What's Inside this Guide:

Discern Explorer®

Quick Reference Guide for Discern Visual Developer

CERNER

Table of Contents

Code Level

This tutorial is intended for users on the *Cerner Millennium*® Cumulative Support Package 2007.18 release of Discern Visual Developer (DiscernVisualDeveloper.exe), also referred to as DVDev. Much of the functionality discussed in this document is also available for earlier releases, but there are significant differences between 2007.18 and some of the earlier versions.

A basic understanding DVDev and the *Discern Explorer*® (CCL) programming language to create programs and reports is assumed.

General Information

The information in this document is intended to provide you with a high-level overview of functionality available in DVDev. The tips provided will improve your abilities for developing programs and troubleshooting issues.

Keyboard Shortcuts

Discern Visual Developer (DiscernVisualDeveloper.exe) contains the keyboard shortcuts shown below. In addition, you will see labels throughout the application that include an underlined letter, such as View. If the label is attached to a box, you can press ALT+<UNDERLINED LETTER> to move the cursor to that box. If the label is on a button or menu, you can press ALT+<UNDERLINED LETTER> to take that action. For example, pressing ALT+V opens the View menu.

To	Press
Opening a menu	ALT + the first letter of the menu name
Open the File menu	ALT+F
Open the Edit menu	ALT+E
Open the View menu	ALT+V
Open the Build menu	ALT+B
Open the Tools menu	ALT+T
Open the Reports menu	ALT+R
Open the Window menu	ALT+W
Open the Help menu	ALT+H

File menu

Opens a new file	CTRL+N
Opens an existing file	CTRL+O
Save	CTRL+S
Print	CTRL+P
Export	CTRL+E
Import	CTRL+I

Edit menu

Undo	CTRL+Z
Redo	CTRL+Y
Cut	CTRL+X
Copy	CTRL+C
Paste	CTRL+V
Delete	DELETE
Select All	CTRL+A
Replace	CTRL+H
Find	CTRL+F
Make selection lowercase	CTRL+U
Make selection uppercase	SHIFT+CTRL+U
Comment Block	CTRL+;
Uncomment Block	CTRL+SHIFT+;
Access Discern Explorer Help	F1 while the (Code Editor) window is open
Bookmarks	
Add/Delete a bookmark	CTRL+F2
Toggle between bookmarks	F2
Toggle between bookmarks backward	SHIFT+F2
Clear All Bookmarks	CTRL+SHIFT+F2

View menu

Full Screen	SHIFT+ALT+ENTER

Build menu

Run <program_name>	CTRL+F5
Include/Compile	CTRL+F7
Run prompt program	CTRL+R
Run ad hoc query	CTRL+Q
Include/Compile Debug Mode	CTRL+SHIFT+F7

Tools menu

Opens Prompt Builder	CTRL+SHIFT+H
Opens Query Builder	CTRL+SHIFT+Q
Opens Layout Builder	CTRL_SHIFT+L
Opens Record Builder	CTRL+SHIFT+1
Code Lookup	CTRL+F1
Add Variable Builder	CTRL+D
Add Subroutine Builder	CTRL+B
Add Code Value builder	CTRL+1
Macros	
Record Quick Macro	CTRL+SHIFT+R
Play Quick Macro	CTRL+SHIFT+P

Output Viewer

Custom Zoom (Report Output window)	CTRL+M

Code Editor

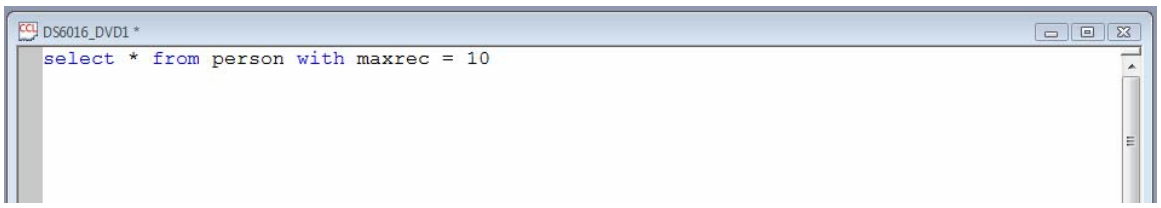
Display tool tip window from within a function's parenthesis	CTRL+SHIFT+SPACE
Find next	F3
Highlight sections of text	CTRL+SHIFT+ARROW KEYS
Listing/Results	CTRL+L

Move between statement keywords	CTRL+} when placed next a keyword
Move between open and close parenthesis	
	CTRL+] when placed next to a parenthesis
List of functions	CTRL+SPACE BAR
Toggle between all open files	CTRL+TAB

Ad Hoc Queries

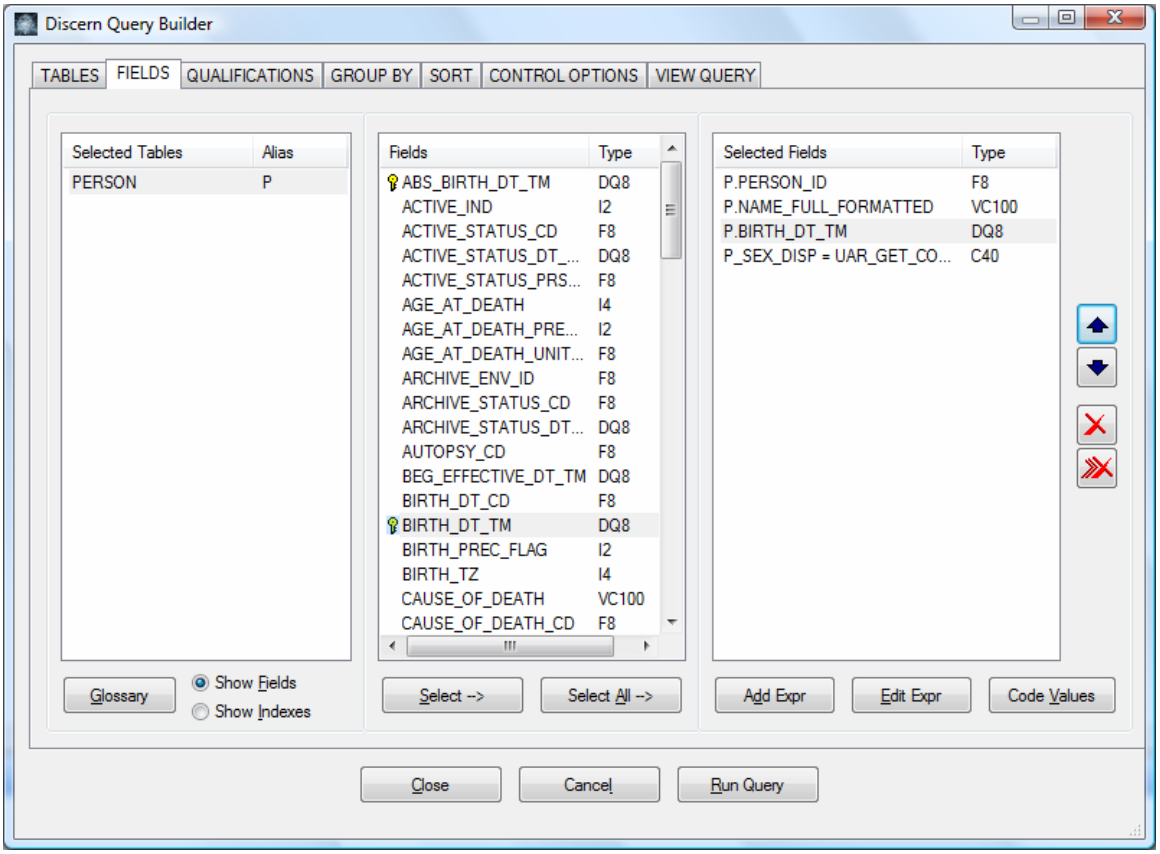
Creating and Running Ad Hoc Queries

Use DVDev (DiscernVisualDeveloper.exe) to create and run ad hoc queries. An ad hoc query can be created using two different methods. The first method is to manually enter your query in the Code Editor window of a blank file.

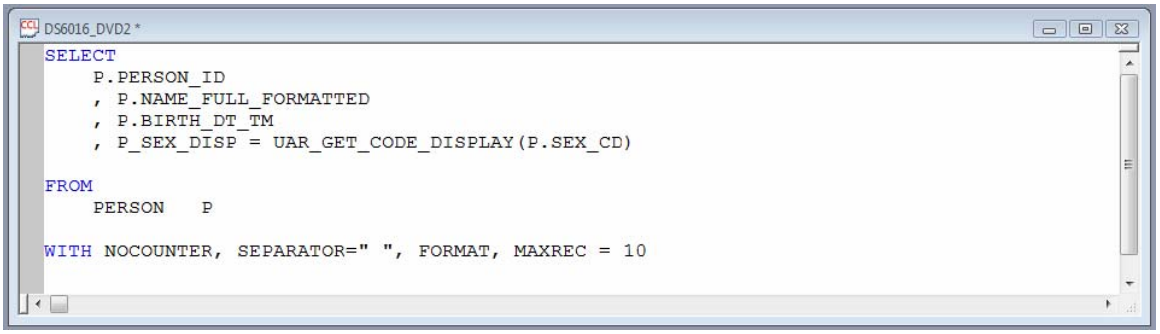


To execute the query, press CTRL+Q or select Run Ad Hoc Query from the Build menu.

The second method of creating an Ad Hoc Query method is to use the Query Builder to create the query.



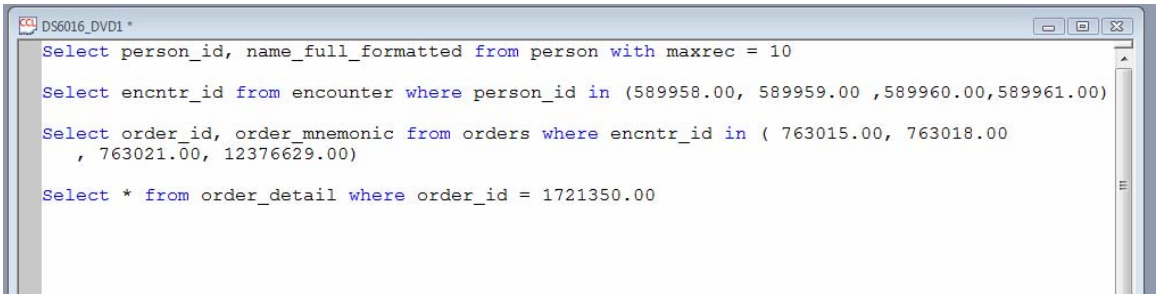
To execute a query created from the Query Builder, click the Run Query button. Another way to execute the query is to exit out of the Query Builder, which writes the query to a file. The following is an example of a query written to a file that was created by the Query Builder:



To execute the query from the file, press CTRL+Q or select Run Ad Hoc Query from the Build menu.

Executing a Single Query in a File Containing Multiple Queries

If you have multiple queries in one file, you can single out which query you want to execute.

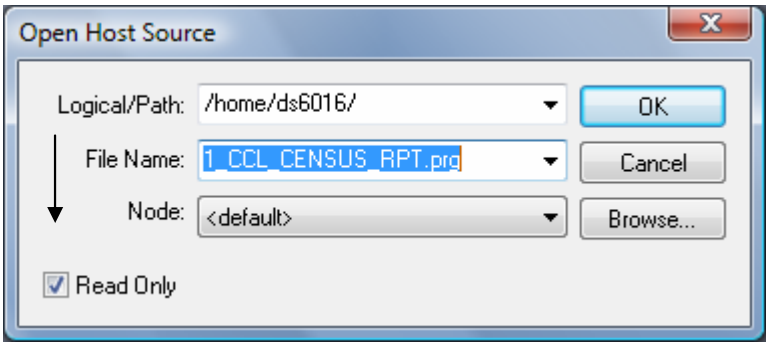


In order to execute the first query, the focus of the cursor is placed anywhere in the first query, such as on the word SELECT, or highlight the query to be executed. Then press CTRL+Q.

Options

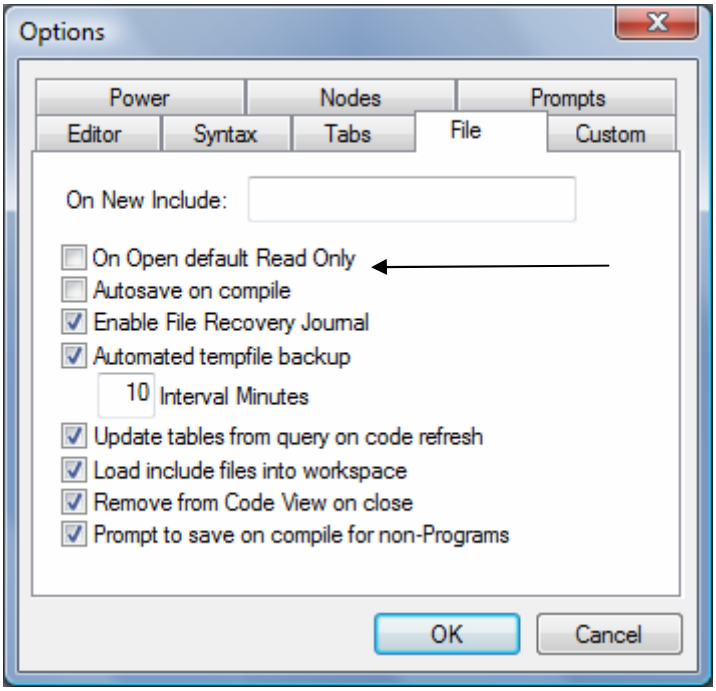
Turn Off the Default Read Only Option when Opening Source Code Files

When you open a file for editing, by default the Read Only preference is set.



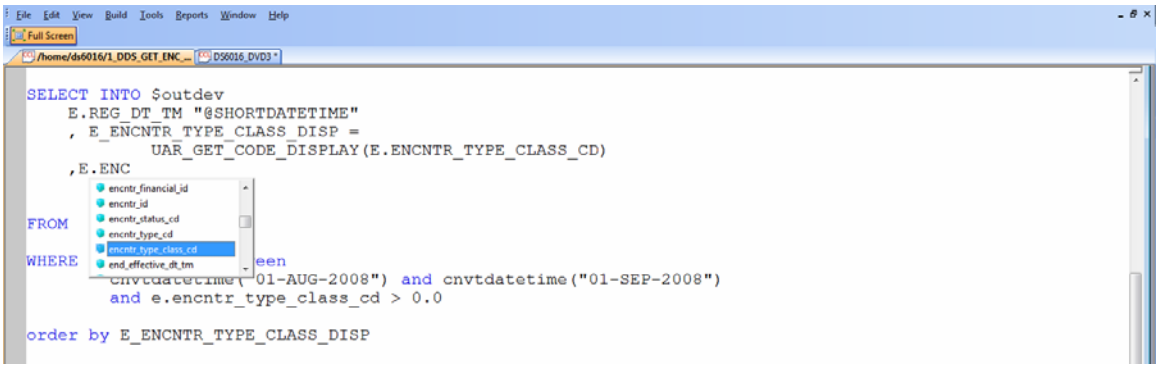
In order to edit a file, you must deselect the *Read Only* option each time you open the file unless you change the system to turn off the default Read Only option.

To turn off the Read Only option, deselect the *On Open default Read Only* option located on the View menu > Options and the click the File tab.

