
Date, Time, and Numeric Utilities

This chapter describes date, time, and numeric utilities that Caché supplies. All of these utilities are character-based.

Date utilities perform conversions between expressions in internal date format used by \$H and expressions in printable formats. You can also use the \$ZD function to convert a date in internal format to one in printable format.

Time utilities perform similar conversions between printable time-of-day formats and the internal format used by \$H.

Other utilities print the current date and time.

Numeric utilities convert between decimal, octal, and hexadecimal number formats and find square roots.

This chapter also discusses the concept of a locale, a set of properties that control the display of the several data conversion functions discussed in this chapter.

Date and Time Utilities	page 11-2
Locales	page 11-9
Numeric Utilities	page 11-20

Date and Time Utilities

The Caché date and time utilities are listed and briefly described in Table 11-1. More details are provided in the following sections.

Table 11-1: Date and Time Utilities

Utility	Purpose
%D	Display the current date
%DATE	Convert dates to an internal format
%DO	Convert an internal date to an external format
%T	Display the current time
%TI	Convert the time to an internal format
%TIME	Display the current time, day, and date
%TO	Convert the time to an external format

Year 2000 Compliance

The Caché database management system is fully Year 2000 compliant. All dates used by the system itself are stored in a format that correctly represents and enables manipulation of dates in the twentieth and twenty-first centuries and beyond. In addition, all dates that are used by Caché but stored outside of a database, such as date stamps stored in database back up and transaction journal files, are stored in the same century-independent format.

Caché provides all support needed for full Year 2000 compliance at the application level. The Caché native date format correctly stores and enables manipulation of dates in the range January 1, 1841 to December 31, 9999. Caché also provides input and output conversion functions that correctly convert between this internal date format and external formats using four digit years.

Caché also includes several features to reduce problems associated with applications that currently display two digit year values. In many cases, these features enable applications to function correctly in the twenty-first century using two digit dates with little or no application change.

Displaying the Current Date

The %D utility displays the current date. Invoking %D at label INT returns the date in the variable %DAT instead of displaying it. For example:

```
%SYS>D ^%D
Nov 22 99
%SYS>D INT^%D

%SYS>W %DAT
Nov 22 99
%SYS>
```

Converting Dates to an Internal Format

The %DATE utility converts dates to an internal format: the number of days since December 31, 1840, the same format used by the \$H system variable. %DATE is very flexible about the input format; some examples of proper formats follow:

```
1/21
1/21/99
21/1/99
Jan. 21, 1999
January 21, 1999
21 Jan 99
21-JAN-99
T (today)
T+1 (tomorrow)
T-1 (yesterday)
T-5 (5 days ago)
```

In the input value, you can use any punctuation character in place of the space or slash (/) characters used in the above examples. You can also spell out the month in full, as in “JANUARY”, or abbreviate it to three characters, as in “JAN”. Unless you specify otherwise, the utility assumes the current year.

The utility returns the internal format of the date in variable %DN.

Invoking %DATE at label INT bypasses the operator dialogue. In this case, pass the external format of the date in variable %DS:

```
%SYS>D ^%DATE

Date: 4-5-99
%SYS>W %DN
57803
%SYS>S %DS="April 5,1999 D INT^%DATE
%SYS>W %DN
57803
```

Testing for Leap Years

You can use the `$$LEAP` entry point to `^%DATE` to test if a given year is a leap year:

```
$$LEAP^%DATE(year)
```

Return values are:

- 1 = year is a leap year
- 0 = year is not a leap year

Using `$ZDATEH` for Most Printable Formats

If you use the `$ZDATE` function (see below), you can convert any of its formats to `$HOROLOG` format quite easily using `$ZDATEH`. The exact action `$ZDATE` performs depends on the parameters you use.

- `$ZDATEH(date)` converts a date in the current locale default date format to the first integer in the `$HOROLOG` format (two or four digits as defined by the sliding date window.)
- `$ZDATEH(date,format)` converts a date in the specified *format* to `$HOROLOG` format. The *format* value is identical to the *format* values used by `$ZDATE`.

