A Macro Program for Generating Customized Reports
from User-Specified Column Variables and Page-Sort Variables
Jim DeFoer, General Dynamics - Fort Worth

OUTLINE

Introduction
Actions Performed in Program
Macro Techniques Used
Explanation of Macro Code
Three Reports Produced by Program

INTRODUCTION

The Macro techniques shown in this poster allow one program to generate many different reports, each with its own unique numeric columns and logical page sorts. That capability permits users to report or analyze many different 'cuts' of their data with one program. Having only one program perform a multitude of reporting tasks reduces the development and maintenance activities of the supporting programmer. The techniques are used in a program at General Dynamics - Fort Worth (GDFW) that generates cost estimate reports of F-16 aircraft. The program has been used for every major proposal at GDFW during the last five years.

Three reports generated by this program are shown in this poster. Each was created by the unique SAS code generated by one Macro program. That Macro code generated the SAS code according to the program-fixed row variable of the report and the user's selection of the report's column and page-sort variables. It created individual column variables from each data value of the column variable selected by the user. It then used those column variables and the page sorts selected by the user to create the specific SORT, SUM, TRANSPOSE, and PUT operations necessary to organize and report the data in the structure desired by the user.

The Macro code and SAS code sections are organized in parallel subsections. The reader will find the SAS code under the same subheading as the Macro code which generated it. This should aid the reader in understanding the form and purpose of the Macro code in that subsection by allowing line-by-line comparisons of the created code versus the generating code.

Only the significant Macro code in the program is shown in this poster. That code is shown in the order of its use in the actual program. The complete program reads an external data set, assigns the data to cost groups, sorts and sums the data, transposes the data to the structure needed for the report, and then reports the data. The comments embedded in the Macro code explain the purpose and function of the following or adjoining lines of Macro code. The comments in the SAS code explain only how the Macro code was resolved.

The section that explains the Macro techniques used in the program is meant as a brief reference of those techniques. Complete explanations may be found in the SAS Guide to Macro Processing, Version 5. Page citations are provided.

The last section provides three examples of the reports generated by the actual program. Those reports describe the estimated hours of effort for a proposed task at General Dynamics - Fort Worth. Note that all reports use the same variable, CEC (cost group), as the left-hand stub or row variable of the customized report, but that each has different numeric column variables and no sort variables.

ACTIONS PERFORMED IN PROGRAM

Each action listed here is fully described in the section describing the Macro code and the Macro-generated SAS code.
1. Collect the variables that will generate the columns and the logical page breaks of the report.
2. Sort and sum the data according to the column variable, the page-sort variables, and the program's fixed row variable.
3. Build Macro variables from the data values of the selected column variable.
4. Create a Data Step for transposing the data (one that is more efficient than PROC TRANSPOSE). Generate a SAS variable for each of the Macro variables that represent the new column variables.
5. Establish a report-writing Data _Null_ Step that
   A. Generates a new logical print page for each change in the data values of that last sort variable specified.
   B. Builds a TOTAL variable of the new SAS column variables.
   C. Writes the column data onto the report using PUT statements.
   D. Prints the values of the page-sort variables below the titles and above the column headings of the report using PUT statements in the HEADER area of the Data _Null_ step.
   E. Writes the column headings onto the report using PUT statements in the HEADER area of the Data _Null_ step.

MACRO TECHNIQUES USED IN PROGRAM

Several techniques are used in more than one section of the code. Accordingly, the numbers below do not correspond to the numbers of the subsections listed above. The techniques are, however, listed in their order of occurrence in the section describing the Macro code.
1. Define a Macro variable with a %LET statement.
2. Simple Macro variable substitution.
3. Define a Macro variable with the CALL SYMPUT function.
4. Iterative %DO loops.
5. %IF-%THEN conditional execution.
6. %SUBSTR function.
7. %LENGTH function.
8. %EVAL function.
9. Double resolution of Macro variables inside of %DO loops.
10. %DO %WHILE loops.
MACRO CODE USED IN PROGRAM
(See Section On Actions Performed In Program)

1. The user’s selection of the column variable of his report is stored in the Macro variable &COLUMN using a %LET statement. A blank entry will cause the program to fail. In this example, the SAS variable YEAR is loaded into &COLUMN.

   %LET COLUMN = YEAR;
   &COLUMN will resolve to YEAR.

   The user’s selection of the logical sort variables of his report are stored in the Macro variables &SORT1 - &SORT5. Blank entries are acceptable. They will resolve to blank during code generation. In this example, organization is loaded into the &SORT1 variable. Blanks are loaded into &SORT2 through &SORT5.

