Keyword List

Program:
Thor_Tool_Repository_AddCurrentFoldefToCKL.PRG

Description: Prompts for a folder or project to search, and add all words found in VCXs, SCXs, and PRGs to the Custom Keyword List.

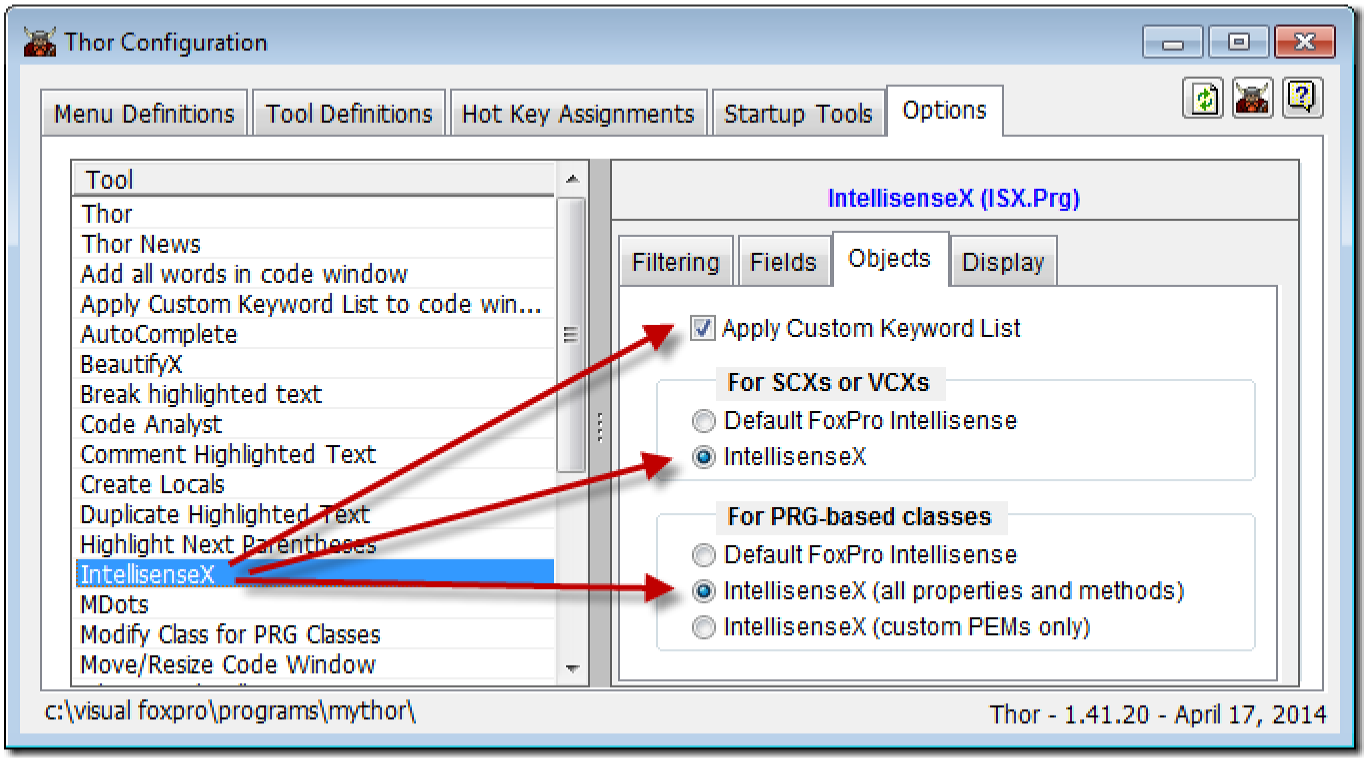
The custom list of keywords provides a standard list mixed case names for properties, methods, and field names.

Based on ISX.Prg by Christof

Author: Jim Nelson

This tool will run for a few minutes as it scours an entire folder or project for all programming words. Eventually, a form will come up showing you the list of all words in found. You can do some editing of the list, but for starters the suggestion is to simply save everything.

Next, go to the options page for IntellisenseX in the Thor Configuration form to select the settings such that IntellisenseX will use the Custom Keyword List.