For more information about `$ZDATEH`, see the *Caché ObjectScript Reference*.

Sliding Window Support `%DATE` Utility Entry Points

The `%DATE` utility supports entry points for inspecting and/or modifying the system-wide default or process-specific sliding window definition. Each entry point defines an extrinsic function which returns the **old** sliding window definition (before the function takes any action) in the form of a string:

```
"OldStartDate^OldEndDate"
```

where *OldStartDate* and *OldEndDate* are absolute dates in `$H` format representing the start and end, respectively, of the **old** sliding window

As a quick rule of thumb for the entry points:

If the entry point name includes...	The entry point deals with...
Def	system-wide default sliding window definition.
Proc	process-specific default sliding window definition.
Date, AbsYear, RelYear	parameters defined for absolute \$H-format dates, absolute years, or relative years, respectively.

The value RETURNED by each extrinsic function is ALWAYS in terms of absolute \$H-format dates. This is necessary in order to provide for the interoperability of the three methods of defining sliding windows.

In the following description of each entry point, “system default” means the system-wide default sliding window definition and the term process window means the process-specific sliding window definition.

\$\$EmptyDefWindow^%DATE()

Set system default to be an “empty” window, so that **all** years display using 4 digits.

\$\$EmptyProcWindow^%DATE()

Set process window to be an “empty” window, so that **all** years display using 4 digits.

\$\$GetDefWindow^%DATE()

Get current system default definition.

\$\$GetProcWindow^%DATE()

Get current process window definition.

\$\$NoDefWindow^%DATE()

Remove current system default so that a 20th-century fixed window is used by default.

\$\$NoProcWindow^%DATE()

Remove process window so that Caché uses a 20th-century fixed window.

\$\$SetDefAbsYearWindow^%DATE(*StartYear*,*EndYear*)

Set system default to begin on the absolute year *StartYear* and end on the absolute year *EndYear*.

\$\$SetDefDateWindow^%DATE(*StartDate*,*EndDate*)

Set system default to begin on the absolute date *StartDate* and end on the absolute date *EndDate*.

\$\$SetDefRelYearWindow^%DATE(*StartYear*,*EndYear*)

Set system default to begin on the relative year *StartYear* and end on the relative year *EndYear*.

\$\$SetProcAbsYearWindow^%DATE(*StartYear*,*EndYear*)

Set process window to begin on the absolute year *StartYear* and end on the absolute year *EndYear*.

\$\$SetProcDateWindow^%DATE(*StartDate*,*EndDate*)

Set process window to begin on the absolute date *StartDate* and end on the absolute date *EndDate*.

\$\$SetProcRelYearWindow^%DATE(*StartYear*,*EndYear*)

Set process window to begin on the relative year *StartYear* and end on the relative year *EndYear*.

\$\$UseDefWindow^%DATE()

Set process window to be the same as the current system default

For example, suppose some time during 1999 you issued this command:

```
S X=$$SetProcRelYearWindow^%DATE(90,10)
```

The command would establish a process-specific sliding window definition which resulted in this process displaying two-digit years for any dates in the range 1/1/1909 through 12/31/2008, inclusive.

Converting an Internal Date to an External Format

The %DO utility converts the internal date in variable %DN to an external format returned in variable %DS. The %DO utility respects any current sliding window definition. For example:

```
%SYS>S %DN=$H D ^%DO
%SYS>W %DS
JUN 7 1999
%SYS>
```

You can use any of several internal entry points into %DO to produce different output formats, as listed below:

Table 11-2: Internal Entry Points into %DO

Entry Point	Result
^%DO	JUN 7 1999
INT^%DO	07 JUN 99
300^%DO	19990607
400^%DO	06/07/99

Using \$ZDATE for Most Printable Formats

For producing most printable formats you will find the \$ZDATE function easier to use and much faster than the %DO utility. The \$ZDATE function converts a specified date in \$HOROLOG format to one of several alternate date formats. The value returned by \$ZDATE depends on the parameters you use.

