```
     // Simple program to illustrate table-driven programming
     // Author: Ted Holt
     //
     // Instructions:
     //   (1) Enter filenames of your choosing into Split commands in
     //       the main calculations.
     //   (2) Compile and call this program.
     //   (3) Look at the report it produces.
     //   (4) Have fun!

     H dftactgrp(*no) actgrp(*new)  option(*srcstmt: *nodebugio)

     Fqsysprt   o    f  132        printer

      * ============================================
      * State table
      * ============================================
     D NbrOfRows       c                   const( 5)
     D NbrOfColumns    c                   const( 8)

     D StateTable      ds
     D   Row                          40a   dim(NbrOfRows) ctdata perrcd(1)

     D ptr             s                 *   inz(%addr(StateTable))

     D State           ds                    dim(NbrOfRows) qualified based(ptr)
     D   Entry                               likeds(Action_t) dim(NbrOfColumns)

     D Action_t        ds                    qualified
     D   Action                        2a
     D   NextState                     2s 0
     D                                 1a

     D Pl              ds           132      qualified
     D  Ndx                            6
     D                                 2
     D  State                          6
     D                                 2
     D  Char                           4
     D                                 2
     D  Column                         6
     D                                 2
     D  Action                         6
     D                                 2
     D  Token                         24
     D                                 2
     D  Drive                          5
     D                                 2
     D  Path                          24
     D                                 2
     D  BaseName                       8
     D                                 2
     D  Extension                      3
```

```
     D                                 2

      /free
       *inlr = *on;
       split ('c:\temp\20141120\mydata.dat');
       split ('\temp\20141120\mydata.dat');
       split ('temp\20141120\mydata.dat');
       split ('\mydata.dat');
       split ('mydata.dat');
       split ('c:mydata.dat');
       split ('mydata');
       split ('c:\temp\20141120\mydata');
       split ('temp\20141120\mydata');
       split ('cx:\temp\20141120\mydata.dat');
       split ('c:');
       split ('.dat');
       split ('c:\.dat');
       split ('c:\tmp\br549.dat');
       return;
      /end-free

       // ================================================================
       // Split:
       //
       // Use table-driven programming to split a file name into drive,
       // path, base name and extension.
       //
       // To keep the table simple, these are (at least some of) the rules.
       //
       // (1) The drive is a single letter and is followed by a colon.
       // (2) Leading and trailing blanks are ignored.
       // (3) Embedded blanks are not permitted.
       // (4) Directory names (in the path) and base names cannot contain periods.
       // (5) Directory, file and extension names must be letters and digits only.

       // Also, to keep the table simple, some things that should be errors
       // (such as a missing base name), are allowed.

     P Split           b

     D                 pi
     D   FileName                    80a   varying const

     D Drive           s                1a
     D Path            s               80a   varying
     D BaseName        s               80a   varying
     D Extension       s               80a   varying
     D Token           s               80a   varying

     D ErrorCode       s                5u 0
     D CurrState       s                3u 0
     D CurrChar        s                1a
     D CharNdx         s                3u 0
```

```
     D StringLength    s              3u 0
     D Column          s              3u 0
     D CurrAction      s              2a
     D EOL             c                   const(x'00')
     D DashLine        s            110a   inz(*all'-')

      /free
        clear Drive;
        clear Path;
        clear BaseName;
        clear Extension;
        clear Token;

        writeln (FileName);
        evalr Pl.Ndx = 'Ndx';
        evalr pl.State = 'State';
        eval  pl.Char  = 'Char';
        evalr pl.column = 'Column';
        evalr pl.Action = 'Action';
              pl.Drive  = 'Drive';
              pl.Path   = 'Path';
              pl.BaseName = 'Base';
              pl.Extension = 'Extension';
              pl.Token = 'Token';
        writeln (pl);

        // Initialize the state machine

        ErrorCode = *zero;
        CharNdx = *zero;
        StringLength = %len(%trimr(FileName));
        CurrState = 1;

        // Begin loop to process the file name, one character at a time.

        dow CurrState <> *zero
        and ErrorCode = *zero;

            // Move to next character.
            CharNdx += 1;

            if CharNdx <= StringLength;
               CurrChar = %subst(FileName: CharNdx: 1);
            else;
               CurrChar = EOL;
            endif;

            // Decide which column of the state table to use.
            select;
               when CurrChar = EOL;
                  Column = 1;
               when CurrChar = *blank;
                  Column = 2;
```

```
            when (      CurrChar >= '0'
                    and CurrChar <= '9');
                Column = 4;
            when CurrChar = ':';
                Column = 5;
            when CurrChar = '\';
                Column = 6;
            when CurrChar = '.';
                Column = 7;
            when (      CurrChar >= 'A'
                    and CurrChar <= 'Z')
                or (    CurrChar >= 'a'
                    and CurrChar <= 'z');
                Column = 3;
            other;
                Column = 8;
        endsl;

        // Retrieve the action from the state table.
        CurrAction = State(CurrState).Entry(Column).Action;

        // Carry out the action.
        select;
          when CurrAction = 'NO';
            // no action
          when CurrAction = 'ER';
            ErrorCode = CharNdx;
          when CurrAction = 'CT';
            Token = (Token + CurrChar);
          when CurrAction = 'TD';
            Drive = Token;
            clear Token;
          when CurrAction = 'TP';
            Token = (Token + CurrChar);
            Path = (Path + Token);
            clear Token;
          when CurrAction = 'TB';
            BaseName = Token;
            clear Token;
          when CurrAction = 'TE';
            Extension = Token;
            clear Token;
          other;
            writeln('Invalid action ' + CurrAction);
        endsl;

        evalr Pl.Ndx = %editc(CharNdx:'4');
        evalr pl.State = %editc(CurrState:'4');
        eval  pl.Char  = currChar;
        evalr pl.column = %editc(Column:'4');
        evalr pl.Action = CurrAction;
        eval  pl.Drive  = Drive;
        eval  pl.Path   = Path;
```

```
            eval  pl.BaseName   = BaseName;
            eval  pl.Extension  = Extension;
                  pl.Token      = Token;
            writeln (pl);

            // Retrieve the next state from the state table.
            CurrState = State(CurrState).Entry(Column).NextState;
          enddo;
          writeln (DashLine);
        /end-free

     P                 e
       // ================================================================
       // Writeln:
       //
       // Write free-form output to the printer.
       // ================================================================
     P Writeln         b
     D                 pi
     D  inString                     132a   varying const

     D Output          ds             132
       Output = inString;
       write qsysprt Output;
     P                 e

       // ================================================================
       // The state table follows below.
       // Each row represents a state.
       // Each column represents an input category.

       // columns:
       //  1 : EOL
       //  2 : blank
       //  3 : letter
       //  4 : digit
       //  5 : colon
       //  6 : backslash
       //  7 : period
       //  8 : other

----1-*--2----3----4----5----6----7----8
eol   *   ltr  dig   :    \    .    other
**
NO00 NO01 CT02 CT04 ER00 TP04 ER00 ER00
ER00 ER00 CT04 CT04 TD03 ER00 ER00 ER00
TD00 ER00 CT04 CT04 ER00 TP04 TB05 ER00
TB00 ER00 CT04 CT04 ER00 TP04 TB05 ER00
TE00 ER00 CT05 CT05 ER00 ER00 ER00 ER00
```