Thor Configuration

Tool
Thor
Thor News
Add all words in code window
Apply Custom Keyword List to code win...
AutoComplete
BeautifyX
Break highlighted text
Code Analyst
Comment Highlighted Text
Create Locals
Duplicate Highlighted Text
Highlight Next Parentheses
IntellisenseX
MDots
Modify Class for PRG Classes
Move/Resize Code Window

IntellisenseX (ISX.Prg)

☑ Apply Custom Keyword List

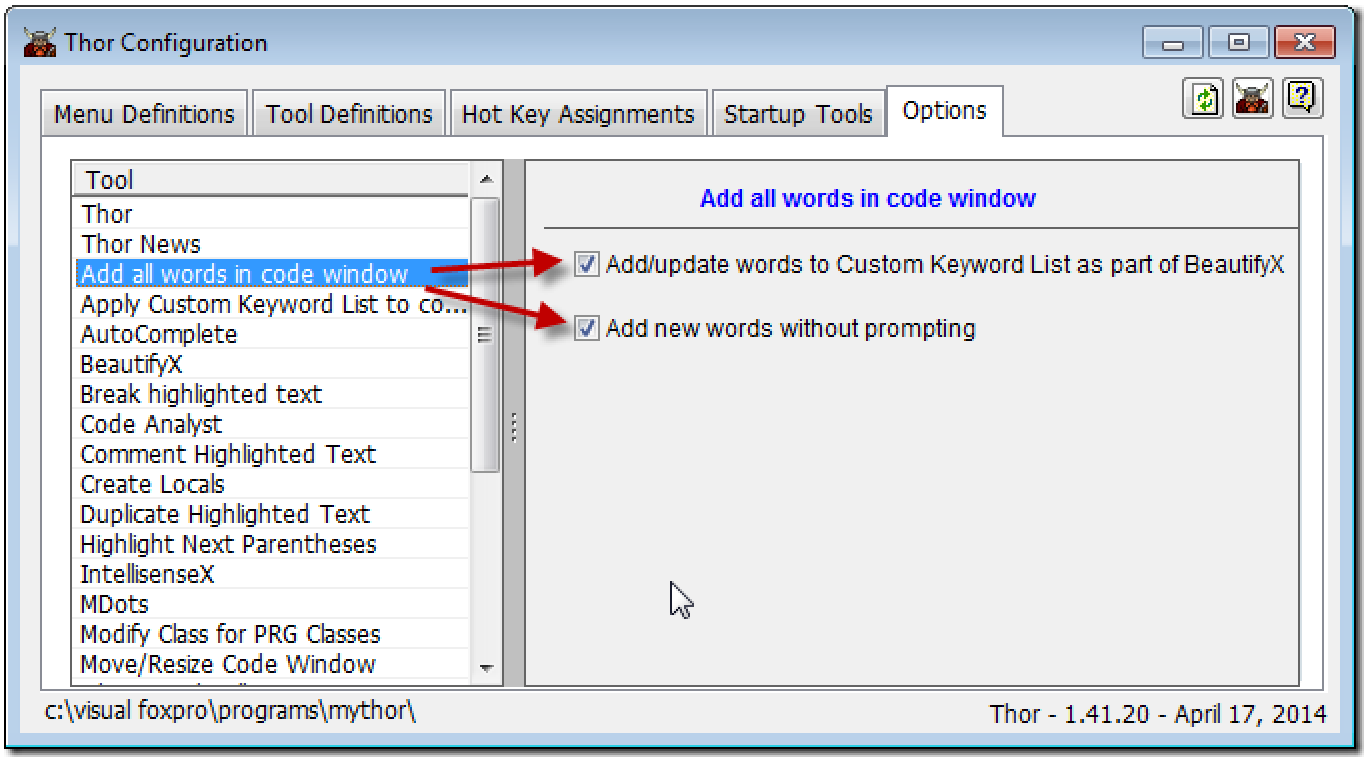
For SCXs or VCXs
Default FoxPro Intellisense [✓ IntellisenseX]

For PRG-based classes
Default FoxPro Intellisense [✓ IntellisenseX (all properties and methods)] IntellisenseX (custom PEMs only)

c:\visual foxpro\programs\mythor\

Thor - 1.41.20 - April 17, 2014

Finally, mark these checkboxes for option “Add all words in code window” as well, so that new words you create going forward, including properties and methods created by PEM Editor or any of its related tools, are automatically added to the Custom Keyword List. Do so even if you don’t used BeautifyX (but more on that in a bit).