- \$ZDATE(*Hdate*) returns the date in a printable format that corresponds to the specified *Hdate*. *Hdate* is an integer value that is the number of days elapsed since December 31, 1840, and can range from 0 to 2980013 (12/31/1840 to 12/31/9999).
- \$ZDATE(*Hdate,format*) returns the date in the specified format.
- \$ZDATE(*Hdate,format[,picture]*) returns the date in the specified *format* and, optionally, *picture*.

For more information on \$ZDATE, see the *Caché ObjectScript Reference*.

Displaying the Current Time

The %T utility displays the current time. Invoking %T at label INT returns the time in the variable %TIM, instead of displaying it. For example:

```
%SYS>D ^%T
10:21 AM
%SYS>D INT^%T

%SYS>W %TIM
10:21 AM
```

Converting the Time to an Internal Format

%TI converts the time to an internal format: the number of seconds since midnight, which is the same format used in the system variable \$H. The utility returns the internal format of the time in variable %TN.

Enter the time as shown below:

```
11:00 AM
12:00 NOON
3:15 PM
```

You must enter the AM or PM, just A or P, or NOON or MID (or MIDNIGHT), but the space between the number and the letter is not required. 24-hour time is acceptable only without colons or other punctuation.

Use INT^%TI Entry Point to Bypass the Operator Dialogue

Invoking %TI with the entry point INT bypasses the operator dialogue. In this case, pass the external format of the time in variable %TS. For example:

```
%SYS>D ^%TI

TIME: 2:00 PM
%SYS>W %TN
50400
%SYS>S %TS="10:30A" D INT^%TI

%SYS>W %TN
37800
```

\$ZTIMEH Function

\$ZTIMEH converts a time value from a format produced by the \$ZTIME function to \$HOROLOG or \$ZTIMESTAMP (special variable) format. The exact value produce depends on the parameters you use.

- `$ZTIMEH(time)` converts a time in the form *hh:mm:ss[.ffff]* to \$HOROLOG format.
- `$ZTIMEH(time,format)` converts a time in the specified *format* to \$HOROLOG format. The *format* value is an integer value identical to the *format* value used by \$ZTIME.

For details on \$ZTIMEH, see the *Caché ObjectScript Reference*.

Displaying the Current Time, Day, and Date

The %TIME utility displays the current time, day and date. When you call %TIME on your CRT, the cursor moves in a square, clockwise pattern around the time and date information, updating the time every 60 seconds. Type an interrupt for your terminal (e.g., <CTRL-C>) to exit the %TIME utility.

When you type at the prompt:

```
D ^%TIME
```


the system responds with the current date and time:

```
** 02:15 PM **  
MONDAY  
22 Feb 99
```

As the cursor moves, it updates the time every minute.

Converting the Time to an External Format

The utility %TO converts the time, in seconds, to an external format. Pass the time as seconds since midnight in the variable %TN. The utility returns the external format in the variable, %TS. For example:

```
%SYS>S %TN=34390
```

```
%SYS>D ^%TO
```

```
%SYS>W %TS  
09:33AM  
%SYS>
```

\$ZTIME Function

\$ZTIME converts a internal system time from the specified \$HOROLOG-format value (*Htime*) to a printable format. The value returned depends on the parameters you use.

- `$ZTIME(Htime)` converts the value in *Htime* to a time in the format: "*hh:mm:ss*"
- `$ZTIME(Htime,format)` converts the value in *Htime* to the time format you specify in *format*.
- `$ZTIME(Htime,format,precision)` converts the value in *Htime* to the time format you specify in *format*. It also displays the seconds carried out to the number of decimal places you specify in *precision*.

For details on \$ZTIME, see the *Caché ObjectScript Reference*.

Locales

This section deals with locales that control data conversion functions. This differs from the National Language locale which defines properties based on character set. For information on National Language Support locales, see the *Caché Advanced System Management Guide*.