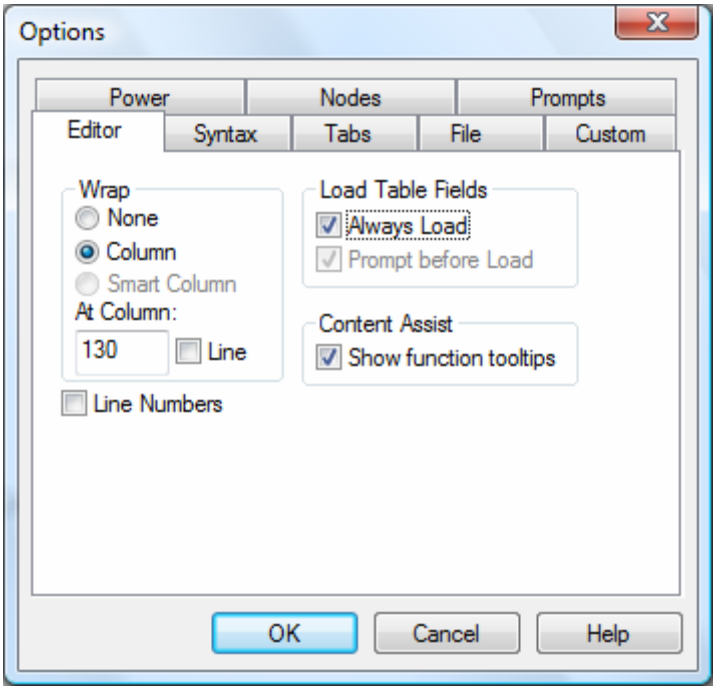


Always Load Table Fields Option

DVDev (DiscernVisualDeveloper.exe) has an option that automatically displays a list of fields from a table which displays when you type the alias of the table in the source code. You can choose the field from the list instead of typing the entire field name. The list of fields can be further limited by typing more of the field name.

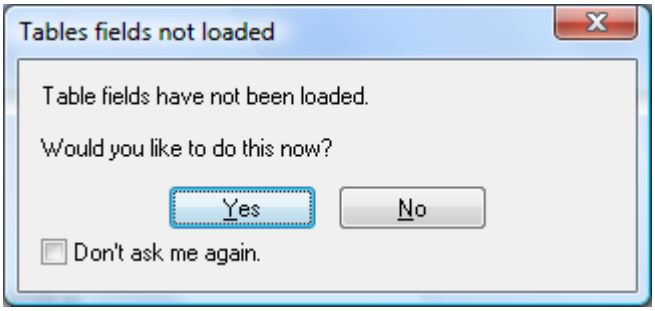


The *Load Table Fields* option is located on the Editor tab from the View menu > Options.



When the *Always Load* option is selected, the list of fields for a table will always load as long as the alias for the table has been defined in query.

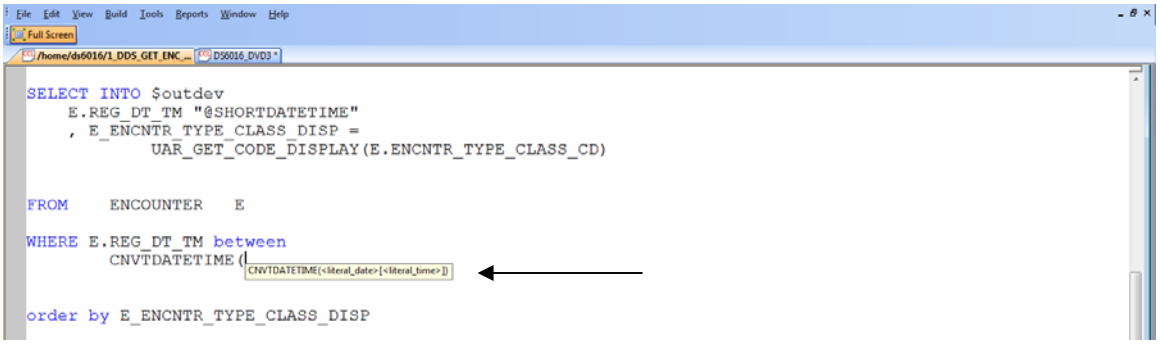
When the *Prompt before Load* option is selected, you will be prompted before any new table is loaded as long as the alias for the table has been defined with the following message:



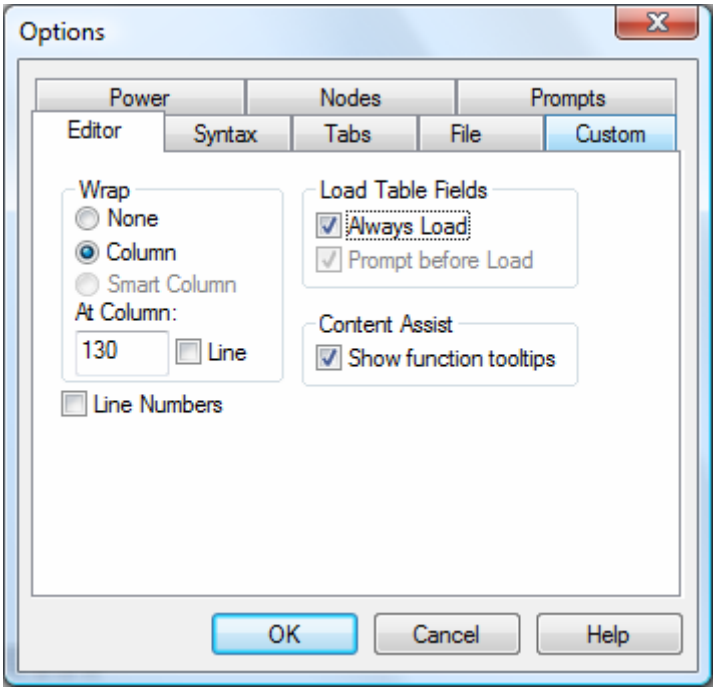
Selecting the *Don't ask me again* option automatically deselects both Load Table Fields options (they can also be manually deselected). The field list for the table will not be displayed under any circumstance.

Show Function Tool Tips Window

When typing functions or commands in your source code, you can choose to show a window that displays the parameters that are expected for that particular command by selecting the *Show functions tooltips* option.



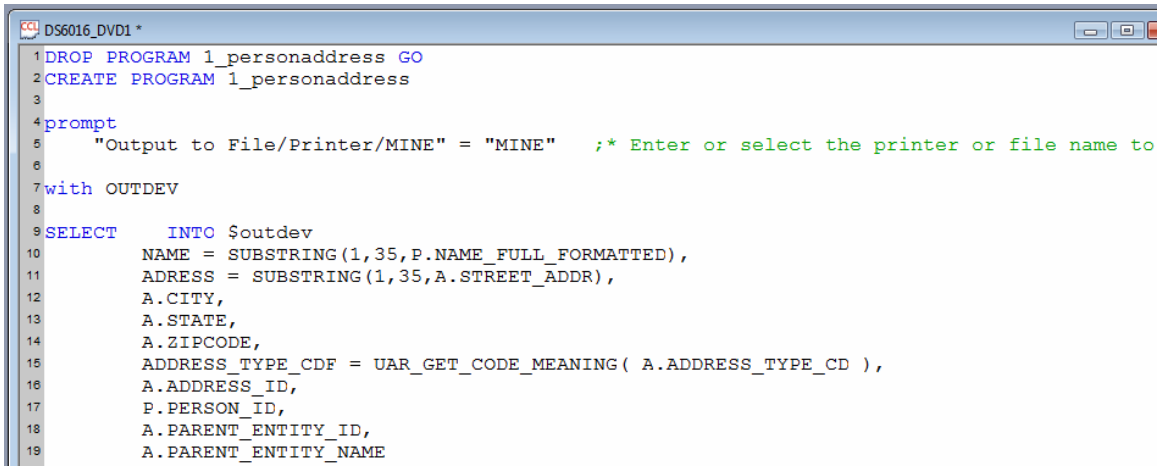
The *Show function tooltips* option is located on the Editor tab from the View menu > Options.



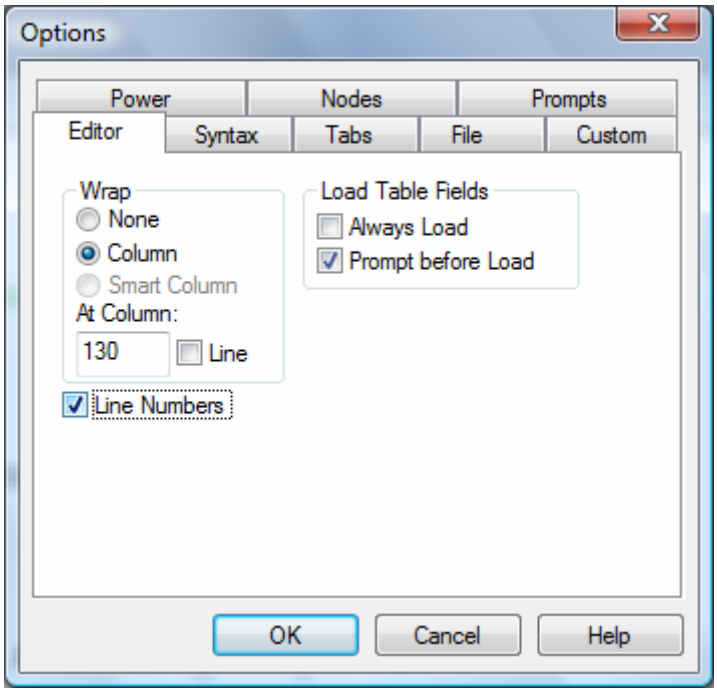
When the *Show function tooltips* option is selected, the syntax box is displayed after the open parenthesis for the function has been typed. If the window closes, and you want to re-display the window, press CTRL+SHIFT+SPACE and the tooltip will re-display.

Displaying Line Numbers

When developing code, it can be helpful to display the line numbers in the Code Editor window.

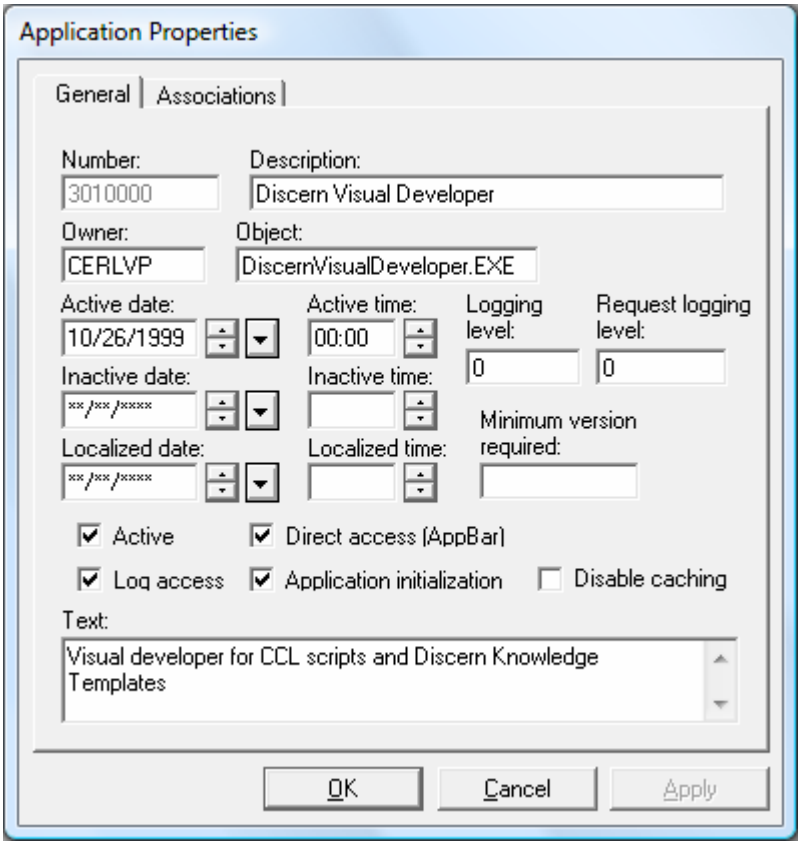


To turn line numbers on, from the Options dialog box listed under the View menu, check the option for Line Numbers in the Editor tab.



Keep Options Settings

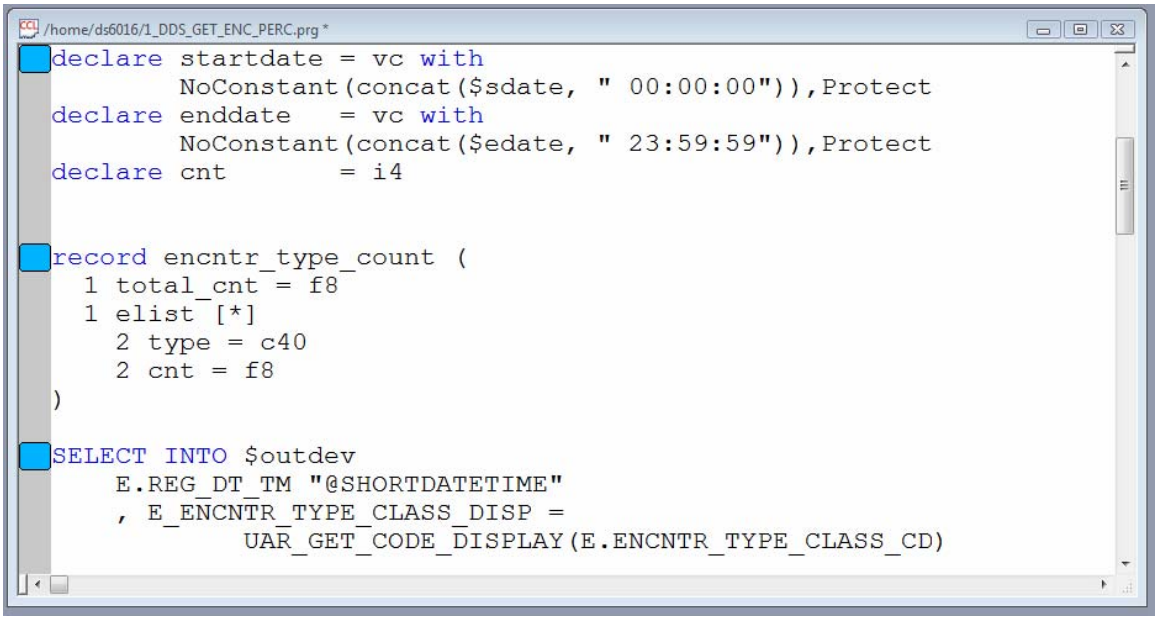
In order to keep the Options selected for a specific user after the application has been closed and reopened, the Application Initialization option must be selected in Appreg.exe for app 3010000 (Discern Visual Developer).



Code Editor Window

Bookmarks

Use Bookmarks to visually mark the start of specified sections of source code. Using Bookmarks is most helpful when working with a file that has a large amount of source code and needing to reference different sections in the code.



To create the bookmark, place the cursor on the line where you want the bookmark to be and press CTRL+F2. There is not a limit on how many bookmarks you can choose. However, the bookmarks do not persist with the source and will not be displayed the next time the source is opened.

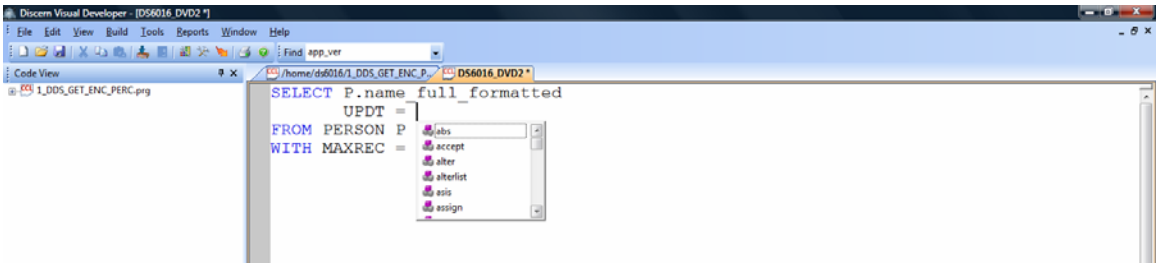
Use F2 to toggle between the different bookmarks.

To clear a single bookmark, place the cursor on the line and press CTRL+F2.