   %LET SORT1 = ORG;
   &SORT1 will resolve to ORG (blank) (blank) (blank).

   %LET SORT2 = ;
   &SORT2 will resolve to blank (blank) (blank) (blank).

   %LET SORT3 = ;
   &SORT3 will resolve to blank (blank) (blank) (blank).

   %LET SORT4 = ;
   &SORT4 will resolve to blank (blank) (blank) (blank).

   %LET SORT5 = ;
   &SORT5 will resolve to blank (blank) (blank) (blank).

2. The report’s data is sorted and summed by the page-sorting variables, the fixed row variable of the report (CEC) and the column variable selected by the user. That summed data is placed in the data set SUMDATA. The order of that data corresponds to the order needed in the report to be generated by the program. SPECDATA is also created in the same Data Step. It contains only the data values of the selected column variable and will be used to create the MACRO variables that will generate the new SAS column variables of the report.

   /* SORT DATA IN PREPARATION OF SUMMING */
   PROC SORT DATA = COSTDATA;
   BY ORG &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 / PAGE SORTS CEC &COLUMN;
   &COLUMN will resolve to ORG (blank) (blank) (blank) (blank) (blank).
   RUN;

   /* SUM DATA TO UNIQUE VALUES */
   DATA SUMDATA(DROP=AMT) / DATA TO BE REPORTED /
   SPECDATA(KEEP=AMT) / DATA TO GENERATE COL VARS /
   SET COSTDATA; / SORTED INPUT DATA /
   BY ORG &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 / PAGE SORTS CEC &COLUMN;
   &COLUMN will resolve to ORG (blank) (blank) (blank) (blank) (blank).
   AMT + AMOUNT;
   IF LAST.&COLUMN THEN /* LAST OCCURRENCE OF EACH DATA VALUE */
   DO;
   AMOUNT = AMT; /* MOVE SUMMED VALUE INTO AMOUNT */
   AMT = 0;
   OUTPUT; /* OUTPUT SUMMED VALUES */
   END;
   OUTPUT SUMMED VALUES;
   END;
   RUN;

3. SPECDATA, the data set containing only the data values of the selected column variable, is sorted by those data values. Then, in the subsequent Data Step, Macro variables of each of the data values of the column variable are created using CALL SYMPUT. Each Macro variable begins with the string ‘VAR’ and ends in an integer representing the sort order of the column attribute to be loaded into that variable. For example, if the column variable is YEAR and the data values of YEAR are 1990, 1991, 1992, 1993, 1994, and 1995, Macro variables VAR1 through VAR5 will be loaded with 1990 through 1995. An underscore (_) will be added to the beginning of each string, since SAS variable names cannot begin with integers. Also, using CALL SYMPUT, the total number of columns that will be printed will be loaded at the end-of-file to the Macro variable &TOTAL_COLS. The value of &TOTAL_COLS will be used to control the interactive %DO loops that will generate the SAS code that sums all column variables into the TOTAL variable and that writes the column data and headings during the DATA _NULL_ print step;

   /* SORT DATA VALUES OF COLUMN VARIABLE */
   PROC SORT DATA = SPECDATA;
   BY &COLUMN;
   RUN;

   /* CREATE MACRO VARIABLES FOR COL VARIABLES OF REPORT */
   DATA _NULL_;
   SET SPECDATA END=EOF; /* DATA TO GENERATE COL VARIABLES */
   BY &COLUMN;
   IF FIRST.&COLUMN THEN /* FIRST OCCURRENCE OF EACH VALUE */
   DO;
   /* CREATE MACRO VARIABLES FOR COL VARIABLES OF REPORT */
   DATA _NULL_;
The next Data Step transposes the data, creating numeric SAS column variables from the data values of the selected column variable. An iterative %DO loop generates IF-THEN-ELSE statements that assign the data values of the column variable to new variables having the same name as the data value. Beginning underscores (_) are excluded from the evaluation since they were added to the beginning of the Macro variable in order that the forthcoming SAS variable would have a legal SAS name.

```sas
%DO J = 1 %TO &IDT_COLS;
%IF &J NE 1 %THEN
%DO;
* IF-THEN-ELSE STATEMENTS TO CREATE SAS VARIABLES FROM THE UNIQUE DATA VALUES OF THE SELECTED COLUMN VARIABLE HAVE BEEN GENERATED.*
* IF YEAR = "1990" THEN_1990 = AMOUNT;*
* ELSE IF YEAR = "1991" THEN_1991 = AMOUNT;*
* ELSE IF YEAR = "1992" THEN_1992 = AMOUNT;*
* ELSE IF YEAR = "1993" THEN_1993 = AMOUNT;*
* ELSE IF YEAR = "1994" THEN_1994 = AMOUNT;*
* ELSE IF YEAR = "1995" THEN_1995 = AMOUNT;*
%END;
ELSE
%END;
/* IF THEN-ELSE STATEMENTS TO CREATE SAS VARIABLES FROM THE UNIQUE DATA VALUES OF THE SELECTED COLUMN VARIABLE HAVE BEEN GENERATED. */
*/&COLUMN RESOLVED TO YEAR */
IF E0F THEN
call symput('TOT_COLS', 'VAR' & IDT_COLS);
run;
```