This is the list of data conversion functions included in the locales discussed in this section:

\$FNUMBER	\$ZDATETIME
\$INUMBER	\$ZDATETIMEH
\$ZDATE	\$ZTIME
\$ZDATEH	\$ZTIMEH

For information on the functions themselves, see the *Caché ObjectScript Reference*.

Definition of Locales

A locale consists of a list of properties that determine how data is displayed. The locale may be either public or private.

A public locale can be used by any Caché process. Currently there is only one public locale available. This is predefined by InterSystems. Future version will allow you to use locales associated with your National Language and choose the default locale using the NLS utility. Unless you specify a private locale for your process it will use the public locale by default.

A private locale is a locale that you design for a particular process or application. It can only be used by that process or application.

At any one time, a process has only one locale, which is either the public locale or a private locale.

The public locale is permanent. Its properties are defined by Caché and are immutable. In contrast, a private locale is temporary, existing only for the lifetime of a process. Its properties are defined for a process and can be further modified by that process at any time.

Limitations of the Locale Implementation

The following restrictions apply to Caché:

- The only possible value for LocaleName is "DEFAULT"
- \$\$DefDCLocale^%NLS always returns "DEFAULT"
- \$\$GetDCLocale^%NLS always returns either "DEFAULT" or "PRIVATE"

Invoking \$\$SetDCLocale^%NLS("DEFAULT") selects the single public shared locale for this process. This function is implicitly performed when any Caché process is created. You may need to set your application to explicitly invoke \$\$SetDCLocale^%NLS("DEFAULT") after using a private locale.

The properties defined in the default locale correspond to the standard default behavior of the data conversion functions before Caché 2.1. Applications requiring the traditional standard behavior of these functions need take no special action, since all processes initially use the default locale unless/until a private locale is created.

Manipulating Locales

An application determines its choice of locale and manipulates its locale properties by calling extrinsic functions which are packaged as entry points into the %NLS utility. The following functions are provided:

\$\$GetDCFormat^%NLS(*PropName*)

Returns the current setting of the property named *PropName* in the current locale. *Propname* must be enclosed by quotation marks unless it is a variable.

\$\$SetDCFormat^%NLS(*PropName*,*PropVal*)

Sets the property named *PropName* to the value *PropVal* and returns the old setting of that property. *Propname* must be enclosed by quotation marks unless it is a variable.

\$\$DefDCLocale^%NLS

Returns the name of the default locale table.

\$\$SetDCLocale^%NLS(*LocaleName*)

Sets the locale named *LocaleName* as the current locale for this process and returns the name of the previous locale used by this process.

\$\$GetDCLocale^%NLS

Returns the name of locale currently used by this process.

Although these values are not case sensitive, Caché will always return either “DEFAULT” or “PRIVATE” in all uppercase letters.

The %NLS extrinsic functions always return the current value for the Getxxx and Defxxx entry points, or the former value for Setxxx entry points.

Error Messages Returned by Locale Extrinsic Functions

If an error is detected, a negative number is returned as an error code. Currently, the possible error codes are:

-100	Invalid locale name or locale property name specified
-101	Attempt to set a locale property to an invalid value

Using Locale Extrinsic Functions

The Extrinsic functions can be used at the command line or within a routine. A complete list of the options available for these functions is in Table 11-3, “Locale Properties,” on page 11-14.

Example of Changing a Locale Setting

This example shows the current setting of plussign, the locale name and changes the setting of plussign which results in the locale changing to a private locale for this terminal session:

```
USER> w $$GetDCFormat^%NLS("plussign")
+
USER> w $$GetDCLocale^%NLS
DEFAULT
USER> w $$SetDCFormat^%NLS("plussign","*")
+
USER> w $$GetDCFormat^%NLS("plussign")
*
USER> w $$GetDCLocale^%NLS
PRIVATE
```

Examples of Setting Illegal Values

This example shows the error that results from specifying an illegal value for a valid property:

```
USER> w $$SetDCFormat^%NLS("plussign","PLUS")
-101
```

