

Visual FoxPro Code Converter - Servoy

FmPro Migrator Developer Edition includes the MacOS X and Windows versions of the Visual FoxPro Code Converter as a compiled stand-alone application. This application converts FoxPro .PRG files and embedded form object code into Servoy JavaScript code.

Visual FoxPro Code Converter 1.17

1/12/2011

Introduction

The Visual FoxPro Code Converter included with FmPro Migrator Developer Edition reads the imported code from FoxPro .PRG files and converts the code into Servoy JavaScript code. Code can be converted individually, as shown using the Development tab, or can be converted for an entire Visual FoxPro project all at once.

Production Conversion Tab

Visual FoxPro Code Converter

File Edit Help

Visual FoxPro Code Converter 1.17

www.FmProMigrator.com

This utility converts Visual FoxPro source code for use with other development environments.

Production Development

Output Directory: 1

Conversion Type: 2

2

Statistics: 3

6 Scripts Processed in 92 ms.

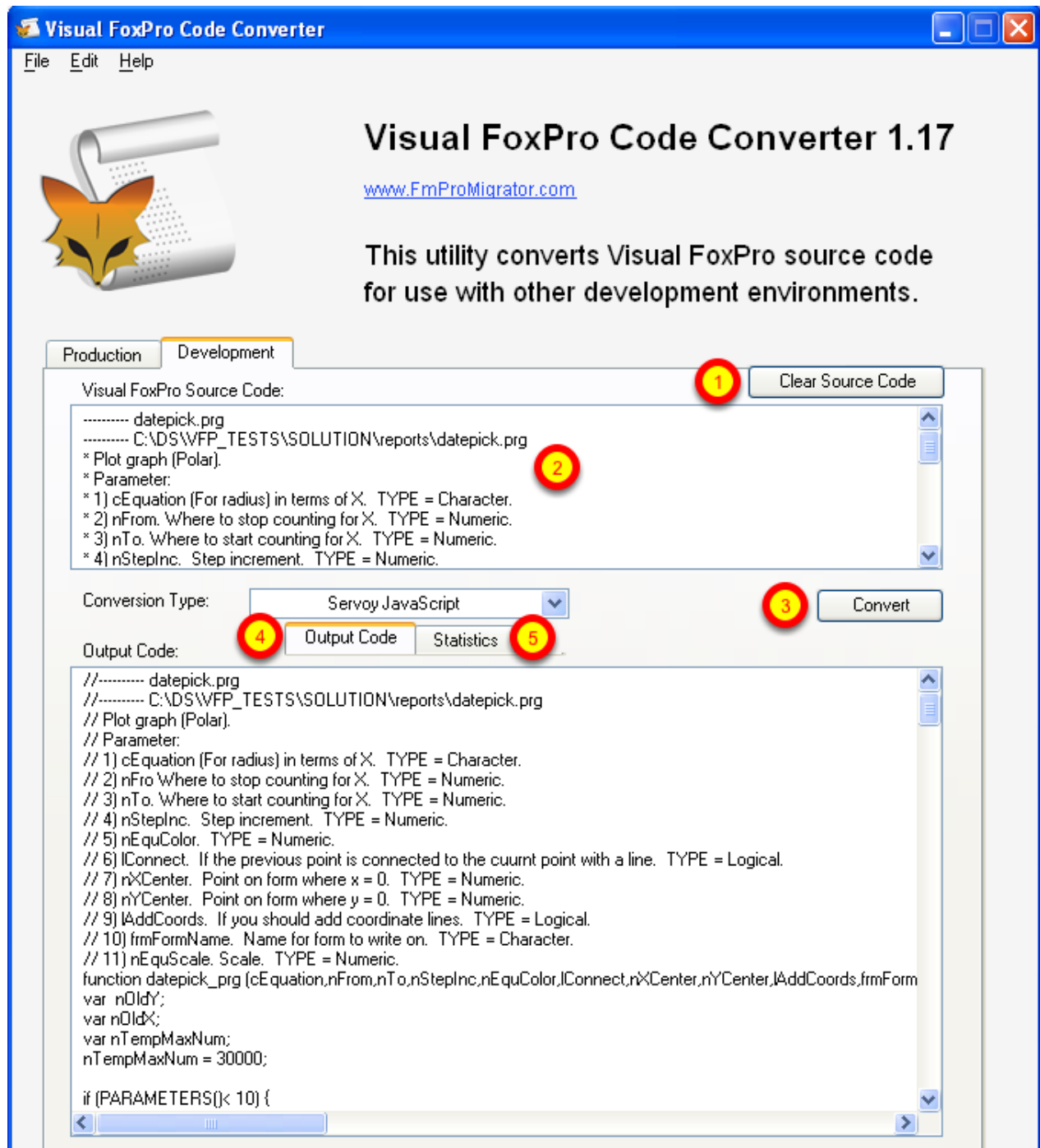
Total Lines Processed: 1426
% Lines Processed: 94%
Blank Lines Processed: 287
% Blank Lines Processed: 20%
Comment Lines Processed: 245
% Comment Lines Processed: 17%
Unsupported Lines Processed: 141
% Unsupported Lines Commented: 9%
UnProcessed Lines: 75
% UnProcessed Lines: 5%