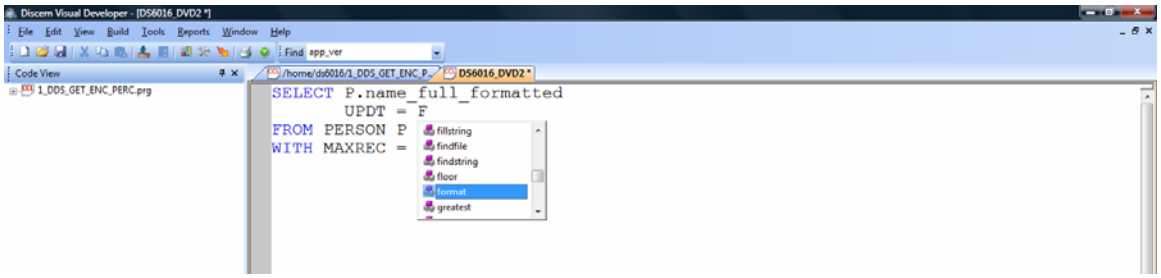
To clear all bookmarks, press CTRL+SHIFT+F2.

Display a Function List

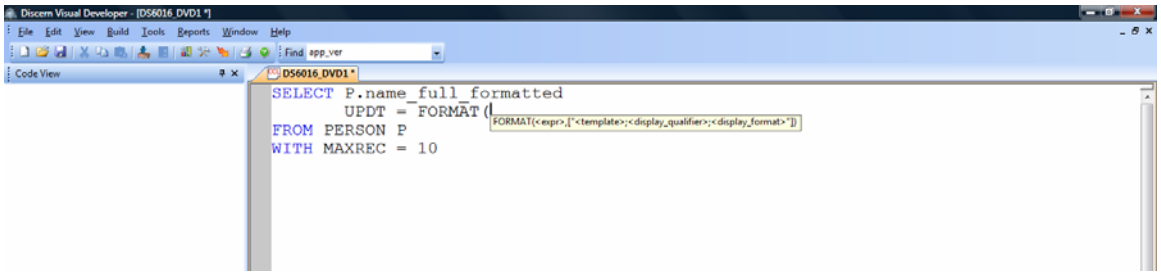
You can choose to select a function from a drop down list instead of typing a function.



Press CTRL+SPACE to show a list of functions to choose from. You can narrow down the list further by typing more of the function name you want to use; for example, entering **F** will show all of the functions that start with F.

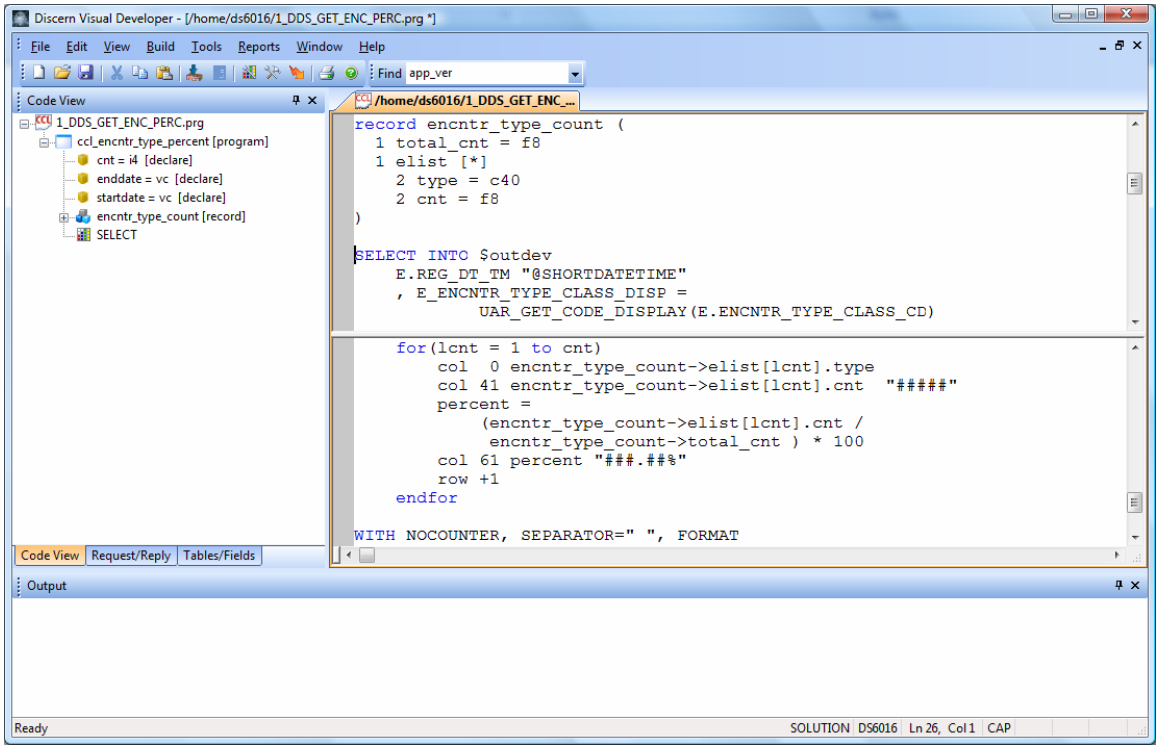


Select the function you want to use, such as FORMAT. If the Tool Tip option is set, the syntax window for the function displays.

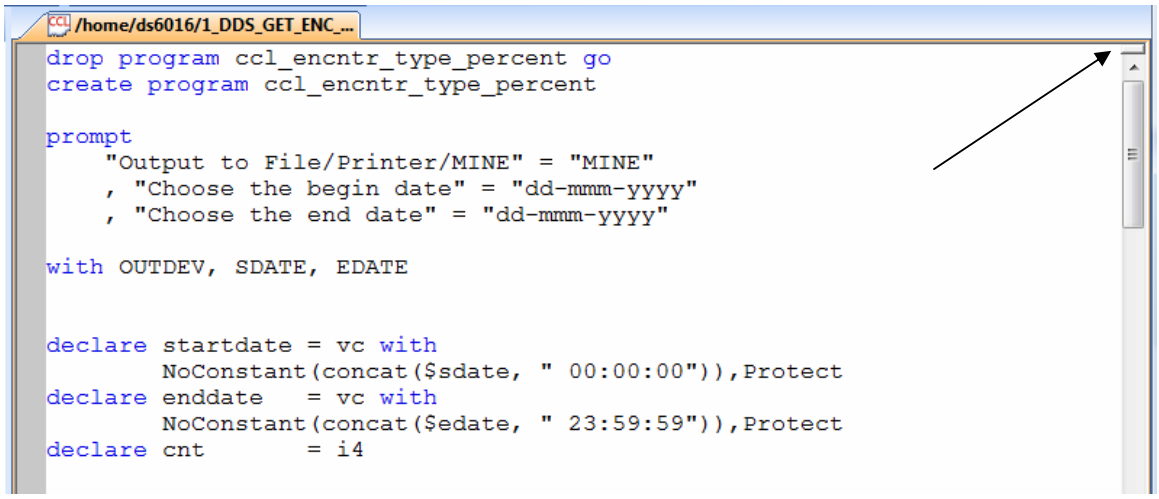


Splitting the Screen on the Code Editor Window

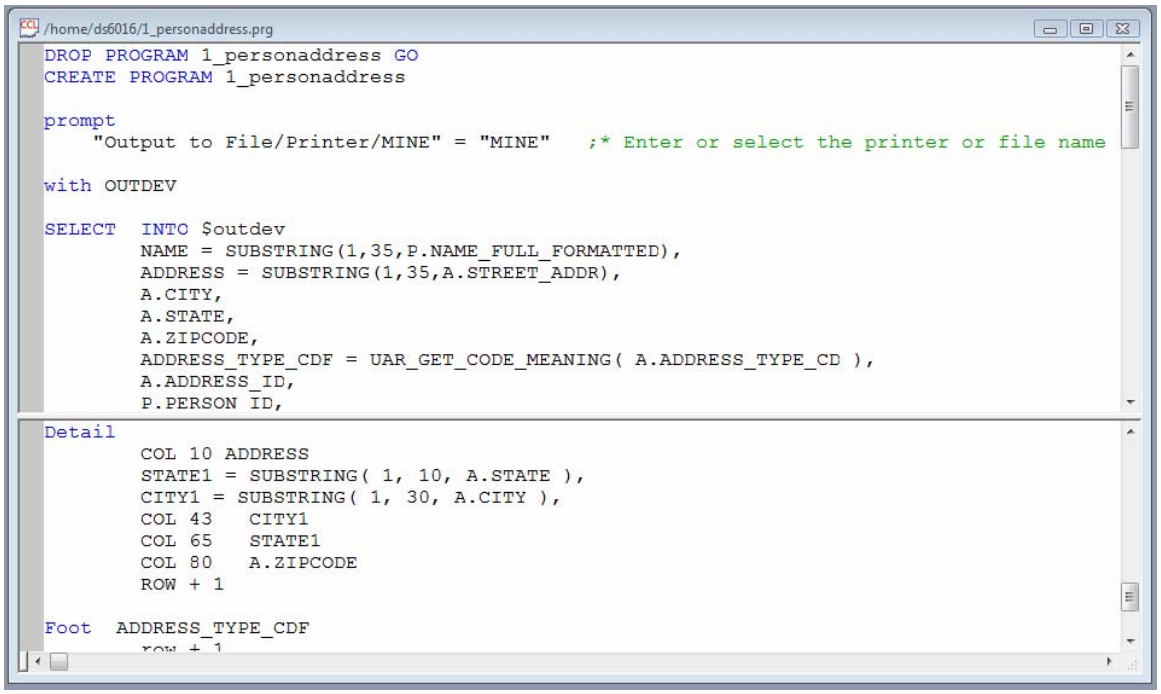
When working with a large source file, it can be helpful to split the screen on the Editor window into two horizontal panes so you can view two sections of the file simultaneously.



In the Code Editor window, the split screen bar is located on the top right hand side, as shown by the arrow below.



To split the screen, place the cursor on the split screen bar. When the cursor changes to a double line icon, drag the bar down to the location you want in the Code Editor window. You can also double-click the screen bar to divide the display in to equal halves.

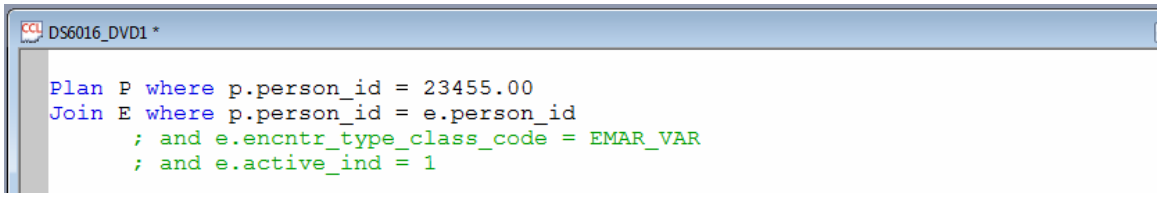


To remove the split screen view, drag the split screen bar back to the top of the window, or double-click on the split screen bar.

Commenting a Block of Code

When testing and debugging your code, it can be helpful to comment or uncomment certain sections of code so that specific lines of code are ignored by the compiler and treated as a comment instead of a command.

If you are only needing to comment out 1 or 2 lines of code, manually entering the semicolon (;) or the exclamation point (!) to the left of the line may be the easiest method to use.



However, if you need to comment out an entire section of code, highlight the section of code and press CTRL+;. This shortcut places a semicolon on multiple lines of code.

```
DS6016_DVD1 *
    select into "nl:"
            p.birth_dt_tm
            ,tz_ind = validate(p.birth_tz,-1)    ;031
    from person p
    where p.person_id = sub_personid
    detail
        if (tz_ind = -1)
;               if (cnvtupper(keyword) = "BIRTHDATE")
;                   str->replace = format(p.birth_dt_tm, "dd-MMM-yyyy;3;")
;               else
;                   if (ekssub->parse_ind)    ;034
;                       str->replace = format(p.birth_dt_tm, "dd-MMM-yyyy hh:mm;;q")
;                   else
;                       str->replace = trim(format(p.birth_dt_tm, "@LONGDATETIME;;q"),3)
;                   endif
;               endif
                if (not((Validate(curtimezone, "Z") = "Z"))
                    str->replace = concat(str->replace," ",curtimezone)
                endif
```

To uncomment a block of code, highlight the code you want to uncomment and press CTRL+SHIFT+;.

Locating Matching IF/ENDIF, FOR/ENDFOR Commands and Parenthesis

When writing and debugging your code, it helps to be able to locate and jump back and forth between matching IF/ENDIF, FOR/ENDFOR commands and matching open and close parenthesis.

```
DS6016_DVD4
FOR (i = colon to non_alpha )

   IF (substring(2,8,order_val) = "volume")

    set order_val_cnt = order_val_cnt = 1

   ENDIF

ENDFOR
```

By placing the cursor to the left of the FOR command and pressing CTRL+], the cursor jumps to the matching ENDFOR command in the file.

By placing the cursor to the left of the IF command and pressing CTRL+], the cursor jumps to the matching ENDIF command in the file.

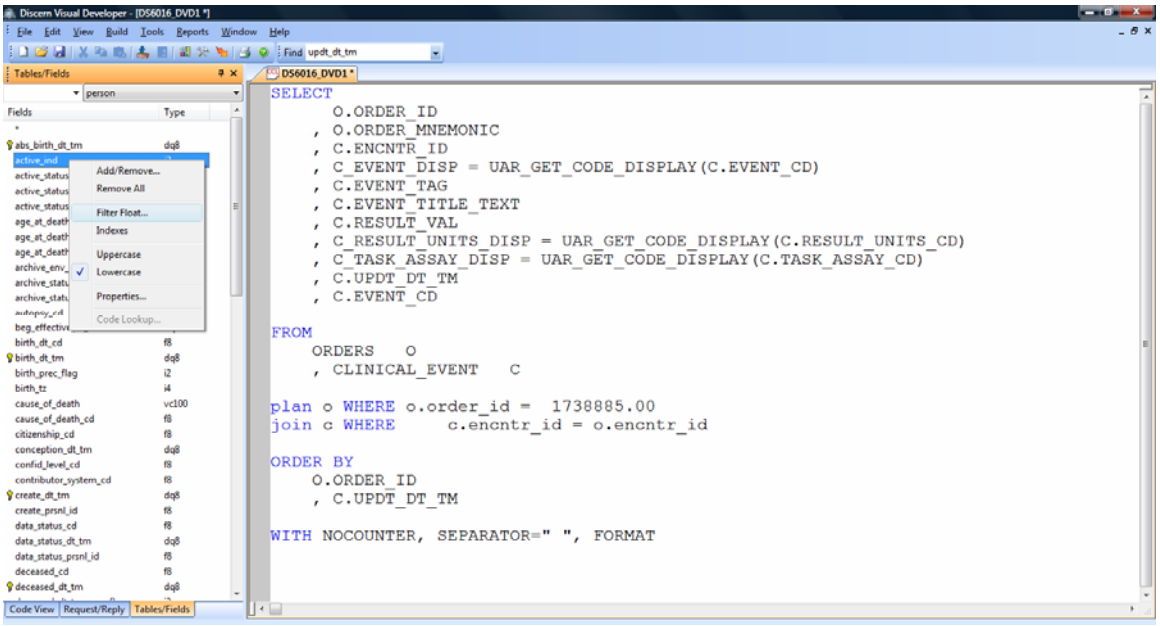
By placing the cursor to the left of a parenthesis, and pressing CTRL+], the cursor jumps to the matching closing parenthesis in the file.

If a matching item does not exist, when you press CTRL+], your cursor does not move.