This command shows the error that results from specifying an illegal property name:

```
USER> w $$GetDCFormat^%NLS("xyz")
-100
```

Example of Resetting the Locale

This resets the terminal to the DEFAULT locale, effectively removing all changes made to the locale:

```
USER> w $$SetDCLocale^%NLS("DEFAULT")
PRIVATE
```

Creating Private Locales

An application creates a private locale when the application calls `$$SetDCFormat^%NLS()` to alter any property of its current locale. Subsequent calls to `$$SetDCFormat^%NLS()` by that process operate directly on that process's existing private locale.

Once a process creates its own private locale, it continues to use it unless an explicit call is made to `$$SetDCLocale^%NLS("DEFAULT")`.

Example of Private Locale Creation

This example shows routine code that creates a private local that changes the date display format from the default DD/MM/YY to European standard MM/DD/YY. A list of the locale properties you can alter is in Table 11-3, “Locale Properties,” on page 11-14.

```

LocaleCk      ; Routine for setting locale.
              ; This routine asks your prefer date format and sets the locale
accordingly
Ask           ; Asking for the data
READ "European or American date format (E/A)",DateSel
Set          ; Setting the locale
              IF (DateSel="E")!(DateSel="e") SET
x=$$SetDCFormat^%NLS("DateFormat",2)
              ELSE IF (DateSel="A")!(DateSel="a") SET
x=$$SetDCFormat^%NLS("DateFormat",1)
              ELSE WRITE !,"No such date format."

```

Locale Properties

The property names are not case sensitive. They may be specified directly as strings enclosed in quotation marks, as string expressions, or may be specified with Caché local or global variables.

The property names currently supported are described in the following table. The Default Value column indicates the property's value in the DEFAULT locale.

Table 11-3: Locale Properties

Property Name	Description	Default Value
AM	The suffix which may optionally be appended to time values prior to 12 noon.	“AM”
DateFormat	The DateFormat code used whenever a \$ZDATE, \$ZDATEH, \$ZDATETIME or \$ZDATETIMEH function call omits its second parameter, or specifies a second parameter value of -1. See “Format Code Descriptions”.	1
DateMaximum	The numeric value that specifies the upper limit of the range of valid dates used whenever a \$ZDATE, \$ZDATEH, \$ZDATETIME, or \$ZDATETIMEH function omits its maxdate parameter or specifies a maxdate parameter value of -1. See “Format Code Descriptions”.	2980013
DateMinimum	The numeric value that specifies the lower limit of the range of valid dates used whenever a \$ZDATE, \$ZDATEH, or \$ZDATETIMEH function omits its mindate parameter or specifies a mindate parameter value of -1. See “Format Code Descriptions”.	0
DateSeparator	The character used to delimit the months, days, and years of those dates which are otherwise entirely numeric. See “ODBC Date Format and Date Separator”.	“/”
DecimalSeparator	The character used by the \$FNUMBER and \$INUMBER functions to separate the whole part of a number from the fractional part of the number. See “Number Formatting” and “ODBC Date Format and Decimal Separator”.	“.”

Table 11-3: Locale Properties (Continued)

Property Name	Description	Default Value
Midnight	The suffix which may optionally be appended to time values corresponding to exactly 12 midnight.	"MIDNIGHT"
MinusSign	The character which may precede or follow a negative numeric quantity formatted by the \$FNUMBER or \$INUMBER function. See "Number Formatting".	"-"
MonthAbbr	The abbreviations of the names of the twelve months, used whenever a \$ZDATE, \$ZDATEH, \$ZDATEIME or \$ZDATETIMEH function call omits its MonthNameList parameter, or specifies a MonthNameList parameter value of -1, and the selected date format uses month name abbreviations (i.e., the effective DateFormat code is 0, 2, 5, 6, or 7).	See "Month and Date Names"
MonthName	The full names of the twelve months used whenever a \$ZDATE, \$ZDATEH, \$ZDATEIME or \$ZDATETIMEH function call omits its MonthNameList parameter, or specifies a MonthNameList parameter value of -1, and the selected date format uses full month names (i.e., the effective DateFormat code is 9).	See "Month and Date Names"
Noon	The suffix which may optionally be appended to time values corresponding to exactly 12 noon.	"NOON"
NumericGroupSeparator	The character used to separate groups of numeric digits within the whole number portion of a number formatted by the \$FNUMBER or \$INUMBER function. See "Number Formatting".	","