----- UnProcessed Keywords List -----
10 THISFORMSET.frmFD.DrawColor (
9 frmFormName.line(nXCenter,0,nXCenter,frmFormName.height)
9 frmFormName.line(0,nYCenter,frmFormName.width,nYCenter)
5 frmFormName.line(nOldX+nXCenter,nYCenter-nOldY,tempplotx+nXCenter,nYCenter-tempploty)
5 frmFormName.pset(tempplotx+nXCenter,nYCenter-tempploty)
4 frmFormName.pset(nXCenter+(nXTemp*nEquScale),nYCenter-y)
4 frmFormName.line(nXCenter+(nOldX*nEquScale),nYCenter-nOldY,nXCenter+(nXTemp*nEquScale),nYCenter-nY)
2 thisforline(this.oldx,this.oldy,this.currentx,this.currenty)
2 THISFORMSET.frmFD.SetCaption
2 THISFORMSET.frmFD.ClearForm
2 THISFORMSET.frmFD.ChgDrawMode
2 THISFORMSET.frmFD.SetPenWidth (
2 THISFORMSET.frmFD.GraphDemo (
2 Main.Show
2 THISFORMSET.frmFD.SetDrawMode (

The Production tab will read and process all of the script records written into the FmPro Migrator MigrationProcess.db3 project database file during the Visual FoxPro conversion process. To perform a production conversion, (1) select the FmPro Migrator project folder containing the

MigrationProcess.db3 file, (2) click the Convert button. (3) The conversion statistics will be displayed within the Statistics field below the Convert button. The generated Servoy JavaScript files will be written into a folder named VFP Converted Scripts within the output folder.

