Name: WriteFormatted

Maintainer: Eric Schmidt, Larry Stewart

Date: June 5, 1980 8:34 PM

Purpose: WriteFormatted is a family of procedures to write strings on an output stream. The user may also define his own formats and incorporate them into WF via procedure calls.

WriteFormatted is based on the UNIX *printf* routine for C programs, Ed McCreight's *FORMAT* routines and Ed Taft's *TEMPLATE* routines for BCPL, and the BCPL string conventions.

Note: The DEFINITIONS file has been renamed "WF.bcd" and its implementor is now "WFImpl.bcd" to conform to the Pilot and Cedar naming conventions.

General Format:

```
WFn["string with escapes", arg1, ..., argn];
```

where n (between 0 and 4) is the number of substitutions into the string by values of arg1, ... Substitution positions in the string are of the form %z, where z is an expression composed of:

- An optional minus sign ('-') which specifies left-adjustment instead of the normal right adjustment of the argument.

- An optional zero ('0') which indicates a field that, if necessary, will be filled with leading zeroes rather than with leading blanks.

- An optional number specifying a field width; i.e. the number of digits for a number, or the maximum number of characters to be printed from a string.

- The field width number may be of the form xx.yy for use by a user supplied floating point output routine. (*See RealDefs.)*

- An option letter 'l' for long (32-bit) numbers.

- A character indicating the conversion to be applied to the argument.

The conversion characters are (upper-case is also allowed):

b - Print one word quantity as octal number (unsigned). No trailing 'B' is supplied.

c - The argument is interpreted as a CHARACTER and printed.

d - Treat as INTEGER, print decimal number.

s - The argument is interpreted as a STRING and printed.

u - The argument is interpreted as a CARDINAL and printed (unsigned).

x - Print one word quantity as hexadecimal number (unsigned).

If no recognizable character follows the %, and it is not a user-supplied routine (see below), then it is printed. Thus % may be printed by %%.

If the optional letter 'l' preceds one of 'b', 'd', 'r', 't', 'u', or 'x' WF treats the corresponding parameter as a POINTER TO a 32-bit quantity:

lb - Treat as POINTER to 32-bit quantity, print in octal (unsigned). No leading 0's or trailing 'B' are supplied.

ld - Treat as POINTER TO LONG INTEGER, print 32-bit integer.

lr - Treat as POINTER to 32-bit quantity which is the difference between two TimeDefs.PackedTime's, print in "hr:mn:sc" format.

lt - Treat as POINTER to TimeDefs.PackedTime, print in TimeDefs.AppendTime[] format.

lu - Treat as POINTER TO LONG CARDINAL, print 32-bit unsigned integer.

lx - Treat as POINTER to 32-bit quantity, print in hexadecimal (unsigned).

Special Characters

Characters preceded by an * are special, and have the following interpretations (upper-case is also allowed):

*n	(12B,newline):	Print a newline character.
*b	(10B,backspace):	Print a backspace character.
*t	(11B,tab):	Print a tab character.
*f	(14B,formfeed):	Print a new page (formfeed).
*nnn	(nnn is precisely three digits):	Print a char. whose ASCII code is Octal nnn.

Unrecognizable characters are simply printed, so * may be obtained by saying **.

User-Supplied Conversions

WriteFormatted maintains a table of 26 conversion routines, one for each letter of the alphabet. The string is scanned left to right and characters are output until a % is reached. The conversion code is used to select the routine to interpret the corresponding argument. Users can set a conversion escape using

```
SetCode[c: CHARACTER, p: PROCEDURE[d: UNSPECIFIED, form: STRING, wp:
    PROCEDURE[CHARACTER]]];
```

which sets character c to be interpreted by procedure p. d is the argument to be interpreted, form is the string of characters between the % and the character char (i.e. "-nnn"), and wp is the procedure that should be used to output characters (i.e. the prevailing output). Default conversions may be overwritten, and should be reset to their original when finished by

```
ResetCode[char: CHARACTER];
```

which resets the code of char to the default. In the case of multi-word items, the procedure must use a pointer to the items. The user-supplied conversion procedure may call WF for any output, as long as it does not use the conversion it is implementing!

Examples:

Print a simple string:

```
WF0["a tab *t is before a carriage return*n"];
```

Print the values of a, b, c, and a string:

```
WF4["print a string %s, followed by three numbers %d, %d, %d*n",str,a,b,c];
```

Use the formatting information to left justify and specify field widths:

```
WF2["%-14s is followed by %6d*n",s,a];
```

Use user supplied routines:

```
SetCode[’m,PrintStrange];

WF2["print a few strange things %m and %m*n",@StrangeThing1,@StrangeThing2];

ResetCode[’m];

WF0["now %m just prints an m"];
```

Print 32-bit quantities:

```
m: LONG INTEGER _ 50000;

WF2["Try %ld, %lb",@m,@m];
```

Other Procedures:

WFC[CHARACTER]; prints a single character.

WFCR[]; prints a carriage return.

WFN[STRING, DESCRIPTOR FOR ARRAY OF UNSPECIFIED]; is a generalization of WF0-4 that takes its arguments in an array.

Formatted Output to Strings and Streams:

Routines SWF0[] through SWF4[] and SWFN[] are available to put the output that would have gone to the termnal on a string. For example:

```
str: STRING _ [20];
SWF1[str,"one %d two",5];
```

will copy the string "one 5 two" onto string *str*. Any previous contents of *str* are lost.

Likewise FWF0[] through FWF4[] and FWFN[] take a StreamHandle parameter and do a stream "put" on each character. No other Stream operations are called. For example:

```
sh: StreamDefs.StreamHandle;
FWF1[sh,"one %d two",5];
```

will write the string onto the stream *sh*.

If you don't like WriteChar:

A procedure identifier is used by WF to output each character. If P and OP are procedures with a character as arguments, the procedure identifier used by WF can be changed via:

```
OP _ SetWriteProcedure[P];
```

so that P will be called for each character (SetWriteProcedure returns the previously used procedure). This can be used to write to other streams or to strings.

Calling

```
OP _ WriteToString[s: STRING];
```

will cause subsequent WFn calls to append things to string s (note the length is never reset and OP is the previously used procedure).

The statement

```
[] _ SetWriteProcedure[OP];
```

will reset to write back to the default stream.

Bugs:

Files: On [ivy]<cedarlib>writeformatted> you will find the source and object for WF and WFImpl.