Table 11-3: Locale Properties (Continued)

Property Name	Description	Default Value
NumericGroupSize	The number of digits in a numeric group formatted by the \$FNUMBER or \$INUMBER function. See “Number Formatting”.	3
PM	The suffix which may optionally be appended to time values at or after 12 noon.	“PM”
PlusSign	The character which may optionally precede or follow a non-negative numeric quantity formatted by the \$FNUMBER or \$INUMBER function. See “Number Formatting”.	“+”
TimeFormat	The TimeFormat code used whenever a \$ZDATETIME, \$ZDATETIMEH, \$ZTIME or \$ZTIMEH function call omits its TimeFormat parameter, or specifies a TimeFormat parameter value of -1. See “Format Code Descriptions”.	1
TimePrecision	The number of fractional digits displayed in a time value whenever a \$ZDATETIME or \$ZTIME function call omits its Precision parameter, or specifies a Precision parameter value of -1.	0

Table 11-3: Locale Properties (Continued)

Property Name	Description	Default Value
TimeSeparator	The character used to delimit the hours, minutes and seconds in a time value. See “ODBC Date Format and Time Separator”.	“:”
WeekdayAbbr	The abbreviations of the names of the seven days of the week, starting with Sunday and ending with Saturday, used whenever a \$ZDATE or \$ZDATEIME function call selects a date format which uses weekday name abbreviations (i.e., the effective DateFormat code is 11).	See “Month and Date Names”
WeekdayName	The full names of the seven days of the week, starting with Sunday and ending with Saturday, used whenever a \$ZDATE or \$ZDATEIME function call selects a date format which uses full weekday names (i.e., the effective DateFormat code is 12).	See “Month and Date Names”
YearOption	The YearOption code used whenever a \$ZDATE, \$ZDATEH, \$ZDATETIME or \$ZDATETIMEH function call omits its YearOption parameter, or specifies a YearOption parameter value of -1. See “Format Code Descriptions” and “Year Option Code and Date Format”.	0

Month and Date Names

The default values for the month/weekday name abbreviations and full names are:

" Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec"

" January February March April May June ... November December"

" Sun Mon Tue Wed Thu Fri Sat"

" Sunday Monday Tuesday Wednesday Thursday Friday Saturday"

As with the `MonthNameList` parameter of the `$ZDATE` function, the first character of the above month/weekday strings specifies the delimiter which separates each of the names in the string. In the case of the month name abbreviations and full names, this same delimiter is also the delimiter used in the formatted date values in between the days, months, and years.

Format Code Descriptions

The `DateFormat`, `TimeFormat`, `maxdate`, `mindate`, and `YearOption` codes are described in the `$ZDATE` intrinsic function in the *Caché ObjectScript Language Reference*.

Number Formatting

When an `$FNUMBER` function call specifies a “.” as one of its format codes, that call returns “.” as its `NumericGroupSeparator` and “,” as its `DecimalSeparator`, regardless of the current locale’s properties. This call uses the `NumericGroupSize`, `PlusSign`, and `MinusSign` properties of the default locale (i.e., 3, “+”, “-”), regardless of the current locale. This results in European number formatting.

ODBC Date Format and Date Separator

The `$ZDATE`, `$ZDATEH`, `$ZDATETIME`, and `$ZDATETIMEH` functions ignore the “DateSeparator” character defined in the current locale when the effective `DateFormat` code is 3, because the ODBC standard dictates that the ODBC date format always uses a hyphen (“-”) as its separator character.