Development Conversion Tab

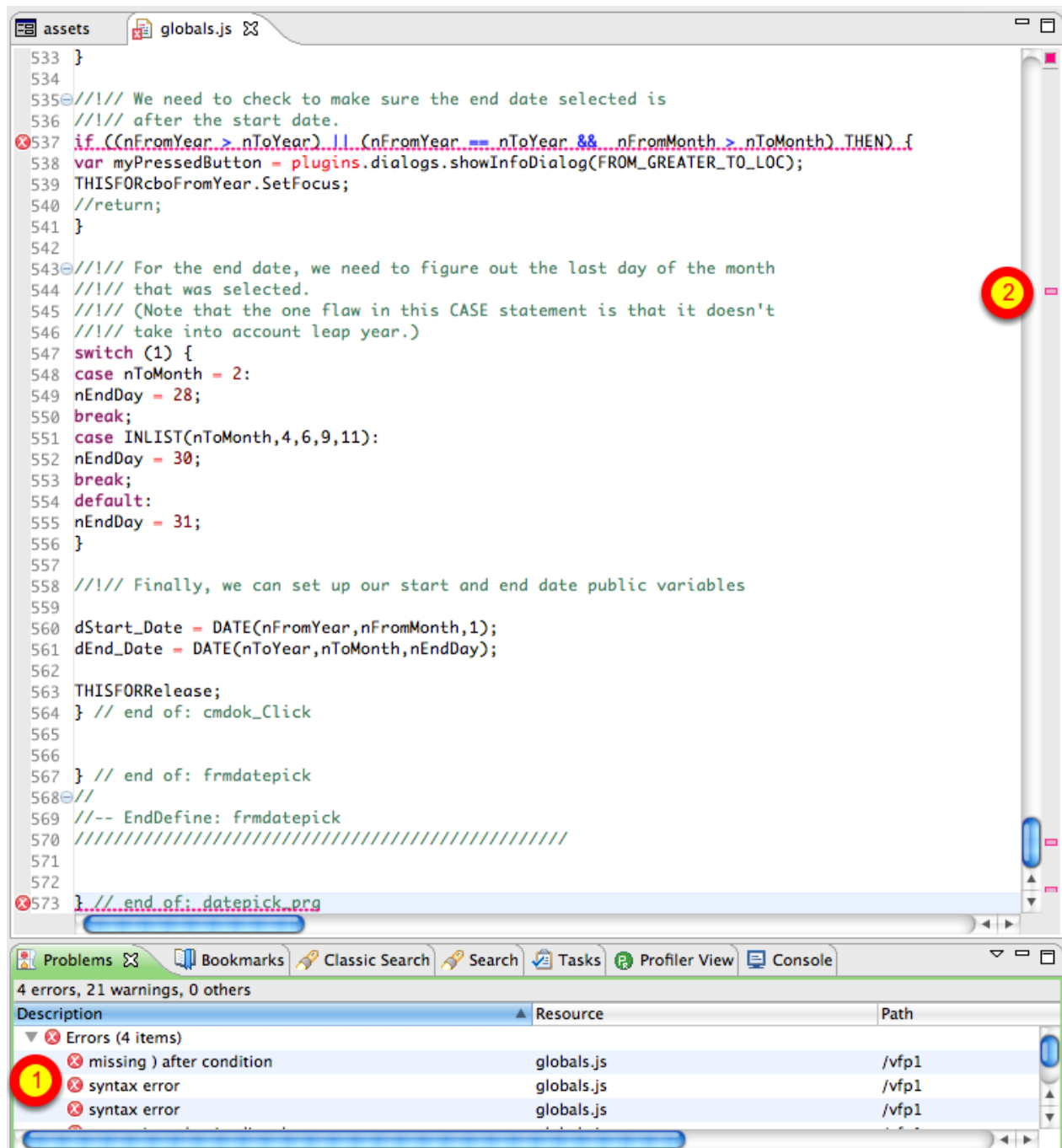


The development conversion tab is designed to provide a place where a single FoxPro script can be quickly pasted and converted.

(1) Click the Clear Source Code button to clear the contents of the Visual FoxPro Source Code field.

Copy source code from any of your Visual FoxPro .PRG files, and (2) paste the code into the Source Code field. (3) click the Convert button. The code will be converted and the contents of the Output Code field will be written into the Output Code field and will also be put onto the clipboard - ready for pasting into the Servoy Eclipse IDE. (5) Clicking the Statistics tab will display the conversion statistics for the converted script.

