Format strings

Format strings specify required formats to general-purpose formatting routines.

Description

Format strings passed to the string formatting routines contain two types of objects -- literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form:

```
"%" [index ":"] ["-"] [width] ["." prec] type
```

A format specifier begins with a % character. After the % come the following, in this order:

- An optional argument zero-offset index specifier (that is, the first item has index 0), [index ":"]
- An optional left justification indicator, ["-"]
- An optional width specifier, [width]
- An optional precision specifier, ["." prec]
- The conversion type character, type

The following table summarizes the possible values for type:

d — Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.

u — Unsigned decimal. Similar to 'd' but no sign is output.

e — Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point.

The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.

f — Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative.

The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.

g — General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present.

Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.

n — Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.

m — Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator, and CurrencyDecimals global variables or their equivalent in a

TFormatSettings data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals global variable or its TFormatSettings equivalent.

p Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.

s String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.

x Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.

For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables or their TFormatSettings equivalent.

Index, width, and precision specifiers can be specified directly using decimal digit string (for example "%10d"), or indirectly using an asterisk character (for example "%*.*f"). When using an asterisk, the next argument in the argument list (which must be an integer value) becomes the value that is actually used. For example,

Delphi example:

```
Format('%*.*f', [8, 2, 123.456]);
```

is equivalent to

```
Format('%8.2f', [123.456]);
```

C++ example:

```
TVarRec args[3] = {8,2,123.456};
Format("%*.*f", args, 2);
```

is equivalent to

```
TVarRec args[1] = {123.456};
Format("%8.2f", args, 0);
```

A width specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the format specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

An index specifier sets the current argument list index to the specified value. The index of the first argument in the argument list is 0. Using index specifiers, it is possible to format the same argument multiple times. For example "Format('%d %d %0:d %1:d', [10, 20])" produces the string '10 20 10 20'.

Note: Setting the index specifier affects all subsequent formatting. That is, Format('%d %d %d %0:d %d', [1, 2, 3, 4]) returns '1 2 3 1 2', not '1 2 3 1 4'. To get the latter result, you have must use Format('%d %d %d %0:d %3:d', [1, 2, 3, 4])