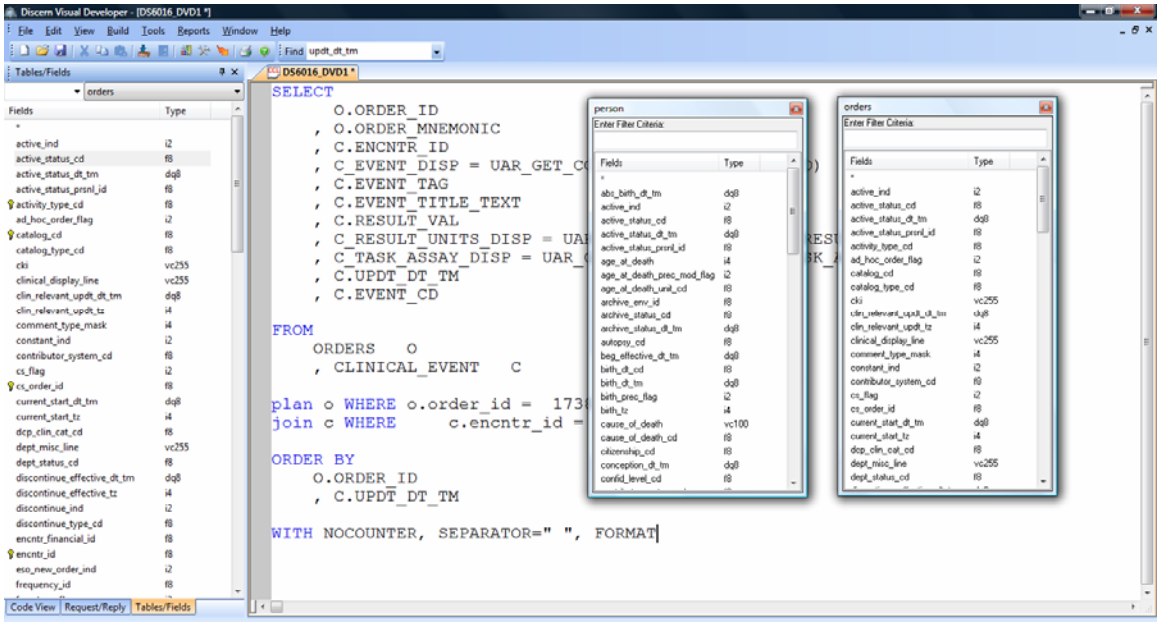
Workspace

Filter Float

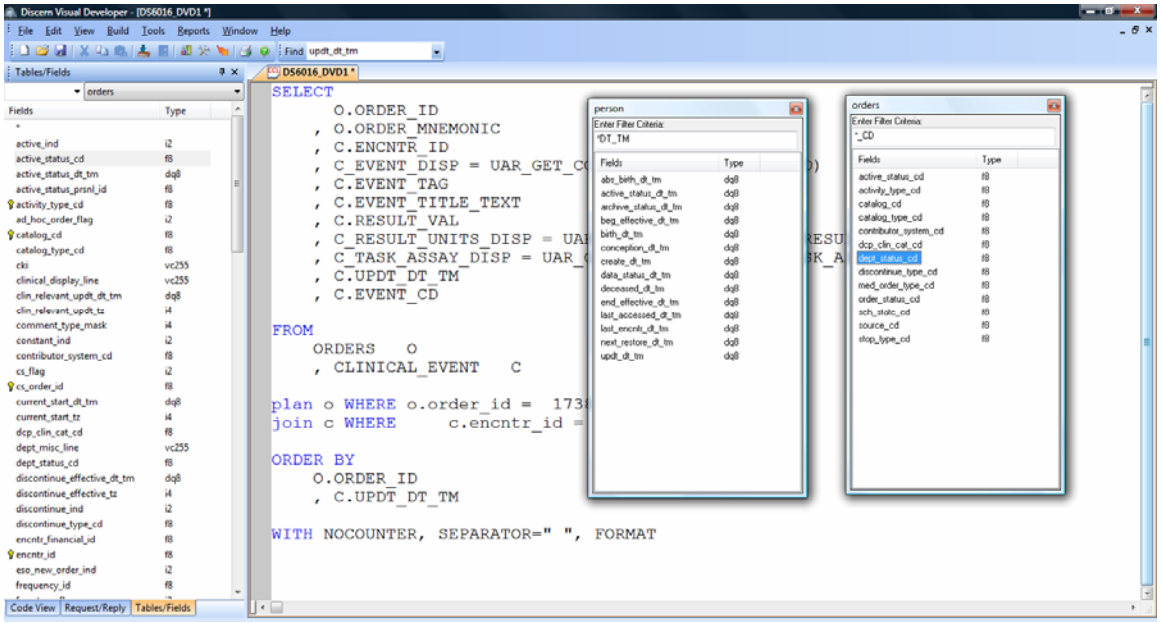
Right-clicking anywhere on the Tables/Fields tab opens a context menu from which the Filter Float option can be selected.



With the Filter Float selected an independent window for the specific table is created. The window can be moved anywhere on your workspace.



The Filter Float also allows you to filter fields for display on the list by text. For example, display all Date fields (*DT_TM) or all Code Value Fields (*_CD).



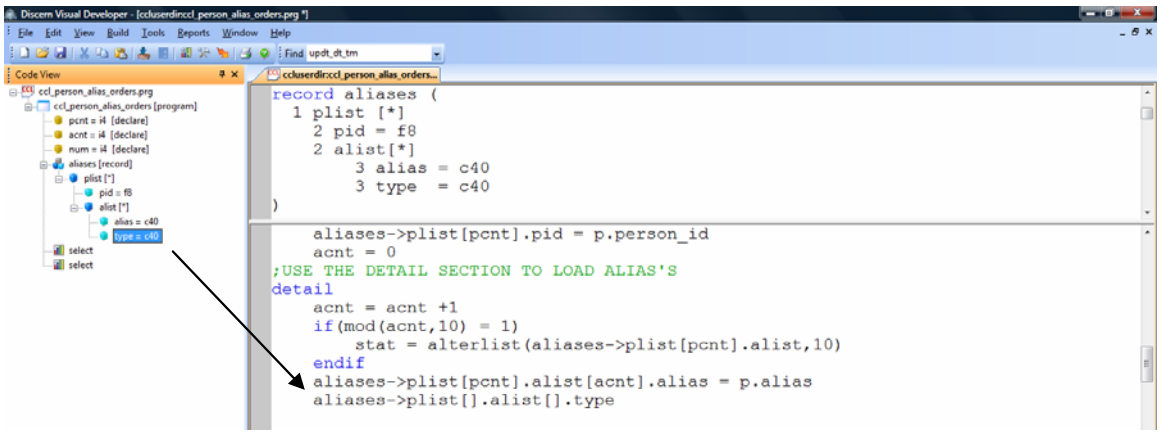
To remove the Filter Float, close the window.

Drag and Drop Items to a File

You can drag and drop items from the Code View, Request/Reply or from the Tables/Fields tabs to your source code file.

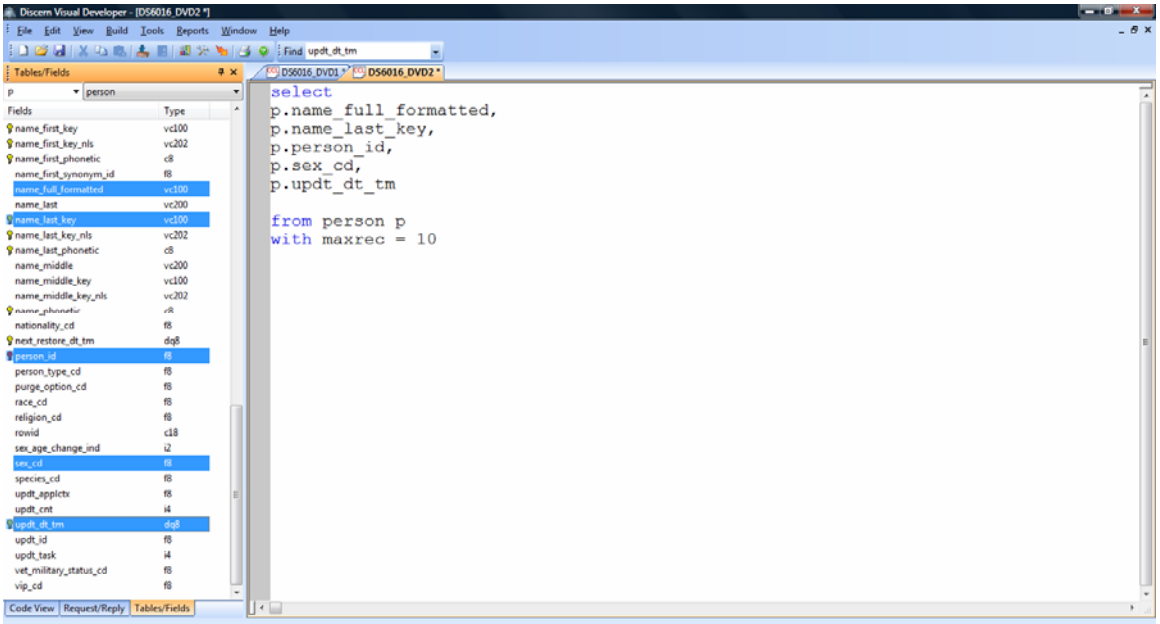
Code View

Use the Code View of a program to drag record items from the tree view to your source file. The Code View shows an outline of a program once the commands have been compiled at least once. In the example below, the record structure item TYPE was dragged from the Code View pane to the source file. Using this method creates the proper reference to the record item from the structure.



Fields Tab

With tables loaded in the Tables Tab, you can drag a column or multiple columns to the source code file. To select multiple columns at the same time, press CTRL while selecting the fields. The selected fields will be highlighted and you can then drag the fields to the source file. If the alias for the table is loaded, then the column name will be preceded with alias.

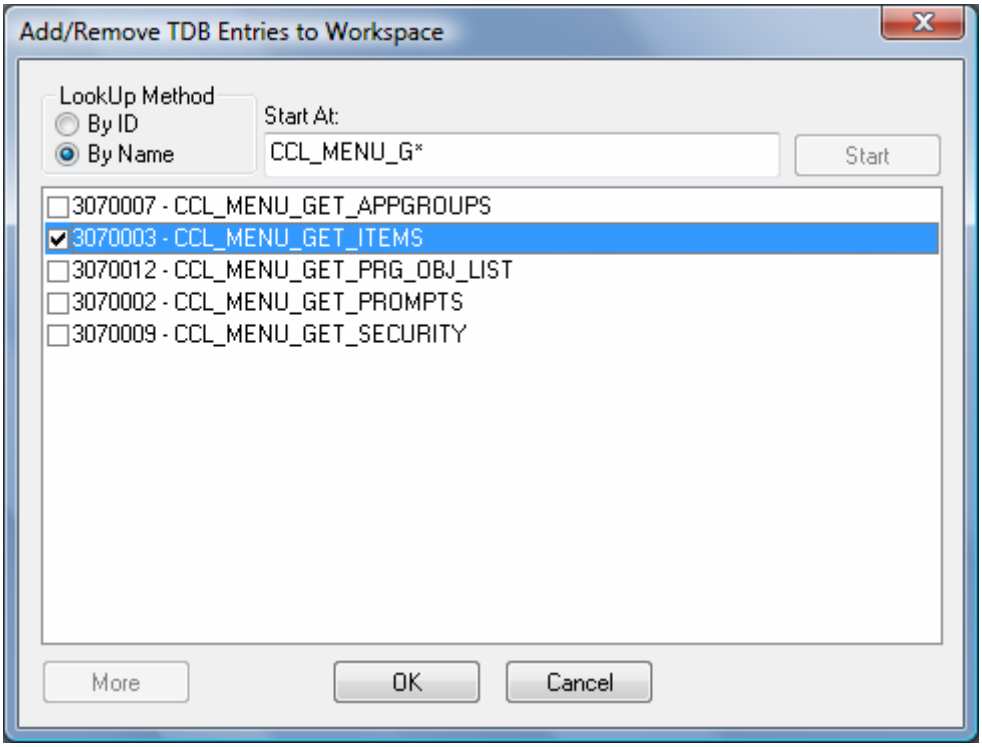


A single field is moved by dragging a specific item to the source file.

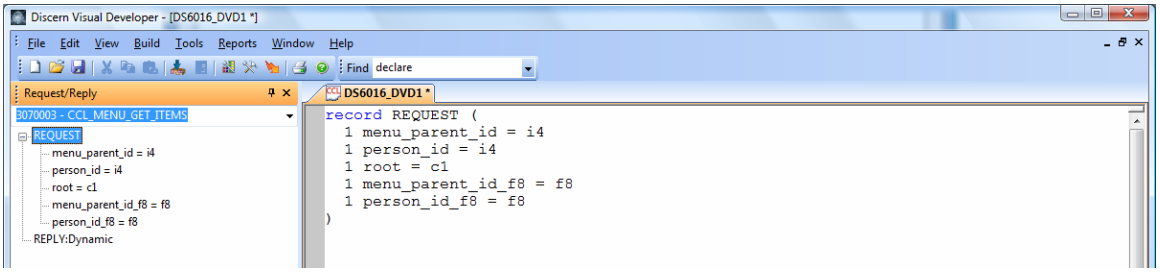
Request/Reply Tab

When Requests or Reply structures are loaded into the Request/Reply tab, you can drag a structure from the workspace to a source code file.

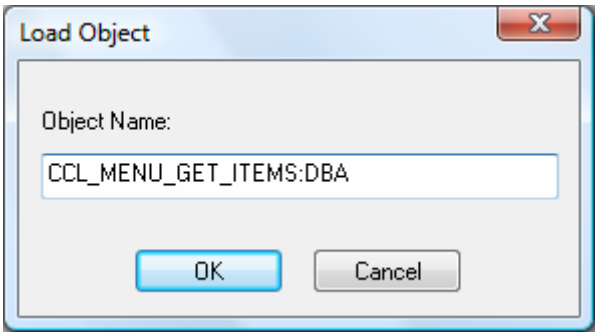
To load a request or reply structure into the Request/Reply tab, from the View menu select Add/Remove > Request/Reply. You can then select a TDB entry by name or ID.



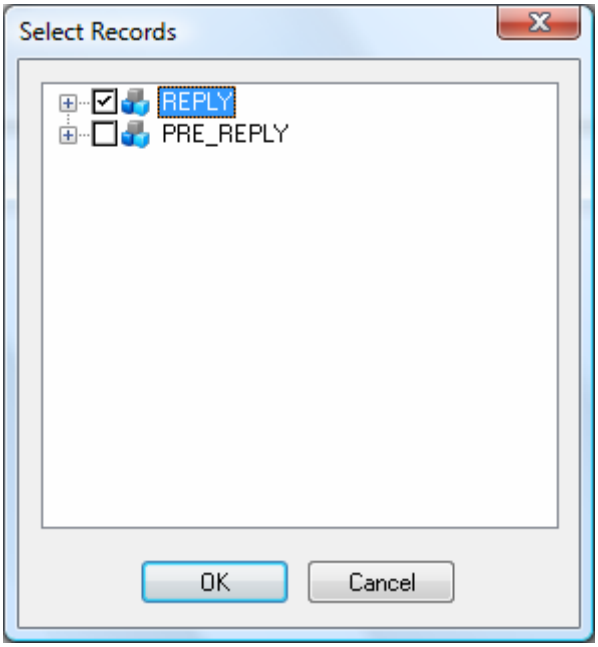
Once the TDB entry is loaded in the Request/Reply tab, you can drag the entire definition to your source file by placing focus on the Record name (for example, REQUEST) and dragging it to the source file.



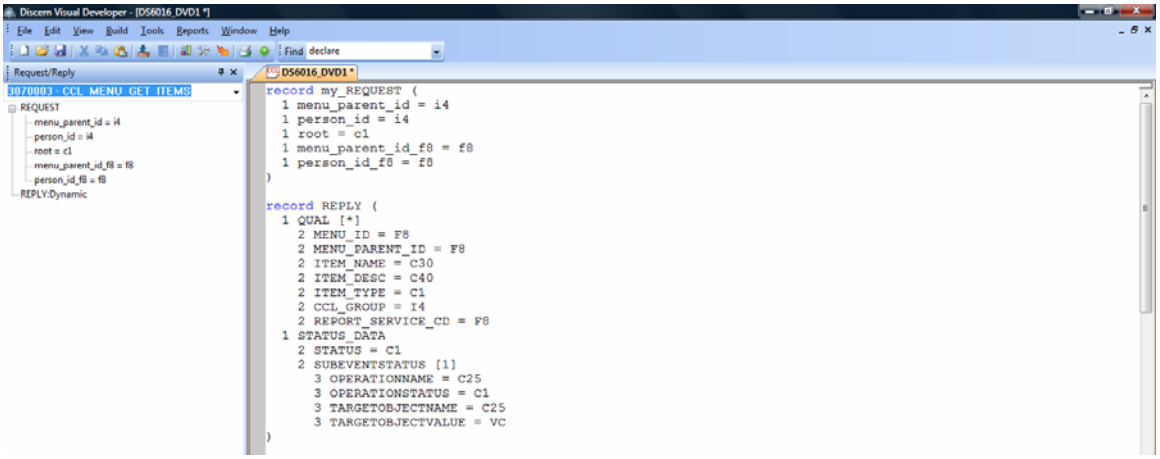
Reply structures defined as Dynamic cannot be dragged to the source file because they are not defined as a TDB entry. A dynamic reply structure is defined by the program that uses the TDB entry. To have access to a reply structure that is defined as dynamic, from the Tools menu select Record Builder > Load Object, and load the object name that has the defined structure.