Pasting Converted Code into the Servoy Eclipse IDE

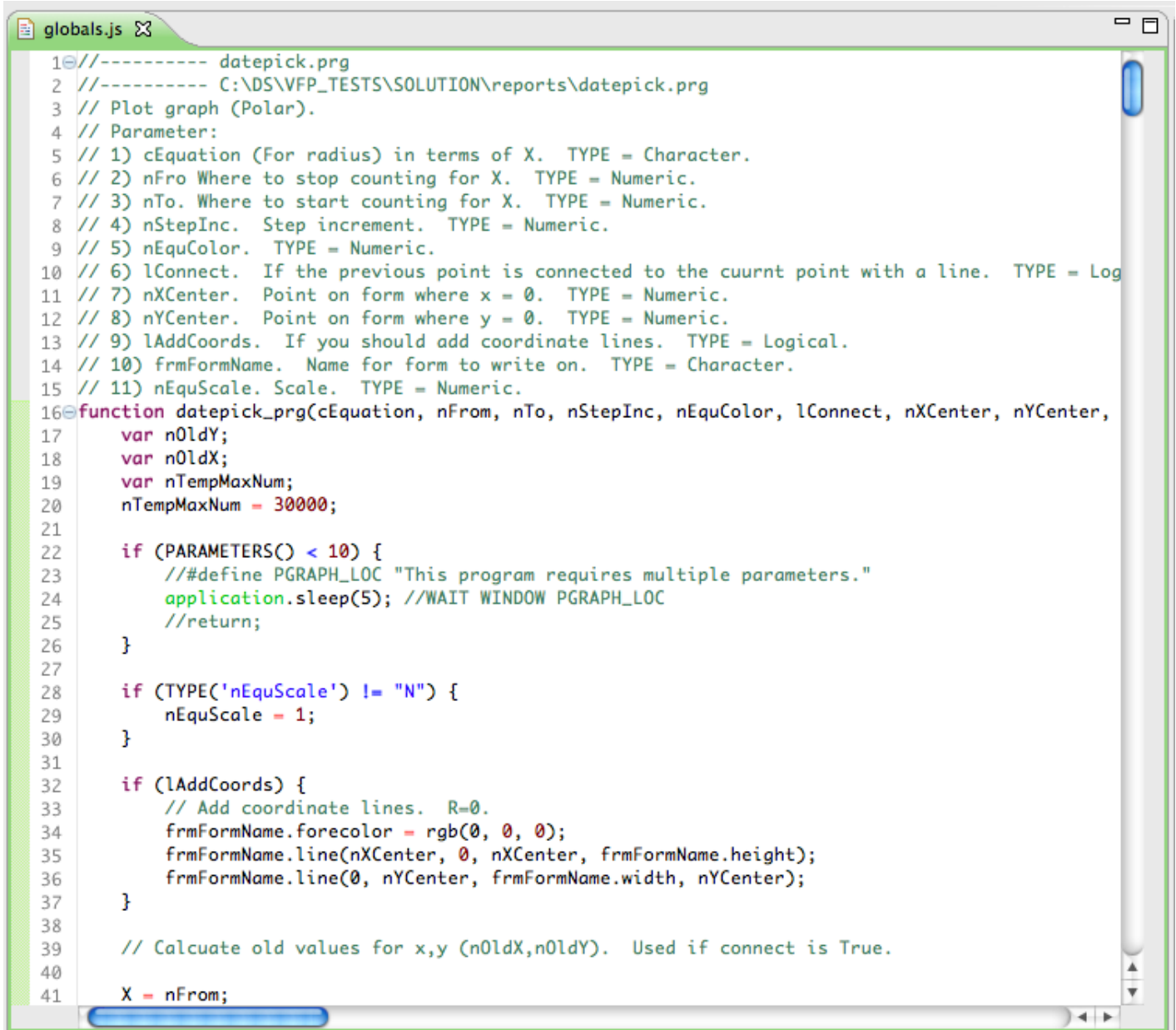


Converted files can be copied and pasted directly into the Servoy Eclipse IDE, or the JavaScript files can be moved into the project folder for editing. However, this is not necessarily the best way to work with converted code files due to the structure of the original Visual FoxPro code. Generally, it is

recommended that the code be pasted and modified in small blocks.

This screenshot shows a 573 line script named datepick.PRG which was converted and pasted into a new globals.js file within the Servoy project. (1) Note that the converted script is showing 4 errors within the Problems tab below the script editor window. (2) Also, the red error bars are displayed along the right hand edge of the editor window. These 4 errors represent a 0.7% error rate for the converted lines of code. Typically the converted script error rate will be 0% - 3%, and these errors will at least need to be commented out in order for the Servoy Eclipse IDE to format and parse the code.

Commenting Errors & Formatting Converted Code



```
1 //----- datepick.prg
2 //----- C:\DS\VFP_TESTS\SOLUTION\reports\datepick.prg
3 // Plot graph (Polar).
4 // Parameter:
5 // 1) cEquation (For radius) in terms of X. TYPE = Character.
6 // 2) nFrom Where to stop counting for X. TYPE = Numeric.
7 // 3) nTo. Where to start counting for X. TYPE = Numeric.
8 // 4) nStepInc. Step increment. TYPE = Numeric.
9 // 5) nEquColor. TYPE = Numeric.
10 // 6) lConnect. If the previous point is connected to the current point with a line. TYPE = Logical.
11 // 7) nXCenter. Point on form where x = 0. TYPE = Numeric.
12 // 8) nYCenter. Point on form where y = 0. TYPE = Numeric.
13 // 9) lAddCoords. If you should add coordinate lines. TYPE = Logical.
14 // 10) frmFormName. Name for form to write on. TYPE = Character.
15 // 11) nEquScale. Scale. TYPE = Numeric.
16 function datepick_prg(cEquation, nFrom, nTo, nStepInc, nEquColor, lConnect, nXCenter, nYCenter,
17     var nOldY;
18     var nOldX;
19     var nTempMaxNum;
20     nTempMaxNum = 30000;
21
22     if (PARAMETERS() < 10) {
23         //#define PGRAPH_LOC "This program requires multiple parameters."
24         application.sleep(5); //WAIT WINDOW PGRAPH_LOC
25         //return;
26     }
27
28     if (TYPE('nEquScale') != "N") {
29         nEquScale = 1;
30     }
31
32     if (lAddCoords) {
33         // Add coordinate lines. R=0.
34         frmFormName.forecolor = rgb(0, 0, 0);
35         frmFormName.line(nXCenter, 0, nXCenter, frmFormName.height);
36         frmFormName.line(0, nYCenter, frmFormName.width, nYCenter);
37     }
38
39     // Calculate old values for x,y (nOldX,nOldY). Used if connect is True.
40
41     X = nFrom;
```