Add all words in code window

- [✓] Add/update words to Custom Keyword List as part of BeautifyX
- [✓] Add new words without prompting

c:\visual foxpro\programs\mythor\

Thor - 1.41.20 - April 17, 2014

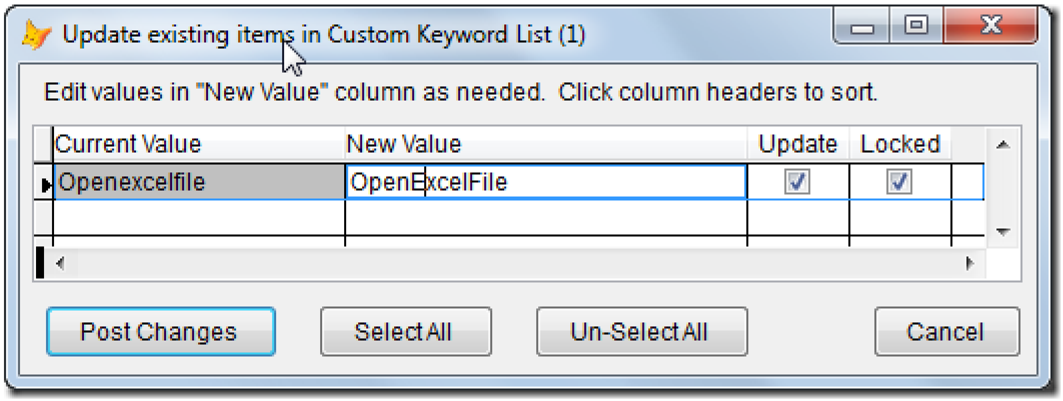
After performing these three steps, you’re on your way.

Updating the Custom Keyword List Programmatically

You can programmatically add words to this list at any time by using any of the following Thor tools:

- *Add all words from folder or project*
- *Add all words in code window*
- *Add PEMS from current class or form*
- *Add fields names from current table*

If there are any new words encountered, a form opens for you to approve the new words; if there are any words found that conflict with words already in the table, a separate form opens for you to select which you want to use.



Update existing items in Custom Keyword List (1)

Edit values in "New Value" column as needed. Click column headers to sort.

Current Value	New Value	Update	Locked
Openexcelfile	OpenExcelFile	✓	✓

What does "Locked" mean?

Some words may have inconsistent usages – "Openexcelfile" or "OpenExcelfile" or "OpenExcelFile" or ? Marking a word as "Locked" indicates that that is your preferred usage and you will never be asked about possible conflicts again.

Updating the Custom Keyword List Manually

You can use the tool *Add highlighted word* to add or update a word directly to the Custom Keyword List (this also marks it as Locked – your preferred usage).

You can also open the table with tool *Browse Custom Keyword List* and make any modifications as needed. This table is found in your "My Tools" folder.

```
_Screen.cThorFolder + 'Tools\My Tools\KeywordList.DBF'
```

Using BeautifyX

If you're not already using Thor tool BeautifyX all the time, it's time to change. It provides a wide range of features and you can choose only the ones work for you (yes, you can use BeautifyX without using FoxPro's native Beautify.)

The options include

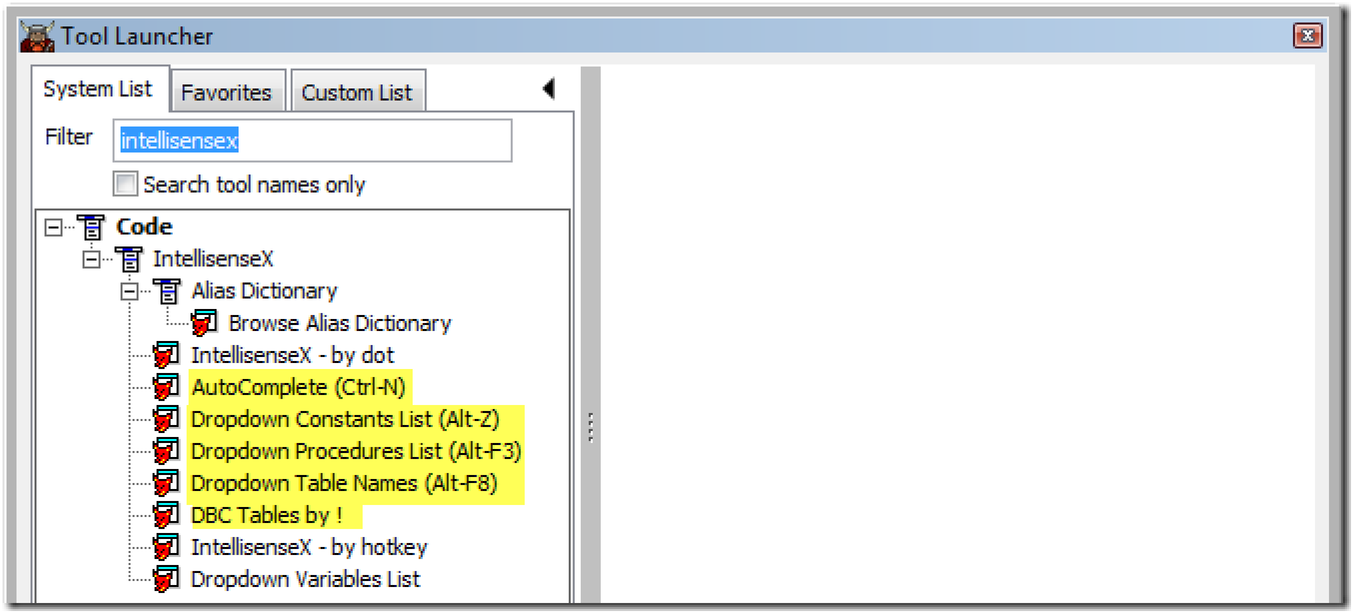
- Apply native Beautify
- Provide consistent spacing around operators and commas, align semi-colons, indentation for continuation lines
- Highly customizable formatting for SQL-SELECT, SQL-UPDATE, SQL-DELETE, and REPLACE statements, (conditionally within TEXT / ENDTEXT structures)
- Run Thor tool "Create Locals'
- Add MDots
- Check for RETURNs between WITH/ENDWITH (these create latent C5 errors)
- Apply Custom Keyword List