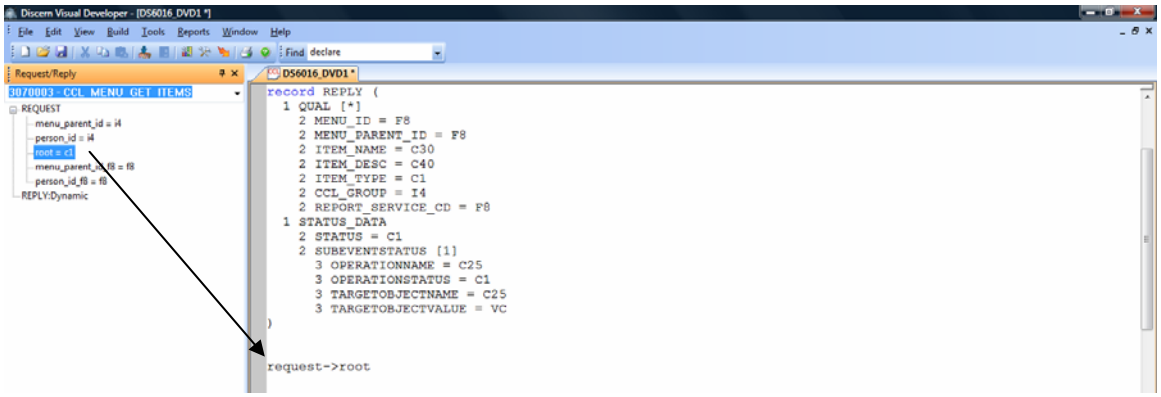
Loading the object allows the REPLY structure to be available to select.



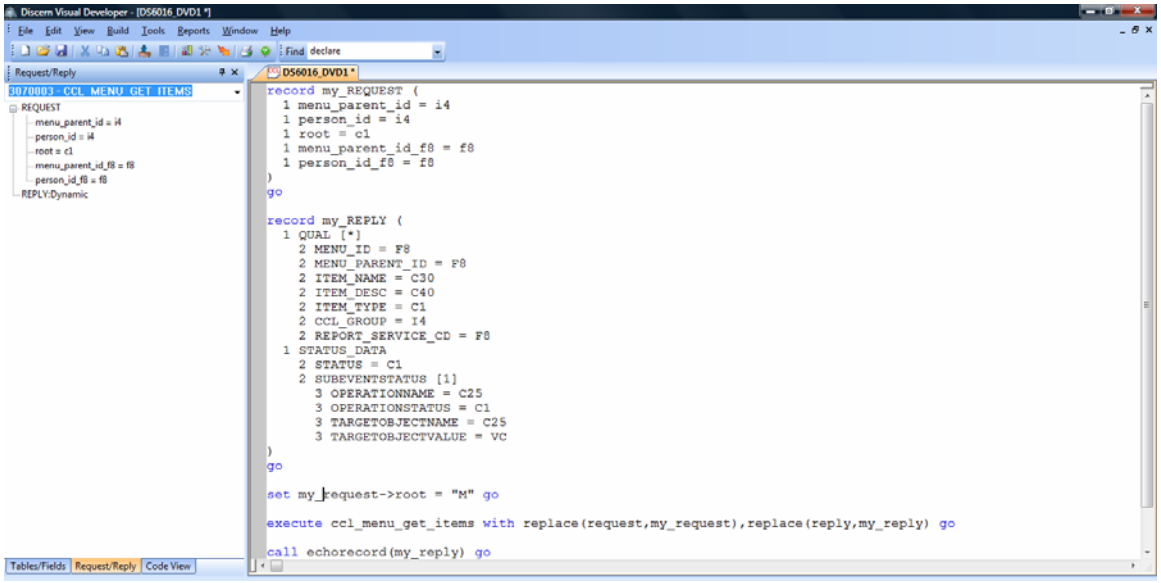
When you click OK, the REPLY structure is directly placed in the source file.



You can select single items from the structure by clicking and dragging them to the source file. Using this method builds the correct syntax for referencing the item.

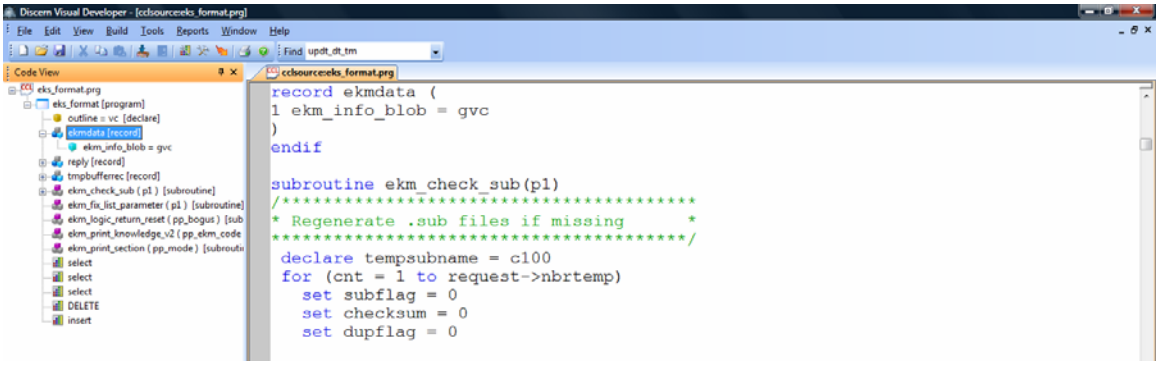


The ability to drag and drop items from a record structure allows you to easily reference the entire definition or items in a structure and to significantly reduce typos. Use this method when developing programs or when creating test scripts for troubleshooting purposes as the following example demonstrates. Review the section *Creating a Test File* for more details on creating a test file.

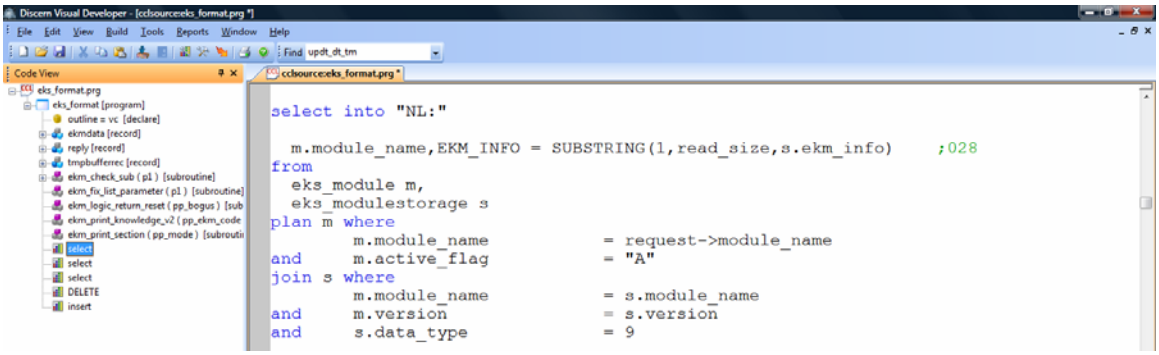


Use Code View to get to different parts of the Program

Double-click any item listed in the Code View tree to navigate to that part of the source code. In the following example, double-clicking *ekmdata[record]* from the Code View tree causes DVDev to navigate to where the record structure is defined in the source file.



Double-clicking on the first select listed in the Code View tree causes DVDev to navigate to that query in the source file.



Troubleshooting using Compiling/Executing

Finding Compile Errors

When a program is included using DVDev and syntax errors are found by the compiler, you can easily identify the area where the problem has occurred.

The following program has a missing WHERE clause in the JOIN statement in the qualification section:

```
drop program 1_person_address go
create program 1_person_address

prompt
    "output to file/printer/mine" = "mine"   ;* enter or select the printer or file
with outdev

select into $outdev
    p.person_id
    , p.name_full_formatted
    , a.address_id
    , a.street_addr

from
    person   p  , address   a

plan p  where p.person_id > 0

join a  p.person_id =  a.parent_entity_id
       and a.parent_entity_name = "person"

with nocounter, separator=" ", format, maxrec = 100
end
go
```

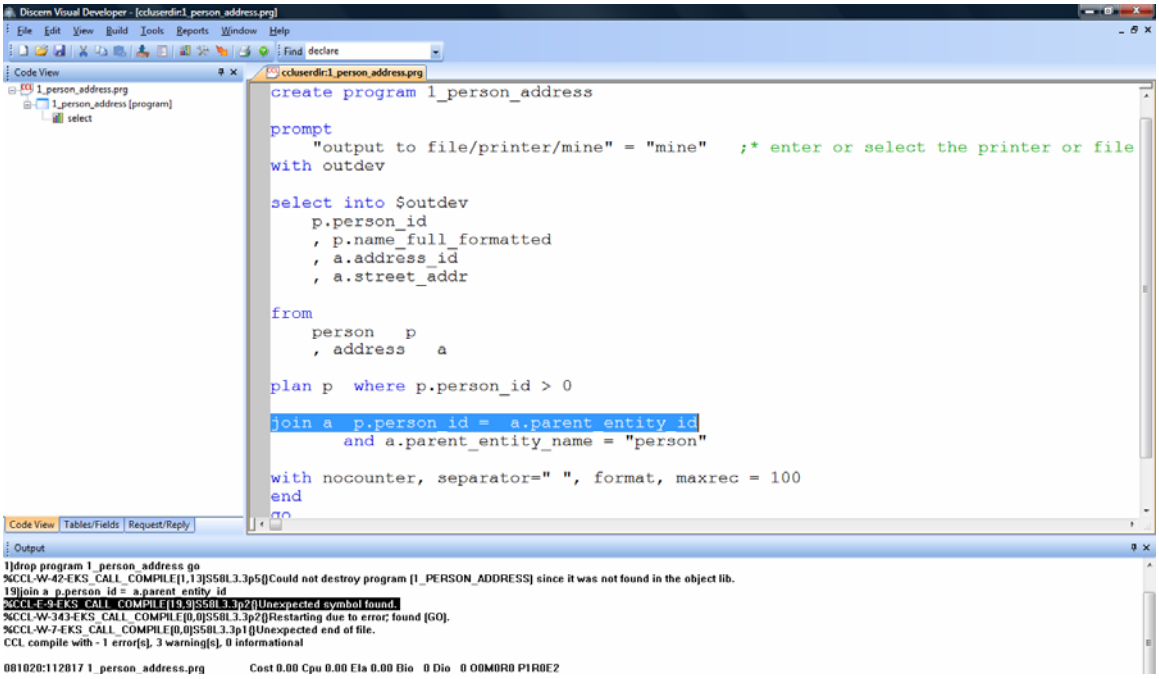
When the program is included using DVDev, an error is displayed in the Output dialog box:

%CCL-E-9-EKS_CALL_COMPILE(19,9)S58L3.3p2{}Unexpected symbol found

```
1]drop program 1_person_address go
%CCL-W-42-EKS_CALL_COMPILE(1,13)S58L3.3p5{}Could not destroy program (1_PERSON_ADDRESS) since it was not found in the object lib.
19]join a  p.person_id =  a.parent_entity_id
%CCL-E-9-EKS_CALL_COMPILE(19,9)S58L3.3p2{}Unexpected symbol found.
%CCL-W-343-EKS_CALL_COMPILE(0,0)S58L3.3p2{}Restarting due to error; found (GO).
%CCL-W-7-EKS_CALL_COMPILE(0,0)S58L3.3p1{}Unexpected end of file.
CCL compile with - 1 error(s). 3 warning(s). 0 informational

081020:112817 1_person_address.prg        Cost 0.00 Cpu 0.00 Ela 0.00 Bio  0 Dio  0 00M0R0 P1R0E2
```

Double-clicking the first CCL-E listed in the messages highlights a line in the source code where the compiler was no longer able to continue processing.

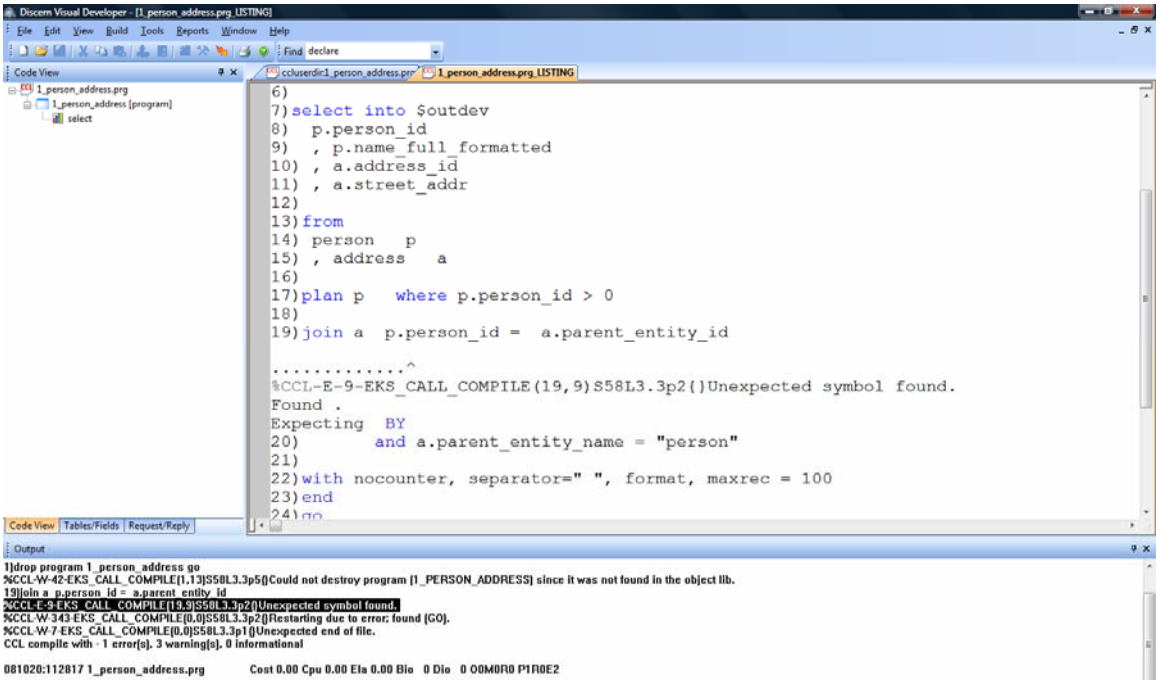


The error message lists the line number and column where the compiler failed:

%CCL-E-EKS_CALL_COMPILE(**19,9**)S58L3.3p2{}Unexpected symbol found.

In the above error message, the problem occurred on line 19, column 9. The line number directly corresponds to the line number in the Listing file where the compiler failed.

Accessing the Listing file (Ctrl_L) shows the how the line number in the error message corresponds to the line number in the listing file.