1267
MACRO CODE

IF LASTCEC THEN DO;
  OUTPUT:
  /* GENERATE SAS CODE TO SET VARIABLES TO MISSING */
  %DO J = 1 %TO &TOT_COL;
  &&VAR&J = .;
  %END;
END;
RUN;

RESOLVED SAS CODE

IF LASTCEC THEN DO;
  OUTPUT:
  /* SAS CODE HAS BEEN GENERATED */
  /* TO SET THE NEW COLUMN VARIABLES TO */
  /* MISSING () AFTER OUTPUT */
  _1990 = .;
  _1991 = .;
  _1992 = .;
  _1993 = .;
  _1994 = .;
  _1995 = .;
END;
RUN;

5A. In the Data _Null_ Step that writes the report, a new logical page is printed each time there is a change in the data value of the last sort variable specified. The Macro code uses a %DO %WHILE loop to determine the last sort variable specified by the user.

DATA _NULL_; /* REPORT WRITING STEP */
  SET RPTDATA; /* TRANPOSE REPORT DATA */
  FILE PRINT;
  BY &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 CEC;
  %LET LASTSORT = ; /* SET LASTSORT TO BLANK */
  %LET J = 1; /* INITIALIZE J TO ONE (1) */
  DO WHILE SORT VARIABLES ARE NOT BLANK /* DO WHILE SORT VARIABLES ARE NOT BLANK */
    %00 %WHILE (&&SORT&J NE ); /* LOAD SORT VARIABLE INTO LASTSORT */
    %LET LASTSORT = &&SORT&J;
    %LET J = %EVAL(&J + 1); /* ADD ONE (1) TO J */
  %END;
  IF &LASTSORT IS NOT EQUAL TO BLANK, THEN ISSUE /* IF LASTSORT IS NOT EQUAL TO BLANK, THEN ISSUE */
    %IF &LASTSORT NE 'II THEN
      IF FIRST.&LASTSORT THEN /* IF FIRST.&LASTSORT THEN */
        PUT _PAGE_; /* PUT _PAGE_; */
    %END;
END;
RUN;

5B. In the Data _Null_ Step, a TOTAL variable is created by a SUM of the column variables generated by an iterative %DO loop. That loop is controlled by the &TOT_COL value that represents the number of column variables which were created in the TRANSPOSE step. The variables are generated using double resolution.

  /* GENERATE SAS VARIABLES TO BE SUMMED INTO TOTAL */
  %DO J = 1 %TO &TOT_COL;
  &&VAR&J
  %END;
);  

5C. In the Data _Null_ Step, the left-hand descriptive variable and the generated SAS column variables are written using an iterative %DO loop controlled by &TOT_COL. The pointer position for the PUT statements is controlled by &POSITION, which is initialized to 35 and advanced by 12 (the length of the numeric fields + 1) for each pass through the loop. The %EVAL function is used to add 12 to &POSITION.

%LET POSITION = 35;
PUT @ 3 CEC $CEC30. /* LEFT-HAND DESCRIPTIVE VARIABLE */
%DO J = 1 %TO &TOT_COL;
  /* PRINT LOCATION AND FORMAT OF EACH COLUMN */
  @&POSITION &&VAR&J COMMAI1.
  /* CALCULATE NEXT PRINT POSITION */
  %LET POSITION = %EVAL(&POSITION + 12);
%END;
/* PRINT LOCATION AND FORMAT OF TOTAL COLUMN */
@&POSITION TOTAL COMMAI3.;

1268
In the HEADER area of the Data _Null_ Step, the character string 'TOTAL' is printed just below the titles of the report if &LASTVAR is blank, which occurs when the user does not specify one or more sort variables. Otherwise, the values of the page-sort variables are printed just below the report's titles.