- Add words to Custom Keyword List.

Other IntellisenseX Tools

There are a number of Thor tools other than *IntellisenseX* that provide dropdown lists of names to select from. These are not the familiar lists that native FoxPro Intellisense provides (member names or field names), but rather other lists that make sense in your FoxPro IDE. These lists are not activated by pressing the dot (like IntellisenseX), so you will need to access them another way (by assigning a hot key or by adding them to the Thor toolbar or a menu).

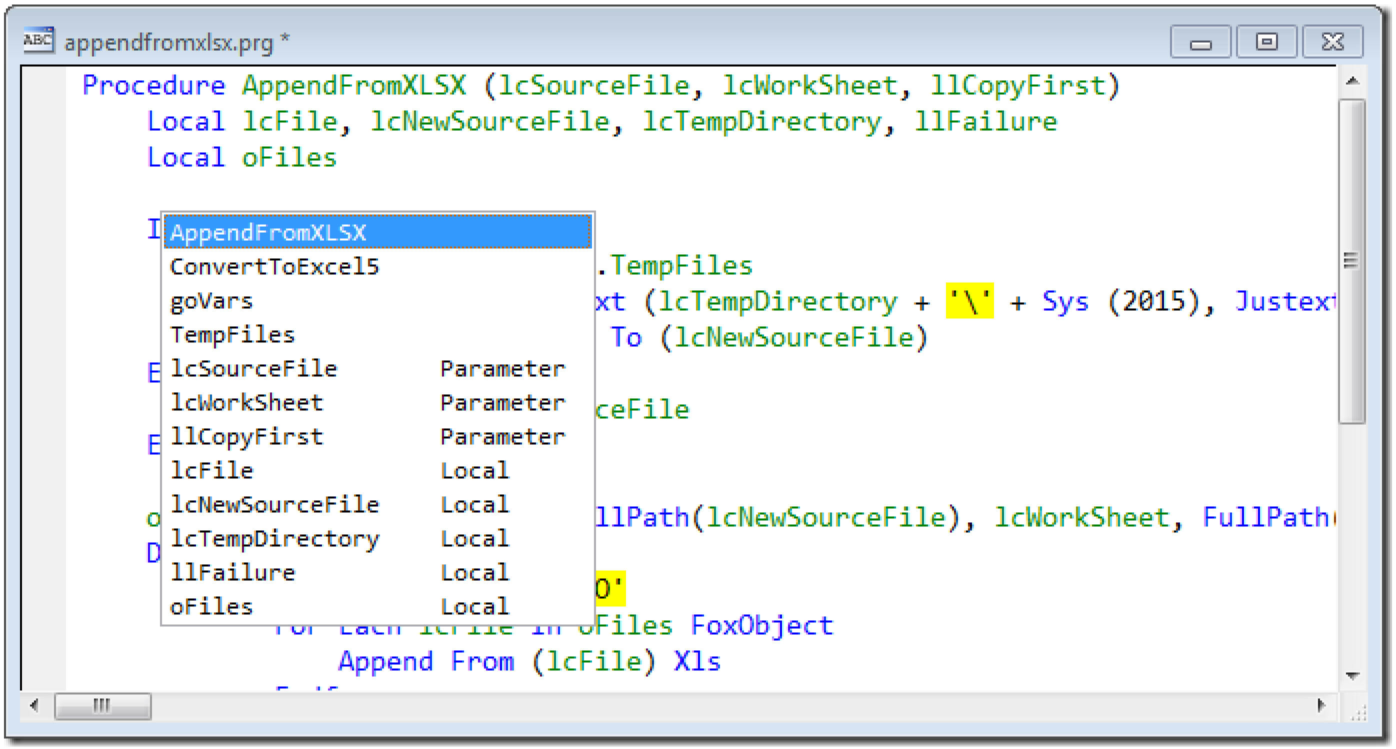
These tools can be found using the Tool Launcher:



Filter: intellisensex

- Code
  - IntellisenseX
    - Alias Dictionary
      - Browse Alias Dictionary
    - IntellisenseX - by dot
    - AutoComplete (Ctrl-N)
    - Dropdown Constants List (Alt-Z)
    - Dropdown Procedures List (Alt-F3)
    - Dropdown Table Names (Alt-F8)
    - DBC Tables by !
    - IntellisenseX - by hotkey
    - Dropdown Variables List

All of these tools work essentially the same way. When you invoke them, you get a dropdown list that you can select from, just as when you select from dropdown lists created by IntellisenseX. The following demonstrates a use of *AutoComplete*, showing all the names referenced in this procedure.


appendfromxlsx.prg *

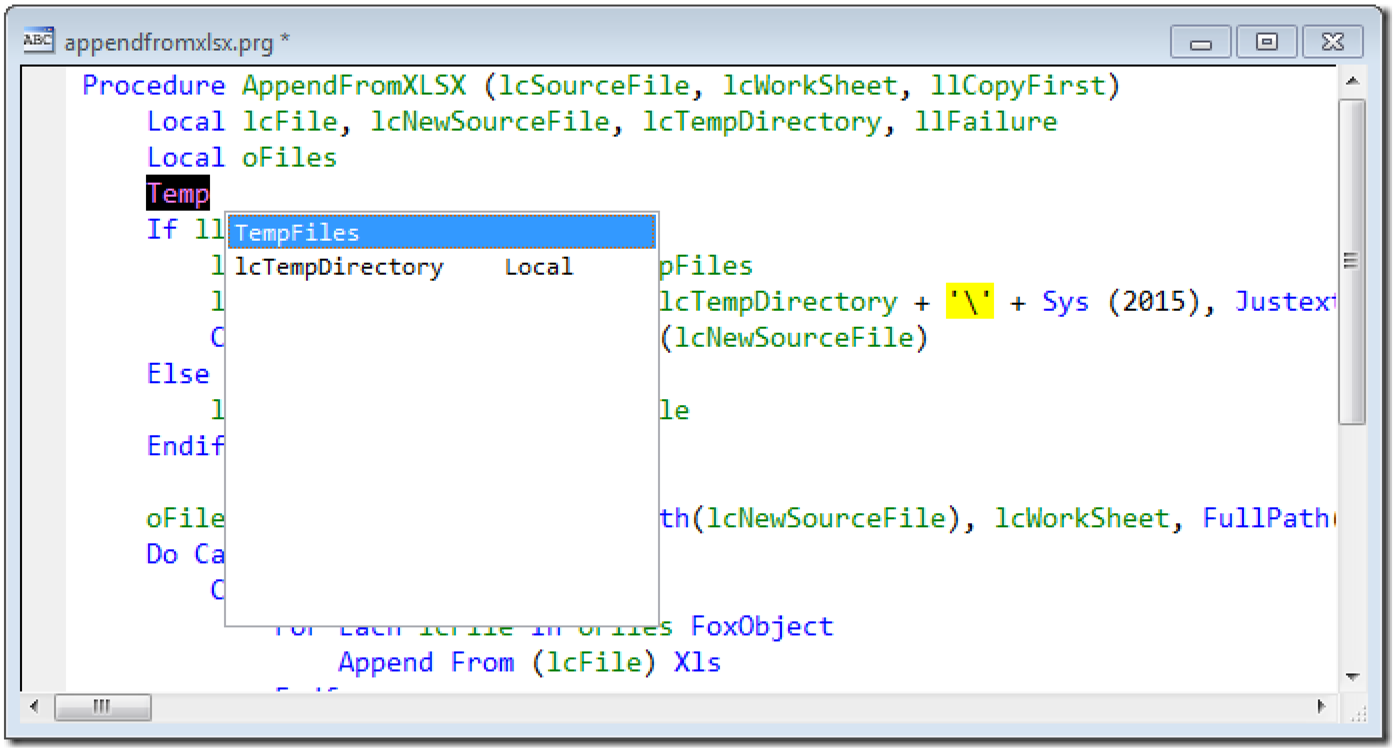
```
Procedure AppendFromXLSX (lcSourceFile, lcWorkSheet, llCopyFirst)
    Local lcFile, lcNewSourceFile, lcTempDirectory, llFailure
    Local oFiles

    I                            .TempFiles
                                 xt (lcTempDirectory + '\' + Sys (2015), Justext
                                  To (lcNewSourceFile)
    E
                                 ceFile
    E

    o                            llPath(lcNewSourceFile), lcWorkSheet, FullPath
    D
                                 0'
            For Each lcFile In oFiles FoxObject
                Append From (lcFile) Xls
```

```
AppendFromXLSX
ConvertToExcel5
goVars
TempFiles
lcSourceFile       Parameter
lcWorkSheet        Parameter
llCopyFirst        Parameter
lcFile             Local
lcNewSourceFile    Local
lcTempDirectory    Local
llFailure          Local
oFiles             Local
```