ODBC Date Format and Decimal Separator

The `$ZDATETIME` and `$ZDATETIMEH` functions ignore the `DecimalSeparator` character defined in the current locale when the effective `DateFormat` code is 3, because the ODBC standard dictates that the ODBC date format always uses a period (“.”) as its decimal separator character.

ODBC Date Format and Time Separator

The `$ZDATETIME` and `$ZDATETIMEH` functions ignore the `TimeSeparator` character defined in the current locale when the effective `DateFormat` code is 3 because the ODBC standard dictates that the ODBC date format always uses a colon (“:”) as its time separator character.

Year Option Code and Date Format

The “YearOption” code defined in the current locale only affects `$ZDATE`, `$ZDATEH`, `$ZDATETIME`, and `$ZDATETIMEH` function calls which have an effective `DateFormat` code of 0, 1, 2, 4, or 7. All other `DateFormat` codes produce a date whose year, if any, has 4 digits regardless of the effective `YearOption`.

This example shows how you could change the MonthAbbr property to use upper case instead of mixed case letters for the month name abbreviations.

```
>s x=$$SetDCLocale^%NLS("DEFAULT")
>f i=0:1:12 w !,?6,i,?12,$ZD(57191,i)
0      01 Aug 99
1      08/01/99
2      01 Aug 99
3      1999-08-01
4      01/08/99
5      Aug 1, 1999
6      Aug 1 1999
7      Aug 01 99
8      19990801
9      August 1, 1999
10     5
11     Fri
12     Friday
>w $$SetDCFormat^%NLS("MonthAbbr",
$ZCVT($$GetDCFormat^%NLS("MonthAbbr"),"U"))
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
>w $$GetDCFormat^%NLS("MonthAbbr")
JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
>f i=0:1:12 w !,?6,i,?12,$ZD(57191,i)
0      01 AUG 99
1      08/01/99
2      01 AUG 99
3      1999-08-01
4      01/08/99
5      AUG 1, 1999
6      AUG 1 1999
7      AUG 01 99
8      19990801
9      August 1, 1999
10     5
11     Fri
12     Friday
```

This example illustrates how the locale can alter the behavior of the \$FNUMBER function.

```
>s x=$$SetDCLocale^%NLS("DEFAULT")
>w $FN(123456789.987654321,"+")
+123456789.987654321
>w $FN(123456789.987654321,"+",")
+123,456,789.987654321
>w $$SetDCFormat^%NLS("PlusSign","*")
+
>w $$SetDCFormat^%NLS("DecimalSeparator",":")
.
>w $$SetDCFormat^%NLS("NumericGroupSeparator",";")
'
>w $FN(123456789.987654321,"+")
*123456789:987654321
>w $FN(123456789.987654321,"+",")
*123;456;789:987654321
>w $FN(123456789.987654321,"+."")
+123.456.789,987654321
```

Numeric Utilities

The Caché numeric utilities are listed in the following table. All these utilities are character-based.

Table 11-4: Numeric Utilities

Utility	Purpose
%DOCTAL	Convert a decimal into an octal
%DX	Convert a decimal into a hexadecimal
%OD	Convert an octal to a decimal
%SQROOT	Compute the square root
%XD	Convert a hexadecimal into a decimal

Converting a Decimal into an Octal

The %DOCTAL utility converts a string representing a numeric value in decimal notation into a string representing a numeric value in octal notation. It prompts you to type a numeric value in decimal notation, then displays that value in octal notation.

If you invoke the utility using the entry point `INT^%DOCTAL`, the utility converts a numeric value in decimal notation taken from the variable `%DO` into octal notation. `%DOCTAL` leaves the value in octal notation in `%DO`.

The following example demonstrates both methods of invoking the utility:

```
%SYS>D ^%DOCTAL

Decimal #: 15500      Octal: 36214
Decimal #: 4096       Octal: 10000
Decimal #: 500        Octal: 764
Decimal #: <RETURN>

%SYS>S %DO=1023 D INT^%DOCTAL

%SYS>W %DO
1777
%SYS>
```

Converting a Decimal into a Hexadecimal

The `%DX` utility converts a string representing a numeric value in decimal notation into a string representing the same numeric value in hexadecimal notation. It prompts you to type a decimal numeric value, then displays that value in hexadecimal notation.