%IF &LASTVAR = '%THEN
  %DO;
  PUT @18 'TOTAL';
  %END;
%ELSE:
  %DO;
    PUT @18 &SORT1 @50 &SORT2 / 
    @18 &SORT3 @50 &SORT4 / 
    @18 &SORT5 ;
  %END;

In the HEADER area of the Data _Null_ Step, the descriptive headings of each numeric column are generated using an iterative %DO loop controlled by &TOT_COLS. &POSITION is again used to control the pointer position for the PUT statement that writes the descriptive headings. Here, however, it is used with the %LENGTH value of the character string that is to be printed. The length of character string is subtracted from 11 (the numeric field width) and that difference is added to the &POSITION value using the %EVAL function. For example, if the string were '1990', the %LENGTH value would be 5 and, thus, 6 (11-5) would be added to the value of &POSITION. Adding 6 to &POSITION places the '1990' string at the rightmost 5-character position of the numeric field.

%LET POSITION = 35;
PUT %DO J = 1 %TO &TOT_COLS:
  /* CALCULATE LENGTH OF HEADING STRING */
  %LET Y = %LENGTH(&VAR&J);
  /* CALCULATE STARTING LOCATION OF HEADING STRING */
  %LET START = %EVAL(&POSITION+(11-&Y));
  /* ISSUE HEADING STRING TO BE WRITTEN TO PRINT FILE */
  @&START &VAR&J
  /* DETERMINE LOCATION OF NEXT HEADING STRING */
  %LET POSITION = %EVAL(&POSITION + 12);
%END;
PUT %LET Y = 5;
/* LOAD LENGTH OF TOTAL STRING TO Y */
/* CALCULATE STARTING LOCATION OF TOTAL STRING */
%LET Z = %EVAL(&POSITION+(13-&Y));
/* ISSUE TOTAL STRING */
@&Z 'TOTAL';
EXPLANATION OF MACRO TECHNIQUES USED IN PROGRAM.

1. Define a Macro variable with a %LET statement.
   
   %LET COLUMNS = YEAR;

   A character string can be directly assigned to a Macro variable using the %LET statement. %LET statements can appear anywhere in a SAS program. See pages 72-73, SAS Guide to Macro Processing, Version 5.

2. Simple Macro variable substitution.

   %COLUMN = YEAR;

   Whenever the Macro processor sees an ampersand (&) joined immediately to a character string, it searches the appropriate symbol table for a Macro variable of that name and substitutes the character string stored there for the Macro variable. Macro Variables can be used anywhere in a SAS program. See pages 24-25, SAS Guide to Macro Processing, Version 5.

3. Define a Macro variable with the CALL SYMPUT function.

   CALL SYMPUT(VAR,YEAR) CALL SYMPUT(,lUT _COLS',TOT _COLS)

   CALL SYMPUT(argument1,argument2) assigns the string value in argument2 to the Macro variable of that name. Argument1 may be a quoted string, such as 'TOT _COLS'; it may be a SAS variable containing a character string, such as VAR, which contains the strings 'VAR1', 'VAR2', etc. Argument2 may be a quoted character string or a SAS variable. CALL SYMPUT can only be used in a Data Step. See pages 124-129, SAS Guide to Macro Processing, Version 5.

4. Iterative %DO loops.

   %DO J = 1 %TO &TOT _COLS;
   &&VAR&J = ;
   %ENO;

   An iterative %DO loop repeats the actions inside the loop until the control variable of the loop equals the upper bound of the loop. It has a built-in counter that increments the value of the control loop after each pass through the loop. In this example where &TOT _COLS = 6, the &&VAR&J will be resolved six times and the strings resulting from &&VAR&J = ; will be emitted six times. See pages 54-55, SAS Guide to Macro Processing, Version 5.

5. %IF-%THEN conditional execution.

   %IF &J NE 1 %THEN
   %DO;
   ELSE
   %END;

   SAS statements located within an %IF-%THEN statement are emitted only when that statement evaluates to true. In this example, the ELSE statement will be released only when &J is not equal to 1. See pages 12 and 66, SAS Guide to Macro Processing, Version 5.

6. %SUBSTR function.

   %SUBSTR(&&VAR&J,2,4)

   The %SUBSTR(argument,starting position,length) function returns a substring of characters. It begins its substring operation at the starting position identified in the function and continues for a length specified in the function. If &&VAR&J resolves to '_1990, the returned string would be 1990. See page 102-103, SAS Guide to Macro Processing, Version 5.