You can also begin typing part of the name before invoking the tool, such as in this example where “Temp” was entered before invoking *AutoComplete*.


appendfromxlsx.prg *

```
Procedure AppendFromXLSX (lcSourceFile, lcWorkSheet, llCopyFirst)
    Local lcFile, lcNewSourceFile, lcTempDirectory, llFailure
    Local oFiles
    Temp
    If ll                                 pFiles
        l                                 lcTempDirectory + '\' + Sys (2015), Justext
        l                                 (lcNewSourceFile)
        C
    Else
        l                                 le
    Endif

    oFile                                 th(lcNewSourceFile), lcWorkSheet, FullPath
    Do Ca
        C
            For Each lcFile In oFiles FoxObject
                Append From (lcFile) Xls
```

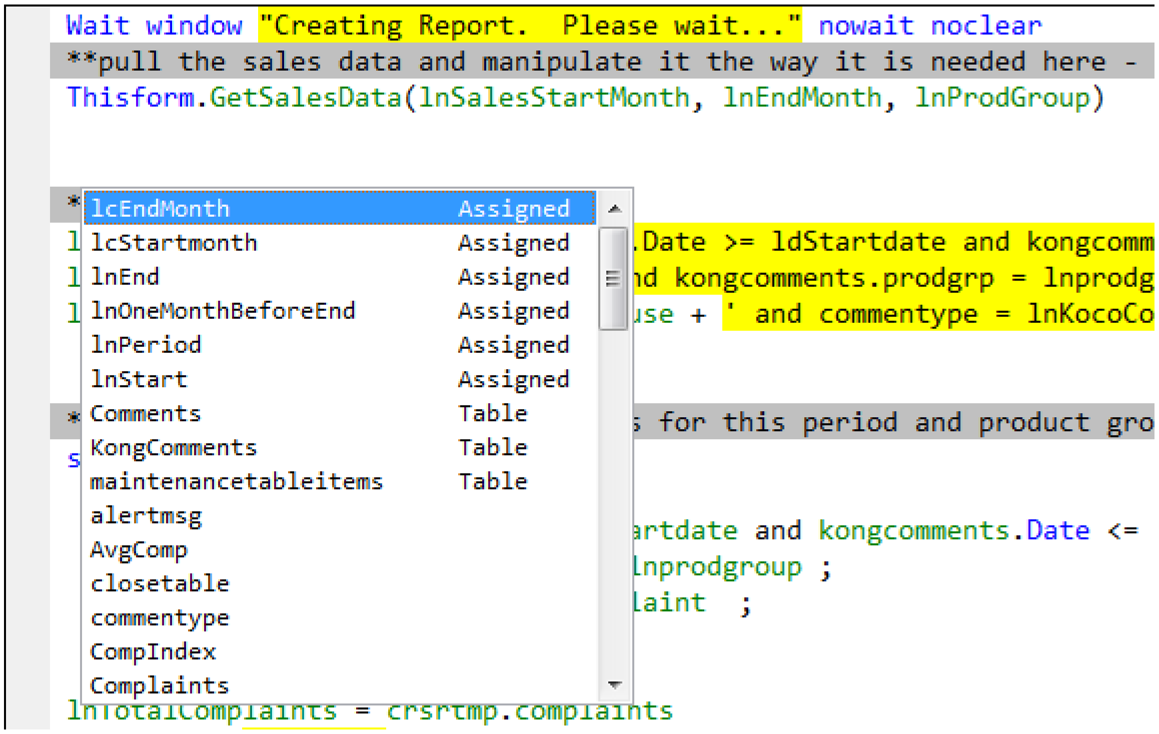
```
TempFiles
lcTempDirectory    Local
```

Finally, if you type in enough of a name to uniquely identify it (such as "New" in this example, which matches only "lcNewSourceFile"), the match is pasted in immediately without even showing the pop-up. When this becomes familiar, it is extremely handy, reducing both keystrokes and keying errors.