Making a few changes to the code results in resolving the 4 errors. Selecting Source -> Format formats the code and applies proper indenting to make the code easier to read. The code can now be tested with the debugger or edited further.

In general, comments should be located within the method they reference, instead of being above the function definition as is commonly done within FoxPro code - or the comments may be removed during the code formatting process. The converted code has also been changed a little bit from the original in an attempt to wrap the contents of .PRG code within a function, by adding a function definition in place of the first PARAMETER statement near the top of the code. A closing curly bracket was also added at the end of the code as well.

Understanding the Conversion Statistics

The code conversion statistics are designed to provide a general idea of the percentages of processed, unprocessed and unsupported lines of code which were converted. Here is an explanation for these statistics:

Total Lines Processed: The total number of lines which were processed.

% Lines Processed: - The percentage of lines for which some type of processing was done. Each line will either be modified or passed thru unchanged. The following statistics lines provide more details of what was done during the processing.

Blank Lines # or %: - Blank lines will be counted as processed, because nothing needs to be done to them even during a manual conversion since white spaces aids code readability.

Comment Lines # or %: - Comment lines are modified by changing the FoxPro comment character with the JavaScript comment character.

Unsupported Lines # or %: Unsupported Lines are counted in the processed lines percentage, because the lines have been analyzed and determined to be unusable within the context of a multi-user server based environment. A detailed listing of the Unsupported Keywords is provided at the end of the report. These lines are commented to prevent errors.

UnProcessed Lines # or %: The UnProcessed lines are not counted within the total processed percentage, because they did not match an implemented keyword. Each line is passed thru with a semicolon added to the end of the line. Lines containing the equality operator are not considered UnProcessed, and they are also passed thru with a semicolon added to the end of each line.

UnProcessed Keywords List - During script processing, a log is kept of the keywords which were not processed because they did not match a known FoxPro keyword. The number of times each keyword was found is logged along with the first two words of the instruction line. This list provides a quick way to determine if there are FoxPro keywords which need to be added to the conversion process.

Unsupported Keywords List - Each Unsupported Keyword is logged during processing along with a number indicating how many times it was found during processing. This list can provide a quick way to review whether there are FoxPro keywords which could potentially be added to the list of supported keywords in the future. For instance if a particular keyword is observed many times, and a logical replacement can be for the keyword, then the process can be improved considerably.

preParse Processing

Prior to the line-by line processing of each line of source code, a preParse processing step is performed on each script.

The tasks performed at this stage of processing include:

Removal of Line Continuation Characters - The Visual FoxPro line continuation character is a semicolon, which conflicts with the JavaScript line ending character - therefore it must be removed for compatibility purposes. Also, concatenating each of the individual lines together into one single new line makes it easier to parse the original line of code. The total number of lines counted for statistical purposes represents the number of lines which exist in the source script after removing the line continuation characters.

Common FoxPro Function Replacement - Some commonly used FoxPro functions, operators and constants have direct replacements within Servoy JavaScript code. The following replacements are made across the entire script:

Operator and Constant Replacement

AND => &&

OR = ||

NOT = !

<> => !=

m. => empty

.T. => true

.F. => false

tab => empty

Function Replacement

UPPER()=> toUpperCase()

LOWER() => toLowerCase()

LEN() => str.length()

VAL() => parseFloat()

INT() => parseInt()

LTRIM() => stringTrim()

RTRIM() => stringTrim()