Testing/Troubleshooting Specific Commands

While working in DVDev (DiscernVisualDeveloper.exe), it can be helpful to use a blank file to test specific commands or sets of commands. Compiling a file in DVDev causes all of the commands in the file to be executed. Error messages and the output of call echo() or call echorecord() commands are written to the Listing file. For example, suppose you wanted to test how the T(<num>) display qualifier effects the display of significant digits to the right of a decimal point. You could test this functionality by adding the display option to an existing query or creating a simple test query. However, it is often easier to see the effects if you format some hard coded values.

```
call echo(format(1.0,"####.##")) go
call echo(format(1.0,"####.##;T(1)")) go
call echo(format(1.0,"####.##;T(2)")) go
call echo(format(1.2,"####.##;T(1)")) go
call echo(format(1.2,"####.##;T(2)")) go
call echo(format(1.200200,"####.######")) go
call echo(format(1.200200,"####.######;T(1)")) go
call echo(format(1.200200,"####.######;T(2)")) go
call echo(format(1.000000,"####.######")) go
call echo(format(1.000000,"####.######;T(1)")) go
call echo(format(1.000000,"####.######;T(2)")) go
```

To execute the above commands, the commands are placed in a blank file in DVDev and then executed by selecting Include/Compile from the Build menu. The output of the call echo() commands are displayed in the listing file which can be accessed by selecting Listing from the View menu or pressing Ctrl+L.

```
1)call echo(format(1.0,"####.##")) go
   1.00
1)call echo(format(1.0,"####.##;T(1)")) go
   1
1)call echo(format(1.0,"####.##;T(2)")) go
   1.0
1)call echo(format(1.2,"####.##;T(1)")) go
   1.2
1)call echo(format(1.2,"####.##;T(2)")) go
   1.2
1)call echo(format(1.200200,"####.######")) go
   1.200200
1)call echo(format(1.200200,"####.######;T(1)")) go
   1.2002
1)call echo(format(1.200200,"####.######;T(2)")) go
   1.2002
1)call echo(format(1.000000,"####.######")) go
   1.000000
1)call echo(format(1.000000,"####.######;T(1)")) go
   1
1)call echo(format(1.000000,"####.######;T(2)")) go
   1.0
```

Testing Whether a Variable is Populated

When troubleshooting program issues, you sometimes need to determine what information is stored in a variable. For example, suppose there is a variable in a program called BUN_VAR that is being used in a qualification that does not appear to be pulling back any data:

```
WHERE O.Catalog_CD = BUN_VAR
```

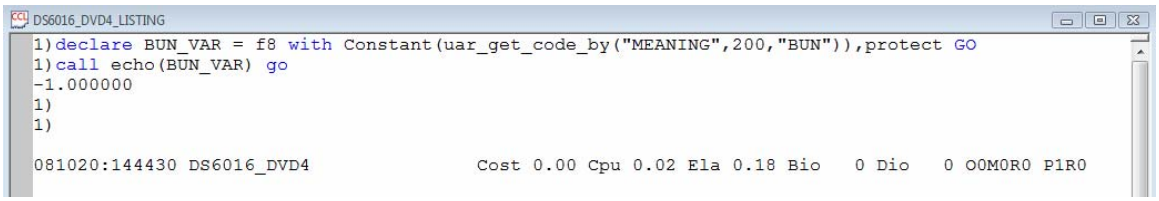
The BUN_VAR variable used in the above qualification is populated using the UAR_GET_CODE_BY( ) function. The following line is used in the program to populate the variable:

```
declare BUN_VAR = f8 with
Constant(uar_get_code_by("MEANING",200,"BUN")),protect
```

You want to validate that the variable is properly populated. One easy method is to copy the code that creates and populates the variable, paste it in a blank file in DVDev. Place the GO command at the end of the statement. Then use the Call Echo ( ) command passing in the variable you are checking. Each command must have the GO command at the end. The commands are executed by selecting Include/Compile from the Build menu, or pressing Ctrl+F7. For example, the following Declare ( ) command populates the BUN_VAR variable. Then the Call Echo ( ) is used to show the contents of the variable which can be seen in the listing file.

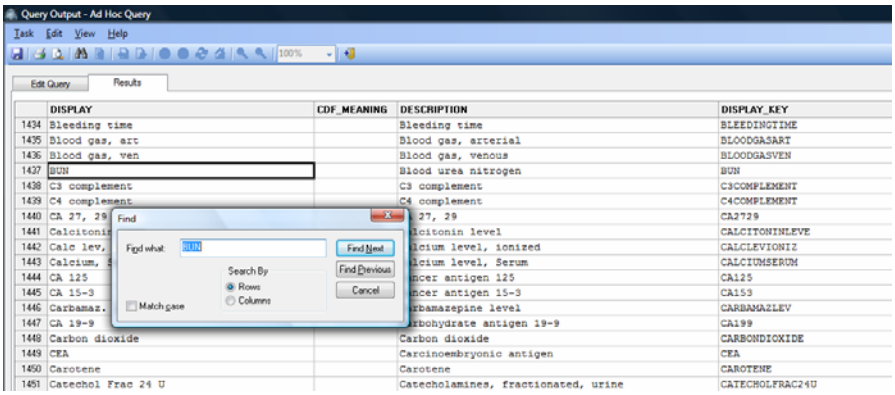
```
declare BUN_VAR = f8 with
Constant(uar_get_code_by("MEANING",200,"BUN")),protect GO
call echo(BUN_VAR) go
```

After executing the commands from the blank file, the Listing file shows the contents of the call echo() command.



In this case we would expect to see a code_value representing a BUN orderable from the call echo( ) command, but instead there is -1, which indicates that the UAR function failed. To understand why the UAR function failed, first look at what information is stored for that Code Set.

The CDF_MEANING column in Code Set 200 is not populated. This information can be seen by using the Code Lookup from the Tools menu and placing a 200 in the Code Set Number parameter.



The DISPLAY_KEY column is populated and is a good choice to use in the UAR_GET_CODE_BY() function. The DISPLAY_KEY will be more consistent because all values stored in this field will always be upper-case. The parameter in the UAR_GET_CODE_BY() function is changed from MEANING to DISPLAYKEY as shown below.

```
declare BUN_VAR = f8 with
Constant(uar_get_code_by("DISPLAYKEY",200,"BUN")),protect GO
call echo(BUN_VAR) go
```

After executing the commands (Ctrl+F7), the Listing file shows the content of the call echo() command. The code_value shown in this Listing may not be the same as the code_value in your environment.

```
DS6016_DVD4_LISTING
1)declare BUN_VAR = f8 with Constant(uar_get_code_by("DISPLAYKEY",200,"BUN")),protect GO
1)call echo(BUN_VAR) go
672363.000000
1)

081020:152111 DS6016_DVD4              Cost 0.00 Cpu 0.00 Ela 0.17 Bio   0 Dio   0 O0M0R0 P1R0
```

Use CCL_RPT_AUDIT_LOG to Troubleshoot Prompt Programs

Information is stored about the execution of programs that have an associated prompt form and have been executed through Visual Explorer, Explorer Menu, DVDev or any other application that uses DiscernOutputViewer.ocx. The audit report program, CCL_RPT_AUDIT_LOG, shows information about the execute statement that is created by the prompt form and passed to Discern Explorer. The information from the report can be used to perform the following tasks:

- See the parameters passed to a program.
- Compare the parameters passed to a program when the program runs successfully and when it does not run successfully.
- Aid in debugging program issues.
- Execute a prompt program with the parameters listed from the report.
- Study the performance of the execution of a program.
- Audit users who are executing report.

See and Compare parameters passed to a program

The following prompt program accepts input from the user to locate records that were updated on the PERSON table within the date range entered at the prompts. The program has three prompts defined. The first prompt is for an output device; the next two prompts are date prompts. In order to demonstrate how to use CCL_RPT_AUDIT LOG to compare parameters passed to a program, we have set up the following program to fail. The second prompt has been assigned the incorrect date/time formatting option and will send the date to the program in a format that cannot be accepted by the CNVTDATETIME() option.

```
drop program 1_ccl_adt_log go
create program 1_ccl_adt_log

prompt
  "Output to File/Printer/MINE" = "MINE"
  , "Enter a Start Date:" = "CURDATE"
  , "Enter an End Date:" = "CURDATE"

with OUTDEV, SDATE, EDATE
```

```
select into $OUTDEV
p.person_id,
p.name_full_formatted
from person p
where p.updt_dt_tm between cnvtdatetime($SDate) and cnvtdatetime($EDate)

with format, separator = " ", maxrec = 100

end
go
```

The second prompt form is the Start date that the user enters at run-time and that is defined as following:

General Tab:

- Prompt Display: Enter the Start date:
- Prompt Name: SDATE
- Control Type: Date Time
- Prompt Type: String

Date/Time Tab:

- Command Line Format: Type, mm/dd/yy (note that this date/time formatting option cannot be passed to the CNVTDATETIME( ) function in this form.)

The third prompt form is the End date that the user enters at run-time and that is defined as following:

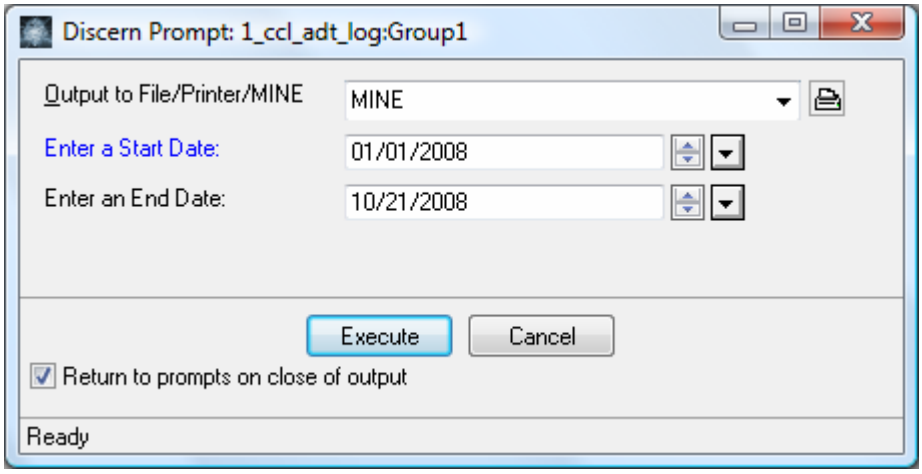
General Tab:

- Prompt Display: Enter the End date:
- Prompt Name: EDATE
- Control Type: Date Time
- Prompt Type: String

Date/Time Tab:

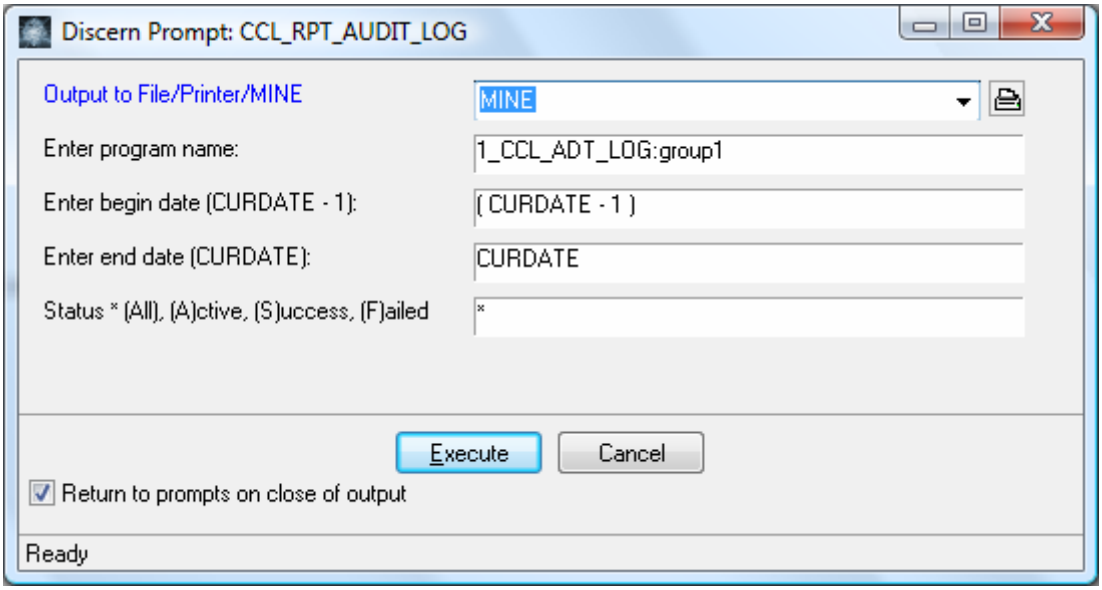
- Command Line Format: DD-MMM-YYYY (note that this date/time formatting option can be passed to the CNVTDATETIME( ) function in this form.)

When the program is executed, the prompt form is displayed. The user must enter a valid date range where data is expected to be returned. The following prompt form shows a date range entered by the user:



When the user selects Execute, the Output Viewer is displayed with no results because nothing qualifies for this report regardless of the date range entered at the prompt. Use the program CCL_RPT_AUDIT_LOG to help understand what parameters were passed to the program. CCL_RPT_AUDIT_LOG is executed by selecting Run Prompt Program from the Build menu.

Note: If the object you are researching is a Group1, you must add **:group1** to the object name.



The following is an example report from CCL_RPT_AUDIT_LOG:

```
      Report name(s): 1_CCL_ADT_L                          Discern Explorer Report Audit
      Audit status:   ALL REPORT STATUSES
      Start date:    20-OCT-2008 00:00:00.00
      End date:      21-OCT-2008 23:59:59.00

Object type:  Status:   Elapsed time:  Date/time:        #Records:  User:          Params:

Program:   1_CCL_ADT_LOG:GROUP1
  QUERY       SUCCESS   < 1 Second     10/21/2008 11:18:00     -2   Plumber, Joe    "MINE", "01/01/2008", "21-OCT-2008"
```

The report shows the program executed one time successfully passing the parameters “MINE”,”01/01/2008”,”21-OCT-2008”. Notice the format of the first date/time prompt is different from the second date/time prompt. The query accepts the date in the format designated from the prompt form and passes it to the program. The dates are referenced in the qualification which uses the CNVTDATETIME function to convert the literal dates to a date/time value:

where p.updt_dt_tm between cnvtdatetime($Sdate) and cnvtdatetime($Edate)

The CNVTDATETIME() function looks for the literal date to be formatted as "DD-MMM-YYYY". The function cannot accept a literal date in the form of “MM/DD/YYYY” and is returning a null value for the second parameter. To fix this issue, you could change the qualification in the program to convert the date from the form to a format that can be passed to the CNVTDATETIME() function:

where p.updt_dt_tm between cnvtdatetime(cnvtdate2($Sdate),0)
and cnvtdatetime($Edate)

However, it may be just as easy to keep the qualification as is and change the way the date is passed to the program by modifying the Command Line Format in the prompt form to **“DD-MM-YYYY”**.

With the Command Line Format changed to “DD-MM-YYYY” in the prompt form, and with the program executed again, the results are now returned. The following report from CCL_RPT_AUDIT_LOG reflects both times the program was executed.

```
      Report name(s): 1_CCL_ADT_L                          Discern Explorer Report Audit
      Audit status:   ALL REPORT STATUSES
      Start date:    20-OCT-2008 00:00:00.00
      End date:      21-OCT-2008 23:59:59.00

Object type:  Status:   Elapsed time:  Date/time:        #Records:  User:          Params:

Program:   1_CCL_ADT_LOG:GROUP1
  QUERY       SUCCESS   < 1 Second     10/21/2008 12:20:14    100   Plumber, Joe   "MINE", "01-JAN-2008", "21-OCT-2008"
  QUERY       SUCCESS   < 1 Second     07/29/2008 11:18:00     -2   Plumber, Joe   "MINE", "07/29/2007", "29-JUL-2009"
```

The two rows that start with QUERY represent the two times the program was executed. The most recent execution of the program is the first row. This report indicates that the most recent run was successful and ran in less than 1 second at 10/21/2008 12:20:14, retrieving 100 records. The user who executed the program was Joe Plumber and he passed in the parameters as listed in the last column. The formats of the dates passed to the program are in the format of “DD-MM-YYYY”. The second entry is the previous run of the program which shows that there were no records retrieved even though the program was successful and the formats of the dates passed to the program are different.

Execute a prompt program with parameters listed from CCL_RPT_AUDIT_LOG

Another typical use of the CCL_RPT_AUDIT_LOG is to research issues with prompt program. Let's say a user reports that a program is sporadically malfunctioning. They are not quite sure what is entered at the prompts when it works and when it does not work. The CCL_RPT_AUDIT_LOG report shows the failed and successful executions of the program and the parameters passed for each execution. Using the parameters from the report, the troubled program can be executed to reproduce what the user is seeing when the program fails.