7. %LENGTH function.

   %LENGTH(&&VAR&J)

   The %LENGTH(argument) function returns the number of characters in the resolved value of the argument it is examining. If the resolved value is _1990, the length would be five (5). See page 96, SAS Guide to Macro Processing, Version 5.

8. %EVAL function.

   %EVAL(%LENGTH(&&VAR&J)-1)

   The %EVAL function performs simple integer arithmetic that does not involve integers greater than 2^32-1. In this example, it subtracts one (1) from the integer value returned from the %LENGTH function. Here, that would be 5, so the value returned by the %EVAL function would be 4 (5 - 1). See page 95, SAS Guide to Macro Processing, Version 5.

9. Double resolution of Macro variables inside of %DO loops.

   %DO J = 1 %TO &TOT _COLS;
   &&VAR&J = ;
   %END;

   Double resolution results from the Macro processor's orientation to change double ampersands to a single ampersand (&) during each scanning of the tokens and then to rescanning any strings resulting from a resolution. Thus, a resolved string is scanned at least twice.

   When J = 1, &&VAR&J resolves to &VAR1 after the first pass of tokenization. The && becomes a single &, the VAR remains as is, and the &J becomes a 1, creating &VAR1. Then, during the second pass, &VAR1 resolves to the string stored in the symbol table for that variable, which — in this example — is the SAS variable _1990. No further resolution occurs during the third pass, so the complete string emitted by this %DO loop would be ' _1990 = ;'. %DO loops can occur only within the Macro code of a SAS program. See pages 6-8 and 28-31, SAS Guide to Macro Processing, Version 5.

10. %DO %WHILE loops.

    %LET J = 1;
    %DO %WHILE(&&SORT&J NE &LASTSORT);
    %LET J = %EVAL(&J + 1);
    %END;

   A %DO %WHILE loop repeats the actions within the loop while a condition holds true. The evaluation occurs at the top of the loop, so the actions of the loop are never performed if the condition is untrue initially. Usually, %DO %WHILE loops are used with a counter that is explicitly initialized before the loop is started and explicitly incremented at the bottom of the loop before the control condition is evaluated. Like iterative %DO loops, %DO %WHILE loops can occur only within the Macro code of a SAS program. See page 59, SAS Guide to Macro Processing, Version 5.
### THREE REPORT EXAMPLES

**REPORT GENERATED FROM COLUMN = NRRC (NON-RECURRING/RECURRING).**

<table>
<thead>
<tr>
<th>HOURS</th>
<th>N_RECNRG</th>
<th>RECURRING</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research &amp; Engineering</td>
<td>0</td>
<td>44,239</td>
<td>44,239</td>
</tr>
<tr>
<td>Electronic Systems</td>
<td>0</td>
<td>3,422</td>
<td>3,422</td>
</tr>
<tr>
<td>Electronic Systems ICL</td>
<td>0</td>
<td>3,761</td>
<td>3,761</td>
</tr>
<tr>
<td>F-16 Program Mgmt.</td>
<td>0</td>
<td>9,741</td>
<td>9,741</td>
</tr>
<tr>
<td>F-16 Program Office</td>
<td>0</td>
<td>3,840</td>
<td>3,840</td>
</tr>
<tr>
<td>Logistics</td>
<td>659</td>
<td>29,941</td>
<td>36,500</td>
</tr>
<tr>
<td>Total Engr. Direct Hours</td>
<td>659</td>
<td>29,941</td>
<td>36,500</td>
</tr>
</tbody>
</table>

| Manufacturing   |          |           |       |
| Tool Engineering | 0  | 16,562   | 16,562 |
| Tool Manufacturing Airframe | 0  | 8,000   | 8,000  |
| Tool Manufacturing Mse.  | 0  | 3,014   | 3,014  |
| Plant Engineering  | 215 | 91      | 306   |
| Factory Fabrication | 0  | 39,116  | 39,116 |
| Factory Assembly   | 0  | 86,155  | 86,155 |
| Electronic Bench   | 0  | 33,232  | 33,232 |
| Field Operations   | 0  | 24,370  | 24,370 |
| Mod & Test Aircraft | 0  | 307     | 307   |
| Quality Assurance  | 0  | 725     | 725   |
| Total Mfg. Direct Hours | 215 | 211,572 | 211,572 |