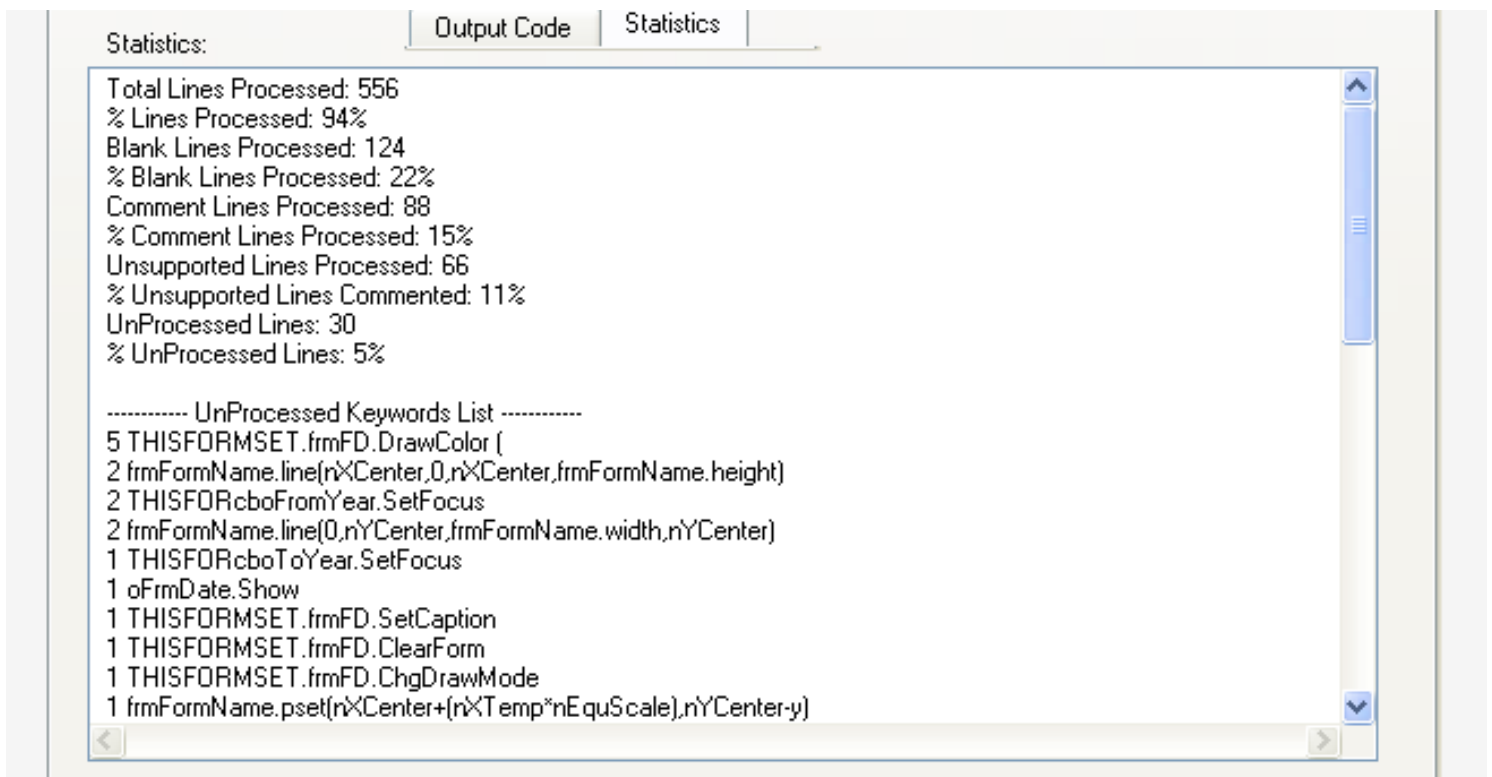
ALLTRIM() => stringTrim()

SUBSTR() => str.substr()

CHR() => String.fromCharCode()

ASC() => charCodeAt()

Manual Code Changes



```
Statistics:
Output Code
Statistics

Total Lines Processed: 556
% Lines Processed: 94%
Blank Lines Processed: 124
% Blank Lines Processed: 22%
Comment Lines Processed: 88
% Comment Lines Processed: 15%
Unsupported Lines Processed: 66
% Unsupported Lines Commented: 11%
UnProcessed Lines: 30
% UnProcessed Lines: 5%

----- UnProcessed Keywords List -----
5 THISFORMSET.frmFD.DrawColor (
2 frmFormName.line(nXCenTer,0,nXCenTer,frmFormName.height)
2 THISFORMcboFromYear.SetFocus
2 frmFormName.line(0,nYCenter,frmFormName.width,nYCenter)
1 THISFORMcboToYear.SetFocus
1 oFrmDate.Show
1 THISFORMSET.frmFD.SetCaption
1 THISFORMSET.frmFD.ClearForm
1 THISFORMSET.frmFD.ChgDrawMode
1 frmFormName.pset(nXCenTer+(nXTemp*nEquScale),nYCenter-y)
```

The automated conversion processing done by the Visual FoxPro Code Converter will attempt to make as many changes as is practical to convert FoxPro code into compatible Servoy JavaScript code. There are however a variety of changes which can only be performed manually.