If you invoke the utility using the entry point `INT^%DX`, the utility converts the string representing a numeric value in decimal notation found in the variable `%D` into a string representing the numeric value in hexadecimal notation. The utility then assigns the string in hexadecimal notation to the variable `%X`.

The following example demonstrates both methods of invoking the utility:

```
%SYS>D ^%DX
Decimal: 123          Hex: 7B
Decimal: <RETURN>

%SYS>S %D=456 D INT^%DX

%SYS>W %X
1C8
%SYS>
```

Converting an Octal to a Decimal

The `%OD` utility is used to convert octal to decimal notation. When you call `%OD`, it prompts you to type a number in octal notation and displays that value in decimal notation.

If you invoke %OD using the entry point INT^%OD, it converts into decimal notation a numeric value taken from the variable %OD in octal notation. The utility leaves the value in decimal notation in the variable %OD.

The following example demonstrates both methods of invoking the utility:

```
%SYS>D ^%OD

Octal #: 2400      Decimal: 1280
Octal #: 17325    Decimal: 7893
Octal #: <RETURN>
%SYS>S %OD=37777 D INT^%OD

%SYS>W %OD
16383
%SYS>
```

Computing the Square Root

The %SQROOT utility prompts for a numeric value for which the square root is computed. The root is then displayed on your terminal.

When you invoke the utility using the entry point INT^%SQROOT, the utility computes the square root of the value in the variable %X and leaves the result in the variable %Y. The variable %X must be defined.

The following example demonstrates both methods of invoking the utility:

```
%SYS>D ^%SQROOT
Square root of: 100      is: 10
Square root of: 10      is: 3.162277660168379332
Square root of: <RETURN>

%SYS>

%SYS>S %X=100 D INT^%SQROOT
%SYS>W %Y
10
%SYS>
```

Converting a Hexadecimal into a Decimal

%XD prompts you to type a hexadecimal numeric value and returns that value in decimal notation.

If you invoke the utility using the entry point INT^%XD, the utility converts the string representing a numeric value in hexadecimal notation found in the variable %X into a string representing the same numeric value in decimal notation. The utility then assigns the string in decimal notation to the variable %D.

The following example demonstrates how to use the utility:

```
%SYS>D ^%XD
Hex: 1CD          Decimal: 461
Hex: <RETURN>
%SYS>
```

Additional Math Functions

Caché includes a variety of additional math functions (listed in the following table) and a special variable.

The mathematical special variable is \$ZPI which contains the value of the constant Pi (3.141592653589793238).

Table 11-5: Mathematical Functions

Function	Purpose
\$ZABS	Returns the absolute value of any number
\$ZARCOS	Returns the trigonometric inverse cosine of a signed decimal number
\$ZARCSIN	Returns the trigonometric inverse sine of a signed decimal number
\$ZARCTAN	Returns the trigonometric inverse tangent of any number
\$ZCOS	Returns the trigonometric cosine of an angle in radians
\$ZCOT	Returns the trigonometric cotangent of an angle in radians
\$ZCSC	Returns the trigonometric cosecant of an angle in radians
\$ZEXP	Returns a value that is the natural logarithm (base e) raised to the specified power
\$ZLN	Returns the natural logarithm (base e) of any number
\$ZLOG	Returns the base 10 logarithmic value of an number
\$ZPOWER	Returns the value of a specified number raised to a specified power

Table 11-5: Mathematical Functions (Continued)

Function	Purpose
\$ZSEC	Returns the trigonometric secant of an angle in radians
\$ZSIN	Returns the trigonometric sine of an angle in radians
\$ZSQR	Returns the square root of any positive number
\$ZTAN	Returns the trigonometric tangent of an angle in radians