**REPORT GENERATED FROM COLUMN = PRRE (PRODUCTION/RETOFIT).**

<table>
<thead>
<tr>
<th>HOURS</th>
<th></th>
<th>PRDCTON</th>
<th>RETROFIT</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research &amp; Engineering</td>
<td>0</td>
<td>44,139</td>
<td>44,139</td>
<td></td>
</tr>
<tr>
<td>Electronic Systems</td>
<td>0</td>
<td>3,422</td>
<td>3,422</td>
<td></td>
</tr>
<tr>
<td>Electronic Systems ICL</td>
<td>0</td>
<td>3,761</td>
<td>3,761</td>
<td></td>
</tr>
<tr>
<td>F-16 Program Mgmt.</td>
<td>0</td>
<td>9,741</td>
<td>9,741</td>
<td></td>
</tr>
<tr>
<td>F-16 Program Office</td>
<td>0</td>
<td>3,840</td>
<td>3,840</td>
<td></td>
</tr>
<tr>
<td>Logistics</td>
<td>659</td>
<td>30,600</td>
<td>30,600</td>
<td></td>
</tr>
<tr>
<td>Total Engr. Direct Hours</td>
<td>659</td>
<td>30,600</td>
<td>30,600</td>
<td></td>
</tr>
</tbody>
</table>

| Manufacturing   |          |         |          |       |
| Tool Engineering | 0  | 16,562  | 16,562  |
| Tool Manufacturing Airframe | 0  | 8,000   | 8,000   |
| Tool Manufacturing Mse.  | 0  | 3,014   | 3,014   |
| Plant Engineering  | 215 | 91      | 306     |
| Factory Fabrication | 0  | 39,116  | 39,116  |
| Factory Assembly   | 0  | 86,155  | 86,155  |
| Electronic Bench   | 0  | 33,232  | 33,232  |
| Field Operations   | 0  | 24,370  | 24,370  |
| Mod & Test Aircraft | 0  | 307     | 307     |
| Quality Assurance  | 0  | 725     | 725     |
| Total Mfg. Direct Hours | 215 | 211,572 | 211,572 |

**REPORT GENERATED FROM COLUMN = YEAR (CALENDAR YEAR).**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research &amp; Engineering</td>
<td>0</td>
<td>337</td>
<td>4,669</td>
<td>23,819</td>
<td>15,414</td>
<td>44,239</td>
</tr>
<tr>
<td>Electronic Systems</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>379</td>
<td>3,043</td>
<td>3,422</td>
</tr>
<tr>
<td>Electronic Systems ICL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,283</td>
<td>1,878</td>
<td>3,161</td>
</tr>
<tr>
<td>F-16 Program Mgmt.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,048</td>
<td>8,693</td>
<td>9,741</td>
</tr>
<tr>
<td>F-16 Program Office</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>413</td>
<td>3,427</td>
<td>3,840</td>
</tr>
<tr>
<td>Logistics</td>
<td>246</td>
<td>91</td>
<td>5,794</td>
<td>10,017</td>
<td>14,452</td>
<td>30,600</td>
</tr>
<tr>
<td>Total Engr. Direct Hours</td>
<td>246</td>
<td>428</td>
<td>10,463</td>
<td>36,959</td>
<td>46,907</td>
<td>95,003</td>
</tr>
</tbody>
</table>

| Manufacturing   |      |      |      |      |      |       |
| Tool Engineering | 0  | 0    | 212  | 14,319| 2,031| 16,562|
| Tool Manufacturing Airframe | 0  | 0    | 102  | 6,910 | 988  | 8,000 |
| Tool Manufacturing MSE  | 0  | 0    | 354  | 2,126| 532  | 3,014 |
| Plant Engineering  | 0  | 0    | 34   | 204  | 88   | 306   |
| Factory Fabrication | 0  | 0    | 1,855| 37,033| 228  | 39,116|
| Factory Assembly   | 0  | 0    | 1,716| 74,149| 10,297| 86,155|
| Electronic Bench   | 0  | 0    | 229  | 32,609| 394  | 33,232|
| Field Operations   | 0  | 0    | 996  | 11,540| 11,864| 24,370|
| Mod & Test Aircraft | 0  | 0    | 1    | 203  | 103  | 307   |
| Quality Assurance  | 0  | 0    | 78   | 647  | 867  | 1,625 |
| Total Mfg. Direct Hours | 0  | 0    | 5,469| 179,116| 27,152| 211,787|

---

**General Dynamics**

Fort Worth Division — Fort Worth, Texas