First, the CCL_RPT_AUDIT_LOG program is executed for the program that is having a problem. The following report is from CCL_RPT_AUDIT_LOG.

```
Report name(s): 1_CCL_ADT_L                                    Discern Explorer Report Audit
Audit status:   ALL REPORT STATUSES
Start date:     21-OCT-2008 00:00:00.00
End date:       22-OCT-2008 23:59:59.00

Object type:  Status:     Elapsed time:   Date/time:          #Records:  User:          Params:

 Program:  1_CCL_ADT_LOG2:GROUP1
QUERY        FAILED      < 1 Second      10/22/2008 12:43:11      0    Plumber, Joe    "MINE", "*"
QUERY        SUCCESS     < 1 Second      10/22/2008 12:41:19    100    Plumber, Joe    "MINE", VALUE(15359739.00, 309310.00)
QUERY        SUCCESS     49.00 Second    10/22/2008 12:38:27    100    Plumber, Joe    "MINE", 30319663.00
```

Looking at this report, we can see that the second column indicates the program was executed three times with two successful executions and one failed attempt. Under the Params column, you can see that when the program was successful, a number or multiple numbers were passed for the second parameter. When the program was not successful, the "*" (which is a character value) was passed for the second parameter. We can use the information in this report to re-create the issue and view the error message. The parameters that are shown in the report when the program failed should be copied and placed in the paste buffer. These parameters will be used in the next step to troubleshoot the program.

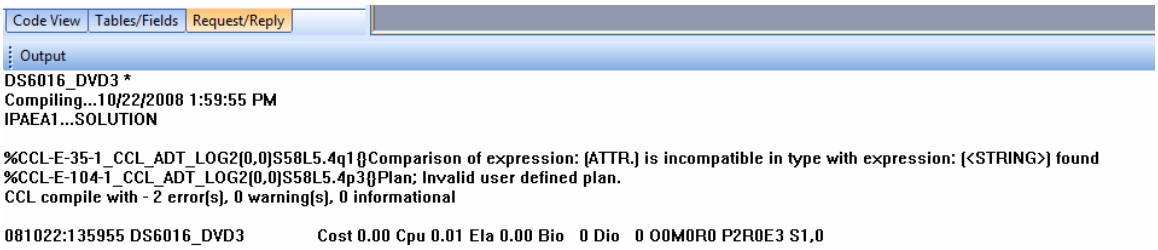
The program with the problem now needs to be executed with the same parameters that caused the program to fail.

The object name is placed in a new blank file and then the parameters captured in the paste buffer should be pasted to the right of the object name. The GO command is added to the end of the statement.

The following shows an example of typing the object name then pasting the parameters into a new blank file. The parameters used to pass to the program were selected from the Params column of the CCL_RPT_AUDIT_LOG report where the program showed it had a failed attempt.

Note: If the object you are executing is a Group1, you must add **:group1** to the object name. To determine if the object is a group1 or 0, execute cclprot for the object and look at the second column of the report.

Clicking the Include/Compile button from the Build menu executes the object. The error message is received and can be seen in the Output window of your workspace. For this example, the following error message is received:



The error message, *CCL-E-35 Comparison of expression is incompatible in type with expression found,* indicates that the program is comparing incompatible data types. Since this program only fails when an asterisk (*) is passed to it, the next step is to open the source code and look to see where the parameter is referenced in the program.

The following is the program that is having the problem:

```
drop program 1_CCL_ADT_LOG2 go
create program 1_CCL_ADT_LOG2

prompt
  "Output to File/Printer/MINE" = "MINE"
  , "Enter an Encounter Type:" = 0.000000

with OUTDEV, Etype

select into $OUTDEV
   e.encntr_id
  , e_encntr_status_disp = uar_get_code_display(e.encntr_status_cd)
  , e_encntr_type_disp = uar_get_code_display(e.encntr_type_cd)

from encounter  e

where e.encntr_type_cd = $Etype
```

```
WITH NOCOUNTER, SEPARATOR=" ", FORMAT, maxrec = 100

end
go
```

The second prompt form is the Encounter type that the user enters at run-time and is defined as following:

General Tab:

- Prompt Display: Enter an encounter type:
- Prompt Name: ETYPE
- Control Type: Code Set
- Prompt Type: Expression

Code Set Tab:

- Code Set: 71
- Display Type: List Box
- Check the box for:
- Multiple Values
- Include Any(*)

The qualification in the code is:

```
Where  e.encntr_type_cd = $Etype
```

The E.ENCNTR_TYPE_CDis an F8 data type. When an asterisk(“*”) is passed from the prompt to the program, the qualification is trying to compare an F8 data type to a character data type. Failure to compare like data types causes a program to fail with the CC-E-35 error message.

The CCL_RPT_AUDIT_LOG shows that when one or multiple code_values are selected, the program runs successfully. Code_values are F8 data types and can be compared to the e.encntr_type_cd which is also an F8 data type.

When the user selects Include Any (*) from the prompt, the asterisk is passed to the program which causes the program to fail. When this option is selected the user wants to query all encounter types. The following is the qualification needed to accommodate this scenario:

```
WHERE E.encntr_type_cd > 0.0
```

However, when the user selects a specific encounter type or multiple encounter types, we want to pass the specific values to the qualification. For example:

```
WHERE e.encntr_type_cd in (43688589.0, 989698.0)
```

The program needs to be able to flex the qualification based on what the user enters at run-time. We can accomplish this with the use of the SELECT IF command. The following program is changed to use SELECT IF to evaluate the data type of what was entered at the prompt. If the input parameter is a character value then do the qualification that is in the “IF” statement. If the input parameter is not a character value, then the logic needs to do the other qualification after the FROM clause.

```
drop program 1_CCL_ADT_LOG2 go
create program 1_CCL_ADT_LOG2

prompt
  "Output to File/Printer/MINE" = "MINE"    ;* Enter or select the printer or file name to
send this report to.
  , "Enter an Encounter Type:" = 0.000000

with OUTDEV, Etype

select

IF ("C*" = reflect(parameter(2,0)))
  WHERE E.encntr_type_cd > 0.0
Endif

into $OUTDEV
  , e.encntr_id
  , e_encntr_status_disp = uar_get_code_display(e.encntr_status_cd)
  , e_encntr_type_disp = uar_get_code_display(e.encntr_type_cd)

from   encounter   e

where e.encntr_type_cd = $Etype

WITH NOCOUNTER, SEPARATOR=" ", FORMAT, maxrec = 100

end
go
```

With the change to the program, the user will be able to successfully run the program when they choose the Include Any(*) option, either a single or multiple encounter types.

When to use “MINE”,“NL:” or “NOFORMS” in the SELECT INTO command

Accessing information about the execution of a program is an integral part of troubleshooting issues with programs. Using DVDev (DiscernVisualDeveloper.exe), the information can be accessed in the Listing from the View menu after the program executes. When a program is executed from a file in DVDev, the Listing may contain error messages, the contents of a Call Echo( ) or Call Echorecord ( ) command, or the output of the query or report.

To get the information to the Listing file, you must define where you want the information to be displayed by using “MINE”, :NL:”, or the “NOFORMS” option in the SELECT INTO <OPTION> command in your query.

```
SELECT into “NL:”
SELECT into “MINE”
SELECT into “NOFORMS”
```

The option you select to use in the SELECT INTO command will depend on the type of information you need to view.

Using SELECT into “MINE”, the Listing will contain:

- the contents from Call Echo( ) and Call Echorecord( ) commands

Using SELECT into “NL:”, the Listing will contain:

- the contents from Call Echo( ) and Call Echorecord( ) commands

Using SELECT into “NOFORMS”, the Listing will contain:

- the contents from Call Echo( ) and Call Echorecord( ) commands
- the output of a query
- the output of the printed information from the report writer section

The contents of Call Echo( ) and Call Echorecord( ) commands can only be accessed in the Listing. If you need to see this information and the program is a prompt program, the information cannot be viewed in the output displayer. Instead the prompt program must be executed by placing the object in a blank file, passing in the needed parameters and then executing the object.

For example, the following program is a prompt program that uses a Call Echorecord( ) command to validate that the record structure has been loaded. It also uses a FOR( ) loop construct in the Foot Report section to write out the contents of the record structure.

```
drop program 1_ccl_person_alias_orders go
create program 1_ccl_person_alias_orders
```

```
prompt
  "Output to File/Printer/MINE" = "MINE"
with OUTDEV

declare pcnt    = i4 with NoConstant(0), protect
declare acnt    = i4 with NoConstant(0), protect

record aliases (
  1 plist [*]
   2 pid = f8
   2 alist[*]
      3 alias = c20
      3 type  = c40
)

select into $OUTDEV
     p.person_id
   , p.alias
   , p_person_alias_type_disp = uar_get_code_display(p.person_alias_type_cd)

From   person_alias   p

where p.active_ind = 1  and exists (  select o.person_id   from orders  o
        where o.orig_order_dt_tm between
        cnvtdatetime(curdate - 7, 0)  and cnvtdatetime(curdate,curtime3)
        and o.person_id+0 = p.person_id)

order by
   p.person_id
   , p_person_alias_type_disp

head report
   pcnt = 0
head p.person_id
   pcnt = pcnt +1
   call echo(pcnt)
   if(mod(pcnt,10) = 1)
     stat = alterlist(aliases->plist, pcnt +1)
     endif
   aliases->plist[pcnt].pid = p.person_id
   acnt = 0
detail
   acnt = acnt +1
   if(mod(acnt,10) = 1)
     stat = alterlist(aliases->plist[pcnt].alist,10)
   endif
   aliases->plist[pcnt].alist[acnt].alias = p.alias
   aliases->plist[pcnt].alist[acnt].type =  p_person_alias_type_disp
```

```
foot p.person_id
  stat = alterlist(aliases->plist[pcnt].alist,acnt)
foot report
  stat = alterlist(aliases->plist, pcnt)
    for(x = 1 to pcnt)
        col 1 "PID: " aliases->plist[x].pid
        acnt = size(aliases->plist[x].alist,5)
        for (y = 1 to acnt)
            col 25  aliases->plist[x].alist[y].alias
            col 45  aliases->plist[x].alist[y].type
           row + 1
        endfor
        row + 1
    endfor

with nocounter, separator=" ", format, maxrec = 100

call echorecord(aliases)

END GO
```

To view the contents of the Call Echo(pcnt) and the Call Echorecord(aliases) commands for this prompt program, place the object name in a blank file in DVDev. Next, any parameters needed for the program must be listed to the right of the object name. This program prompts the user to define where the output should go. (We are attempting to get the information written to the Listing file.)

As the Call Echos by default write to the Listing, any of the options, "MINE", "NL:", or "NOFORMS" can be passed to the first parameter. The GO command must be placed after the command for the compiler to know when to execute. The following example shows how to enter the commands to execute the object passing each of the 3 options: "MINE", "NL:", or "NOFORMS". Only one of these commands would be used in the blank file to execute the program.

```
1_CCL_PERSON_ALIAS_ORDERS "MINE" GO
or
1_CCL_PERSON_ALIAS_ORDERS "NL:" GO
or
1_CCL_PERSON_ALIAS_ORDERS "NOFORMS" GO
```

This program is executed by selecting Include/Compile from the Build menu. The Listing file is available to view after the program executes by selecting Listing from the View menu. The following is a display of the Listing when the object was executed passing in "MINE" as the parameter.

```
1_CCL_PERSON_ALIAS_ORDERS "MINE" GO
```

```
1)1_ccl_person_alias_orders:group1  "MINE" go
1
2
3
4
5
6
7
8
9
10
11
12
13
14

  0)PERSON_ALIAS:P(N#0#0)(Rng:0)Len<450>Dim<1>KEY1<0>Rec<28>

Rec:     28,     29 Row:     28 Qual:     41 Len:    296
>>>Begin EchoRecord ALIASES   ;ALIASES
 1 PLIST[1,14*]
  2 PID=F8   (589863.0000000000                        )
  2 ALIST[1,2*]
   3 ALIAS=C20   (10000130            )
   3 TYPE=C40   (MRN                                    )
  2 ALIST[2,2*]
   3 ALIAS=C20   (999999999           )
   3 TYPE=C40   (SSN                                    )
 1 PLIST[2,14*]
  2 PID=F8   (823932.0000000000                        )
  2 ALIST[1,2*]
   3 ALIAS=C20   (10000141            )
```

The FOR ( ) looping construct in this program contains commands to write data to the output. The output from a reportwriter section or a query can only be viewed in the Listing by using the “NOFORMS” option.

1_CCL_PERSON_ALIAS_ORDERS “NOFORMS” GO

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
 PID:      589863.00     10000130            MRN
                         999999999           SSN

 PID:      823932.00     10000141            MRN
                         111223333           SSN

 PID:      881932.00     10000053            MRN
                         10000212            MRN
                         443332345           SSN

