TML Pascal
Reference Manual Supplement

This supplement documents additional TML Pascal language features that were omitted from the TML Pascal Reference Manual and corrects information which was printed in error. This document is organized by the chapters and chapter sections which are affected by the omissions.

Chapter 1 – Tokens

Character-Strings

In order to support the definition of Menus for the Apple IIGS Menu Manager, the scanner for Pascal string literals has been changed to interpret an occrance of the characters '\0' as a single character whose value is equivalent to *Chr(0)*. The *Chr(0)* value is needed in the string literal as a *separator* between menu items.

Example: *'>>File\N300\0==Open...\N301\0==-\N302D\0.'*

Chapter 5 - Expressions

The @ Operator

The operand types allowed for the @ operator has been extended to allow quoted Pascal string literals.

Example: *@'TML Pascal'*

Chapter 10 – Standard Procedures and Functions

Miscellaneous Functions

The MoveLeft Procedure

Syntax: MoveLeft(source, dest, count)

MoveLeft copies a block of *count* contiguous bytes of storage from *source* to *dest* beginning at the lowest memory address of the blocks (the first byte of *source* and *dest*). *Source* and *dest* are variable references of any type. *Count* is an integer expression. When *source* and *dest* overlap, you should use this procedure if *source* is at the higher memory address.

The MoveRight Procedure

Syntax: MoveRight(source, dest, count)

MoveRight copies a block of *count* contiguous bytes of storage from *source* to *dest* beginning at the highest memory address of the blocks (the last byte of *source* and *dest*). *Source* and *dest* are variable references of any type. *Count* is an integer expression. When *source* and *dest* overlap, you should use this procedure if *source* is at the lower memory address.

The FillChar Procedure

Syntax: FillChar(dest, count, ch)

FillChar fills a block of *count* contiguous bytes of storage with the specified value *ch* beginning at the address of *dest*. *Dest* is a variable reference of any type, *count* is an integer expression, and *ch* is a value of an ordinal type.

The ScanEq Function

Syntax: ScanEq(limit, ch, source)
Result: *Integer*

ScanEq scans a block of memory beginning at *source* for the first occurance of the value *ch*. The scan proceeds until the value *ch* is found, or until *limit* bytes of memory have been scanned. If *ch* is not found within *limit* bytes of memory from the beginning of *source*, the value returned is equal to *limit*. Otherwise, the value returned is the number of bytes scanned before the value *ch* was found.

The Scaneq Function

Syntax: ScanNe(limit, ch, source)
Result: *Integer*

ScanNe operates the same as ScanEq except that it scans for the first byte *not equal* to *ch*.

Appendix B - Compiler Directives

ToolError Check

{$ToolErrorChk+} or {$ToolErrorChk-}

Default: {$ToolErrorChk+}

This directive allows an application to control the automatic generation of error checking code for Apple IIGS Toolbox calls. As discussed in Chapter 5 of the User's Guide and Chapter 10 of the Reference Manual, TML Pascal generates a `STA ToolErrorNum` instruction after every call to a Toolbox routine so that the special TML Pascal global variable *ToolErrorNum* always contains the error code of the most recently called Toolbox routine. A non-zero *ToolErrorNum* indicates an error occured during the execution of the last Toolbox routine, and the value of *ToolErrorNum* is an error code that can be used to determine the cause of the error.

In many cases, an application does not need to check the value of *ToolErrorNum* after Toolbox calls, and would rather not have the `STA ToolErrorNum` instruction generated in order to decrease the code size of an application. To achieve this, the *$ToolErrorChk* directive is turned off.

See the section *How Calling a Tool Routine Works* in Chapter 5 of the User's Guide.

Unit Symbol File Search Prefix

{$P *ProDOS16 prefix*}

Default: {$P 0/ }

This option allows an application to specify any legal ProDOS16 prefix for the purposes of searching for unit symbol files (.USYM files). The TML Pascal compiler does not recompile the interface part of units specified in a USES clause, but rather loads a precompiled symbol table of the declarations in a unit from a *.USYM* file. To search for these files, TML Pascal maintains a current *unit prefix* used to create the full pathname of a *.USYM* file. The default prefix is "0/" which is the ProDOS16 prefix for the current directory. This *unit prefix* can be changed to any legal prefix using this compiler directive. For example,

```
USES    QDIntf,
        {$P /TML/MYSTUFF/ } HandyRoutines;
```

Note that if a *.USYM* file cannot be found using the compiler's *unit prefix* the compiler will also attempt to find the file by using the ProDOS16 prefix 7. If the file cannot be found in either of these locations and error is reported.

Appendix C - Inside TML Pascal

The StdColors Array

TML Pascal's ConsoleIO.Pas unit implements the routine

```
PROCEDURE SetDithColor(Color: Integer);
```

for the purpose of setting the QuickDraw pen "color" to a value between 0 and 15 inclusive for the Super HiRes 640 mode. The routine is provided for beginning programmers generally using the compiler's *Plain Vanilla* type of applications so that they can easily use more than the four true colors available in 640 mode. In actuality, the SetDithColor routine changes the QuickDraw pen pattern to take advantage of a technique known as *dithering.* Dithering is the technique of alternating colors very close together in a pattern, to give the appearance of another color. The SetDithColor routine uses 16 different patterns which are defined in the array

```
VAR  StdColors: array[0..15] of Pattern;
```

StdColors[0] contains a pattern defined by 16 words of $0000, StdColors[1] contains a pattern defined by 16 words of $1111, etc. Finally, StdColors[15] contains a pattern defined by 16 words of $FFFF.

Applications not using the ConsoleIO.Pas unit may still access this array of patterns. The array variable StdColors is declared in the TMLPASCALLIB runtime library and may be accessed using TML Pascal's *$XrefVar* directive. Consider the following program fragment.

```
PROGRAM ColorPatsDemo;

USES QDIntf;

{$XrefVar+}
VAR  StdColors: array[0..15] of Pattern;
{$XrefVar-}

BEGIN
   ...
   SetPenPat(StdColors[3]);
   LineTo(10,20);
   FillRect(r,StdColors[5]);
   ...
END.
```