AutoComplete

Shows all names referenced in the current procedure, except for VFP Keywords. Thus, variables, field names, table names and aliases, procedures, objects, property and method names, and so on; for variables, it also indicates which are assigned but not appear in a LOCAL statement . AutoComplete has its own plug-in to allow you to change the names that are displayed or their order. *(Author's note: This is my favorite among all the lesser known tools and may be the tool I use most often).*

You can control whether an M-Dot is pasted in before any variable you select from this list by going to M-Dots on the "Options" page of the Configuration form.



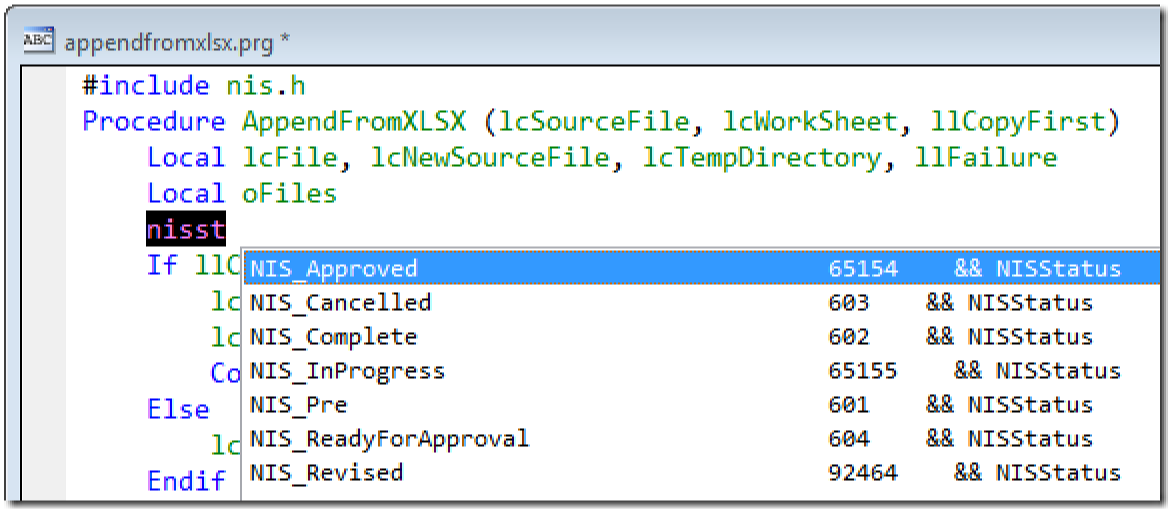
```
Wait window "Creating Report.  Please wait..." nowait noclear
**pull the sales data and manipulate it the way it is needed here -
Thisform.GetSalesData(lnSalesStartMonth, lnEndMonth, lnProdGroup)
```

```
lcEndMonth              Assigned
lcStartmonth            Assigned
lnEnd                   Assigned
lnOneMonthBeforeEnd     Assigned
lnPeriod                Assigned
lnStart                 Assigned
Comments                Table
KongComments            Table
maintenancetableitems   Table
alertmsg
AvgComp
closetable
commentype
CompIndex
Complaints
```

```
.Date >= ldStartdate and kongcomm
nd kongcomments.prodgrp = lnprodg
use + ' and commentype = lnKocoCo
s for this period and product gro
artdate and kongcomments.Date <=
lnprodgroup ;
laint  ;
lnTotalComplaints = crsrtmp.complaints
```

Dropdown Constants List

Shows all constants defined in the current procedure (#Define statements) as well as all constants defined in referenced #Include statements. The dropdown list shows not only the names of matching defined constants, but also their values and any comment on the same line. Furthermore, the matching is done against the comments as well. For instance, in the example below, entering "nisst" matches all the entries that have "NISStatus" (a field name) in the comment.



```
#include nis.h
Procedure AppendFromXLSX (lcSourceFile, lcWorkSheet, llCopyFirst)
    Local lcFile, lcNewSourceFile, lcTempDirectory, llFailure
    Local oFiles
    nisst
    If llC NIS_Approved                          65154    && NISStatus
        lc NIS_Cancelled                         603     && NISStatus
        lc NIS_Complete                          602     && NISStatus
        Co NIS_InProgress                        65155    && NISStatus
    Else   NIS_Pre                               601     && NISStatus
        lc NIS_ReadyForApproval                  604     && NISStatus
    Endif  NIS_Revised                           92464    && NISStatus
```

Dropdown Procedures List

Shows the names of all procedures and functions referenced in either the current project (if there is one open) or the current path. The original intent was to also show the parameters for each procedure or function, but this caused it to be so slow it was worthless. Even as it is, without parameters, it can be very slow.

Dropdown Table Names

Shows all tables in the current path. This tool also has its own plug-in to allow for searching in other folders.