 PID:      945932.00     10000062            MRN
                         234234234           SSN

 PID:     1015935.00     10000078            MRN
                         238939339           SSN

 PID:     1015950.00     10000218            MRN
                         193548130           SSN
```

Note: If the program does not contain prompts, then define the option within the SELECT INTO command within the program and use the <object_name> GO in the blank file to execute the program.

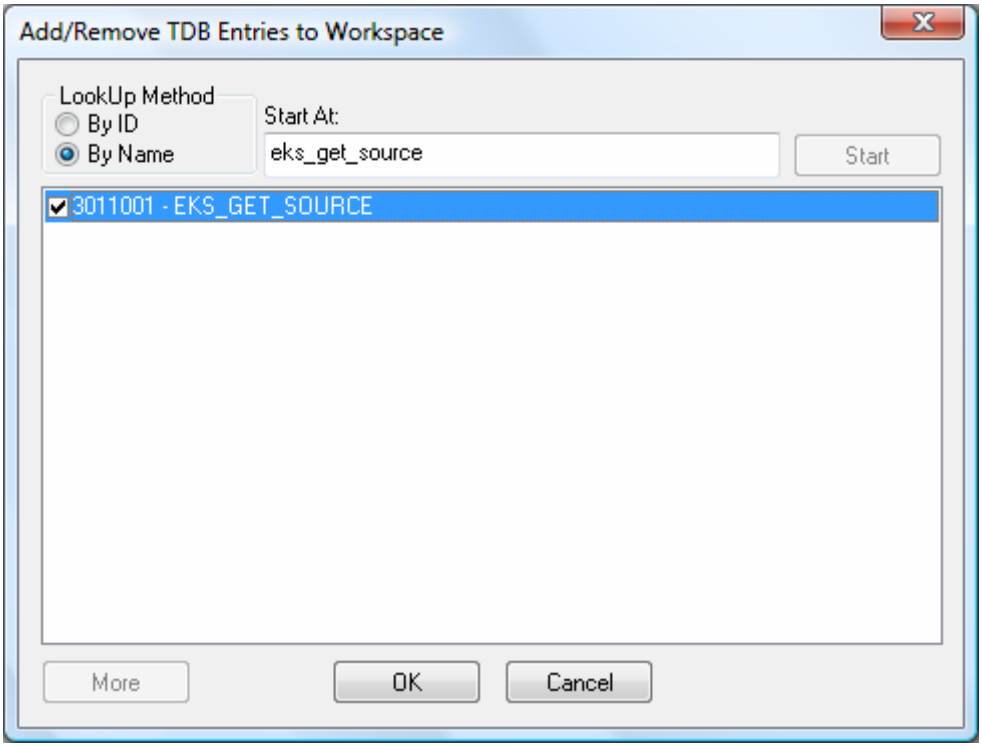
Creating a Test File

There are times when you need to troubleshoot a program that is normally executed from a front-end application. However, as these programs run through application servers, error messages can be difficult to retrieve. Information may go to server log files, which then must be identified. The situation can be further complicated if multiple instances are running. Information from commands such as Call Echo() or Call Echorecord( ) may not be retrievable, especially in a production domain, where the echos are completely turned off to eliminate extra information being written to the server log files.

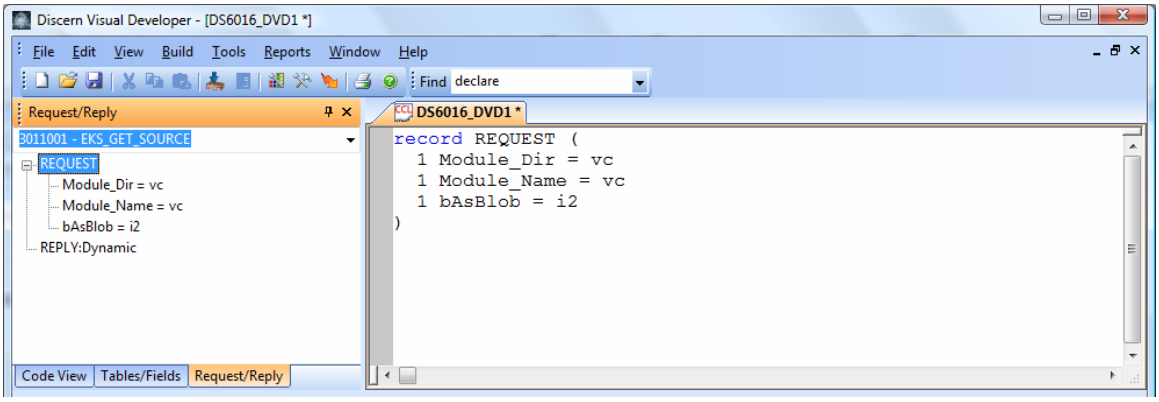
Using a test file to execute your program will eliminate the need to run the program through the front-end application and instead execute the program directly from DVDev. Executing the program directly in DVDev provides easy access to error messages or information from Call Echo( ) and Call Echorecord( ) through the Listing file.

To use this method, you need to know the number or name of the TDB entry that defines the Request and Reply structure. Starting with a blank file in DVDev, the Request and Reply structure is defined, the necessary information needed for the program is set, and then the program is executed. Information from the execution can be viewed in the Listing file.

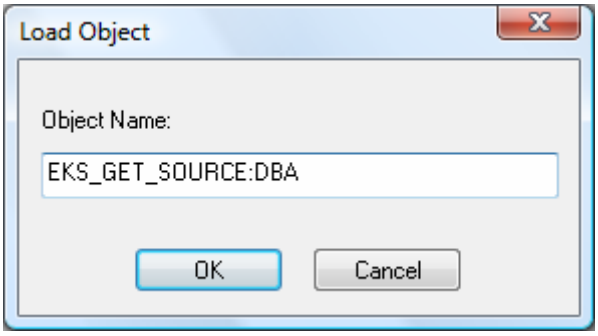
The first step in creating a test file is to load a specific TDB entry to the Request/Reply Tab by selecting Add/Remove Request/Reply from the View menu. Then you can select a TDB entry by name or ID.



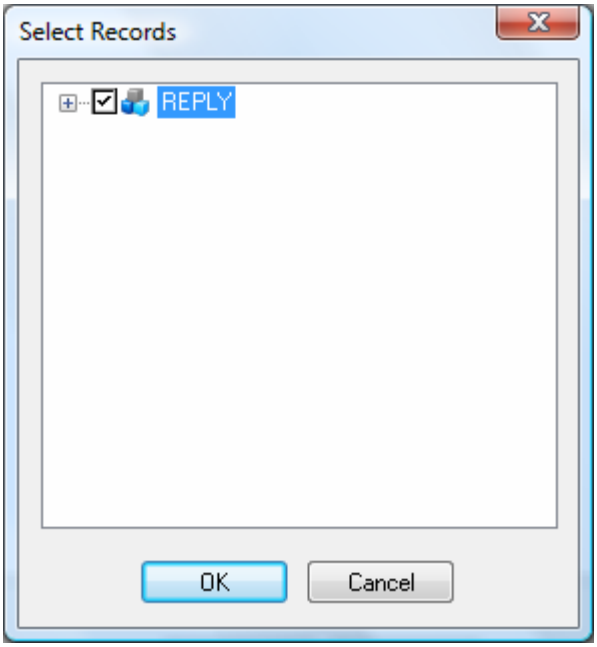
The REQUEST structure can now be dragged to a new blank file.



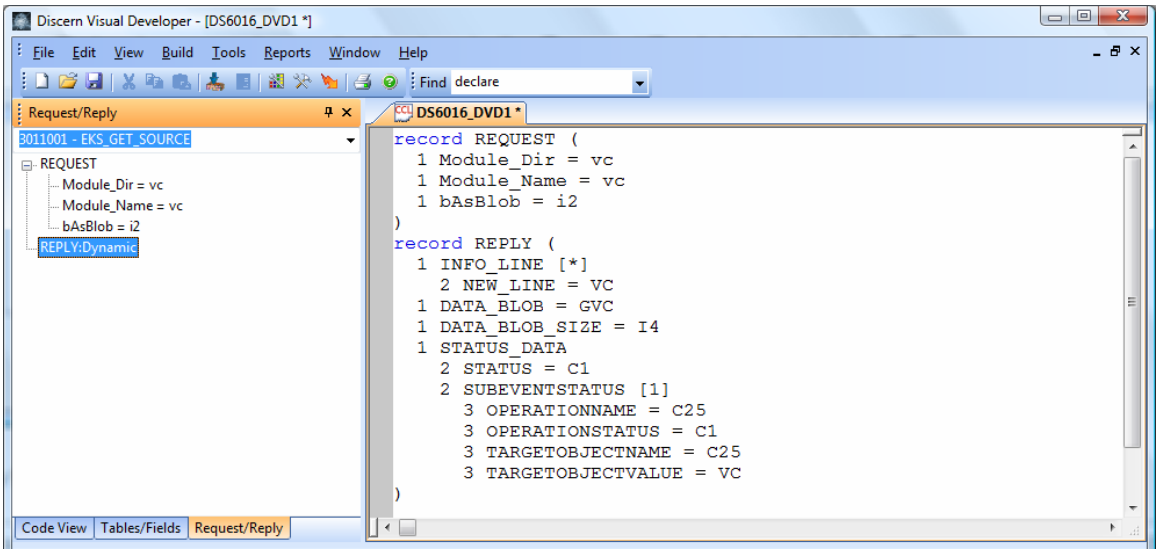
In the above example, the REPLY cannot be dragged to the source file because it is defined dynamically, meaning that this structure is defined in the object. To get to the REPLY structure definition, select the Record Builder from the Tools menu, select Load Object and load the object name that has the defined structure.



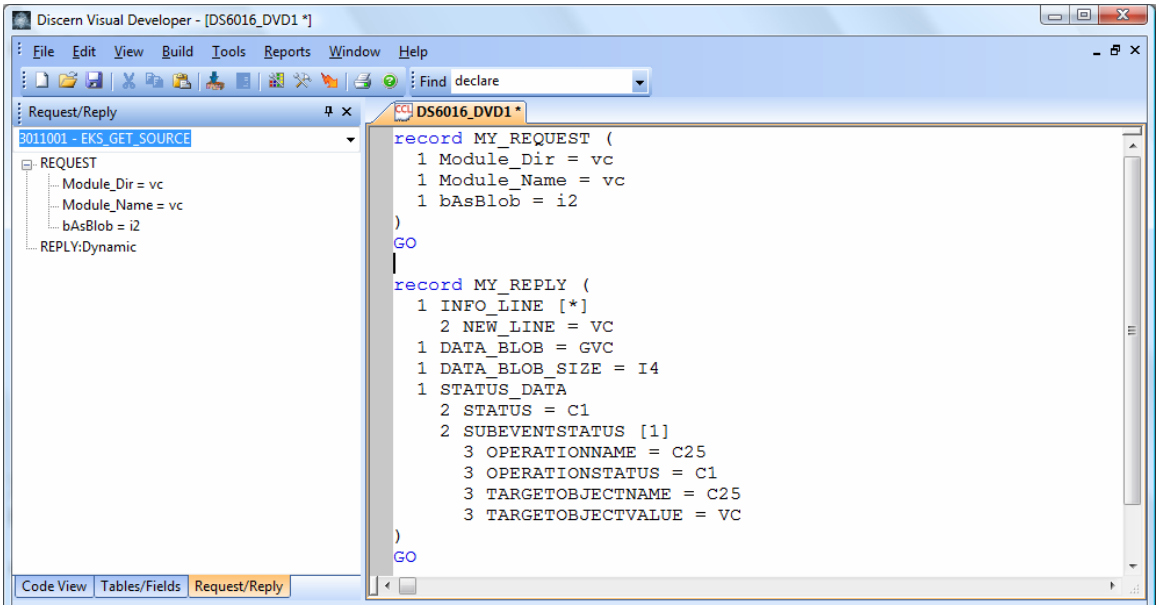
Loading the object allows the REPLY structure to be available.



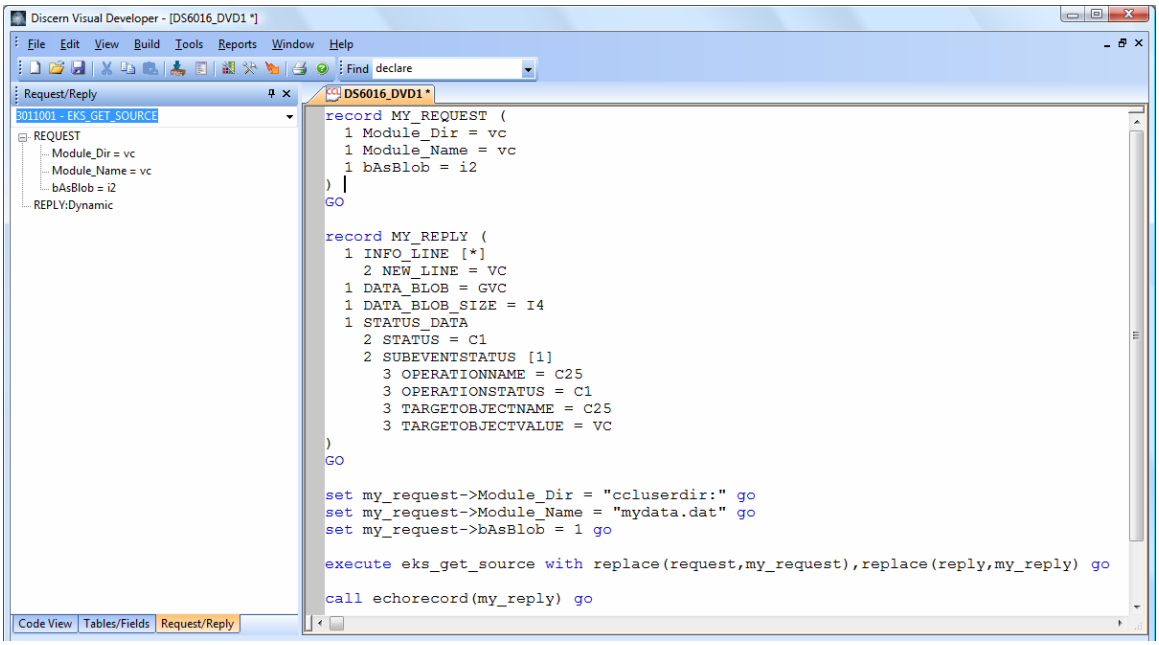
When OK is selected, the REPLY structure is placed in the source file.



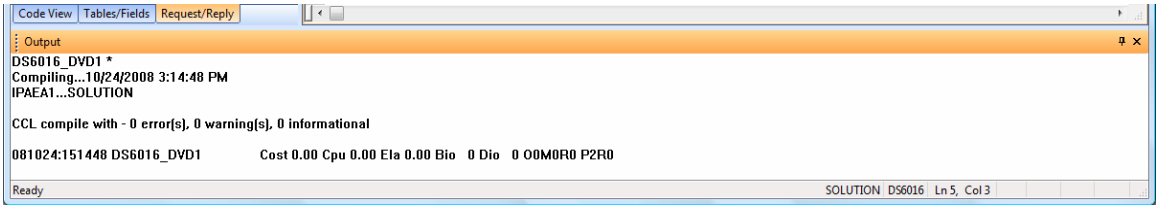
Once the REQUEST and REPLY structures are loaded into the source file, Cerner recommends changing the name of the request and reply to prevent confusion with the request/reply structure defined for DVDev (DiscernVisualDeveloper.exe) that is used for including and compiling. Also, the GO command must be manually placed after each of the definitions. In the following example, REQUEST is replaced with MY_REQUEST. REPY is replaced with MY_REPLY. The GO command is entered after each of the definitions.



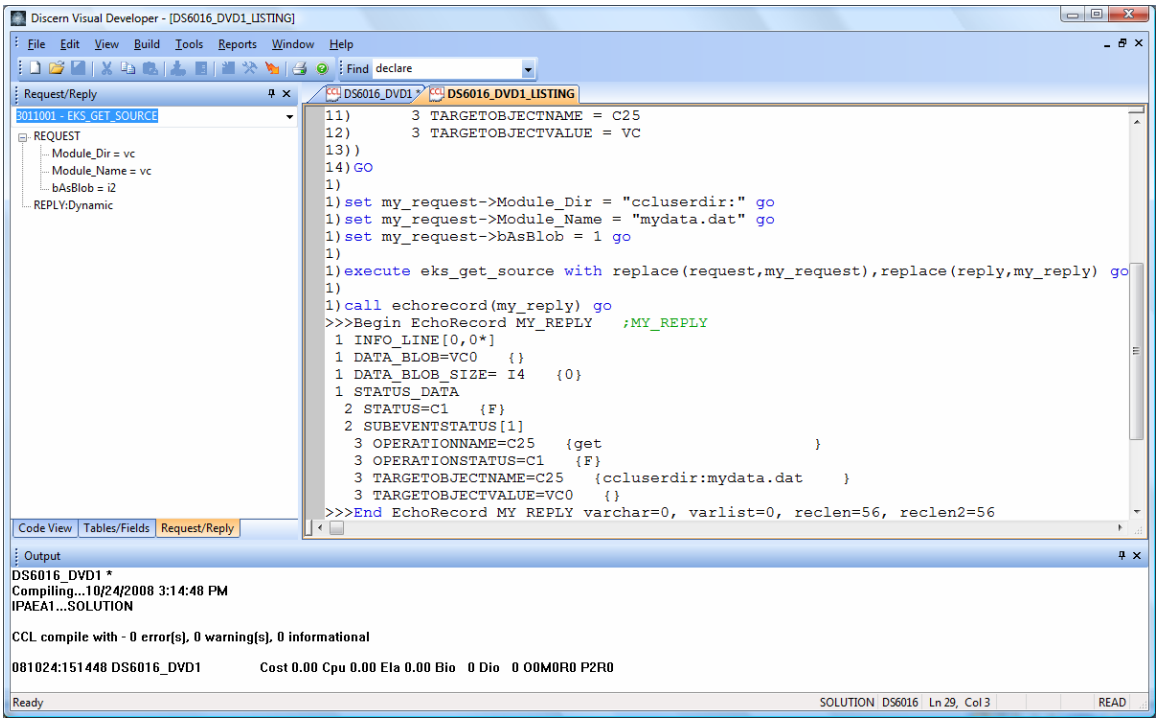
The next step is to populate the items in the request structure that need to be passed to the program using the SET command. Then the program can be executed using the EXECUTE command. In the EXECUTE statement, use the REPLACE() command to search and replace the name of the REQUEST and REPLY structure in the program to match the structures defined in the source file (MY_REQUEST/MY_REPLY). After the execute command, you may want to use the Call Echorecord() command to see the contents of the REPLY record structure. The GO command must be placed after every command.



To execute the commands in the file, select the Include/Compile option from the Build menu. All messages, including error messages, are displayed in the Output dialog box.



The call echo() and call echorecord() commands can be accessed by selecting Listing from the View menu (Ctrl+L).



Document Revision History

Revision Number:	Revision Date:	Description:
001	December 12, 2008	Initial release of this *Cerner Millennium* Support Guide. This guide covers *Discern* Visual Developer functionality in the 2007.18 code release of *Cerner Millennium* and therefore reflects the functionality changes that pertain to that release.