Code restructuring - A FoxPro PRG file can contain bare keywords outside of a Function/Procedure because the whole file is basically treated like a procedure. Servoy developers generally write all code encapsulated within a method (or function). Also, most Servoy methods don't define additional methods, so these will generally need to be restructured to become separate methods.

Windows Client Based Coding - Commands which attempt to write directly to a DBF file on a user's local disk or open/close database files are unsupported, and unneeded within a Servoy application. All database connections are available immediately just by specifying the dataProvider. And all tables are stored centrally instead of on the user's local hard disk.

Visual FoxPro Specific Coding - The SET, SYS() and FORMSET commands are very specific to functionality which only exists within FoxPro - therefore these commands are commented out and marked as unsupported. FoxPro specific functions also come into this category, though some commonly used functions are converted during the preParsing part of the conversion process.

Form Object Creation Coding - Servoy forms are built by the developer (or converted by FmPro Migrator directly from the original FoxPro forms/reports/labels). Generally, Servoy objects are not created on forms programmatically by the developer, so the FoxPro ADD OBJECT commands are

commented out as unsupported.

Command Line Coding - FoxPro projects may contain commands which prompt a user to enter an answer in a command window and wait for the user to enter an the answer. Processing continues once the user has entered the answer followed by a return key. Servoy apps are event driven apps designed with a graphical interface instead of being command line driven apps. In a Servoy app, the user enters info into fields on a form and then clicks in the form background to commit the record or clicks a button to initiate the processing. Therefore the processing of the entered info could be done via a calculated field or by the button click event.

KeyBoard Event Driven Coding - FoxPro code may contain keystroke driven event code. A Servoy application cannot be dependent upon trapping a particular keystroke combination in order to trigger the execution of code because the application may be deployed via a web browser.

List of Supported Keywords

There are 60 keywords supported during the code conversion process. These keywords are:

- *
- &&
- +
- ACTI WIND
- ACTIVATE WINDOW
- ALLTRIM()
- AND
- CASE
- CASE RETURN
- CHR()
- CHR()
- COUNT
- CREATE TABLE
- DIMENSION
- DISPLAY DATABASE
- DISPLAY TABLES
- DO
- DO CASE
- DO FORM
- DO WHILE
- ELSE
- ENDCASE
- ENDDEFINE

ENDDO
ENDFOR
ENDFUNC
ENDIF
ENDIF
ENDPROC
.F.
FOR
FOUND()
FUNCTION
IF
INSERT INTO
INT()
LEN()
LOCAL
LOOP
LOWER()
LTRIM()
MESSAGEBOX
MODIFY DATABASE
NOT
OR
OTHERWISE
PRIVATE
PROCEDURE
PUBLIC
QUIT
RETURN
RTRIM()
SHOW WINDOW
SORT
STORE
SUBSTR()
.T.
UPPER()
VAL()
WAIT

List of UnSupported Keywords

There are 55 keywords which are unsupported during the code conversion process. These keywords are:

\$

?

ADD OBJECT

APPEND

BOF()

BROWSE

CLEAR ALL

CLEAR EVENTS

CLOSE ALL

CLOSE DATA

CLOSE DATABASE

CLOSE TABLES

CONTINUE

CREATE

CREATE DATABASE

CTOD()

DEACTIVATE WINDOW

#DEFINE

DEFINE WINDOW

DELETE

DELETED()

DISPLAY

DISPLAY

DTOC()

EOF()

GO

GOTO

HIDE WINDOW

#INCLUDE

INDEX

LIST

LOCATE

MODIFY STRUCTURE

ON ERROR

ON KEY

OPEN DATABASE

PACK
POP KEY
PUSH KEY
READ
RECALL
RECNO()
RELEASE
RELEASE ALL
RELEASE CLASSLIB
RELEASE THISFORMSET
RELEASE WINDOW
REPLACE
SELECT
SET
SKIP
SUM
THISFORMSET
USE
ZAP