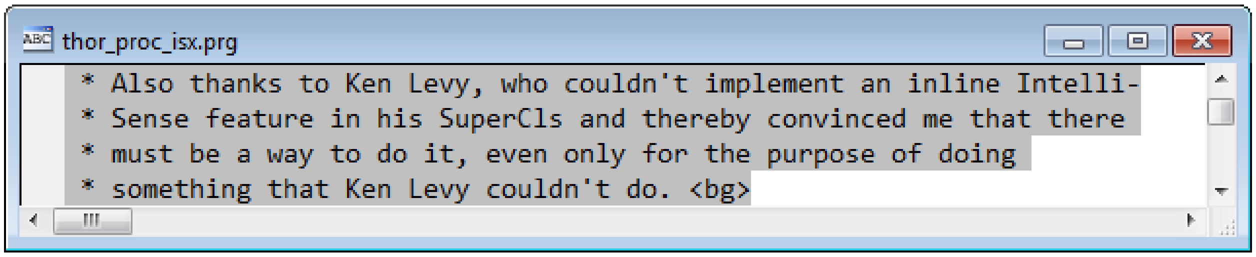
DBC Tables by !

Shows all tables found in a DBC. This one is not quite like the others, as it assigns an On Key Label for when you enter the '!'. Thus, you can only use this tool that way (not from menus or the Thor Toolbar, for instance)

Rhapsody on a Theme of Christof

IntellisenseX is a direct descendant of ISX.PRG, written by Christof Wollenhaupt (with help from many others), who started on it about fifteen years ago. All the basic design and structural work was done by Christof; all the work on IntellisenseX, then, merely extended the cases where a dropdown list could be provided.

By the way, Christof's code includes this interesting comment:



```
* Also thanks to Ken Levy, who couldn't implement an inline Intelli-
* Sense feature in his SuperCls and thereby convinced me that there
* must be a way to do it, even only for the purpose of doing
* something that Ken Levy couldn't do. <bg>
```

This project then is clearly an example of standing on the shoulders of giants.

Summary **

IntellisenseX provides drop-down lists for PEMs from objects and field names from tables for a wide range of objects, tables, and data objects not addressed by FoxPro's native Intellisense. The drop-down lists show additional information in extra column() and can be filtered using "match-anywhere" (just a $) to reduce the number of items in the list.

We'll start by showing how IntellisenseX is installed, show how it works duplicating native Intellisense for PEMs of objects in the code window and command window, and then into the murky area where it does so for those cases where native Intellisense might be expected to work, but does nothing. Then we will move onto new ground—uses in WITH/ENDWITH blocks, references to variables not yet defined with LOCAL ... AS ..., uses in PRG-based classes, and objects from an object factory.

Our attention will then turn to how it handles fields from tables (something ignored by native Intellisense outside of the command window). We will show how tables can be referenced in code windows, including tables that are not yet open; how tables can be accessed by their aliases when referenced in code (USE .... ALIAS ...,); how to see the fields for aliases used in SELECT statements; and how to see fields from a SQL Server database for use in SELECT statements.

Finally, we will delve into the use of the Alias Dictionary, which allows you to create application-wide aliases for both objects and tables. This simple table provides some remarkable and unexpected capabilities.

IntellisenseX is highly customizable and descriptions of the various options and plug-ins used by IntellisenseX will be interwoven into the session. IntellisenseX is delivered by and integrated with Thor. While installation of Thor will not be addressed during the session, there will be help available outside the session for anybody who needs help installing or using it.

You will learn:

How to access PEMs from objects (in SCXs, VCXs, and PRGs) that native Intellisense forgot

How to access lists of fields from native FoxPro tables, in SELECT statements for both native FoxPro tables and SQL Server tables, and from data objects

How to create and use system-wide aliases for objects and tables

How to use other related Thor tools to achieve auto-completion of names used in a procedure and to find #Define compiler constants

See the IntellisenseX Home Page for videos and ThorTWEeTs (#10 thru 22) for in depth discussions of particular aspects of IntellisenseX.

Finally, there is a place for you to address any questions or comments at the Thor Forum.

Biography

Jim Nelson spent the first thirty years of his professional life programming in APL, a long-since extinct programming language. For the last twenty of those years, he was the sole developer for a company that handled workers' compensation self-insurance. His claims handling system was the foundation on which the entire company was built. During those years, he had a reputation throughout the APL community as a builder of developer utilities not available within the APL language.

His VFP career began in 2003, working for the Kong Company in Golden, Colorado. He has worked full-time for them since then, acting as their software development department.

His involvement in VFPx projects (including Thor, IntellisenseX, Finder, PEM Editor, and FoxCharts) mirrors his long time interest in developing developers' tools.

Jim lives in Thousand Oaks, California, with his wife of 41 years.