CA-Visual Objects
Version 1.0a
International Command/Function Reference

January 3, 1995

Overview

This file documents the CA-Visual Objects language revisions, effective as of release 1.0a, designed to support the Microsoft Windows internationalization standards (as specified from the Windows Control Panel, using the International icon). For the most part, this file documents additions to the language.

In cases where a language item has been significantly revised, however, new documentation is provided in this file. New items are marked with an asterisk in the heading. These supersede any printed or online document that you may have.

Using Write to View This Document

If you enlarge the Write window to its maximum size, this document will be easier to read. To do so, click the Maximize button in the upper right-hand corner of the window. Or, click on the system menu in the upper left-hand corner of the Write window (or press Alt+Spacebar), and then choose the Maximize command.

To move through the document, press Page Up or Page Down or click the arrow at the top or bottom of the scroll bar along the right side of the Write window.

To print the document, choose the Print command from the File menu.

For help using Write, press F1.

To read other on-line documents, choose the Open command from the File menu.

GetAMExt() Function

Purpose Returns a string representing the morning extension for time strings in 12-hour format.

Syntax GetAMExt() ---> cAMExt

Description Time strings formatted using the 12-hour time format have an extension to specify if the time is in the morning or the evening. GetAMExt() returns the current setting for the extension identifying morning time strings (that is, those between 12:00:00 midnight and just before 12:00:00 noon). A typical extension for morning time strings is AM, which is how the function derives its name.

The initial default for the morning extension is affected by SetInternational(), which you can refer to for more information. Use SetAMExt() to change the value of this setting.

Examples This example turns on the SetInternational() flag, causing the time string extensions to be determined by International settings in the Windows Control Panel:

```
SetInternational(#Windows)
? GetAMExt()  // Return value varies between systems
? GetPMExt()  // Typically, AM and PM are displayed
```

Prototype GetAMExt() AS STRING PASCAL

Library System Library

See Also GetPMExt(), SetAMExt(), SetAMPM(), SetInternational()

GetPMExt() Function

Purpose Returns a string representing the evening extension for time strings in 12-hour format.

Syntax GetPMExt() ---> cPMExt

Description Time strings formatted using the 12-hour time format have an extension to specify if the time is in the morning or the evening. GetPMExt() returns the current setting for the extension identifying evening time strings (that is, those between 12:00:00 noon and just before 12:00:00 midnight). A typical extension for evening time strings is PM, which is how the function derives its name.

The initial default for the evening extension is affected by SetInternational(), which you can refer to for more information. Use SetPMExt() to change the value of this setting.

Note: If SetInternational(#Windows) is in effect, the evening extension is also displayed as part of the 24-hour time format. This is a Windows standard that is not specific to CA-Visual Objects.

Examples This example turns on the SetInternational() flag, causing the time string extensions to be determined by International settings in the Windows Control Panel:

```
SetInternational(#Windows)
? GetAMExt()  // Return value varies between systems
? GetPMExt()  // Typically, AM and PM are displayed
```

Prototype GetPMExt() AS STRING PASCAL

Library System Library

See Also GetAMExt(), SetAMPM(), SetInternational(), SetPMExt()

LoadResString() Function

Purpose Look up an identifier in a string table and returns its corresponding string.

Syntax LoadResString([<*cDefault*>], <*nStringID*>, [<*xModule*>]) ---> cString

Arguments

<*cDefault*> The default string to return if <*nStringID*> cannot be found in the designated string table. Omitting this value is the same as specifying NULL_STRING.

<*nStringID*> The unique integer identifier that you want to look up. This value is defined when you declare the string table via RESOURCE...STRINGTABLE statement.

<*xModule*> The module containing the string table in which to perform the lookup. This argument can be specified either as a string, in which case it will be the name of a .DLL or .EXE file, or a WORD that identifies the module via its handle. If not specified, LoadResString() assumes the string table resides in the module that is currently executing.

Important! *The module that you specify here is not automatically loaded by the LoadResString() function. Therefore, prior to calling LoadResString(), you must either directly reference the module (for example, by calling one of its functions), or load the module manually using the Windows API function LoadLibrary().*

Description The LoadResString() function allows you to easily read information from a string table in your CA-Visual Objects application. String tables are defined via the RESOURCE statement using the STRINGTABLE keyword and are compiled, like all resource entities, by the Windows resource compiler.

In a CA-Visual Objects application, you can declare as many string table resource entities as you like; however, when the application is built, all of these are combined into a single string table, allowing only one table in each executable file or DLL. This means that within an application and across its library search path, string identifiers in string table resource declarations must be unique. Note, however, that another module (such as a DLL) referenced by an application can also have a string table, which is why LoadResString() allows you to specify which module to reference. Identifiers need only be unique *within* a module (that is, an .EXE or .DLL file).

Examples This example illustrates a technique for looking up an objects HyperLabel:Caption property using a string table. Using this technique facilitates internationalization of an application by isolating the natural language portion of the application that needs to be translated into a string table:[TBD PUT IN ALTERNATIVE WAY OF DOING THIS]

```
oDCMyControl:HyperLabel := HyperLabel{#MyControl, ;
               LoadResString("&File", MyFileString, MyLanguage),,}
```

```
DEFINE MyFileString := 100

GLOBAL MyLanguage AS STRING

METHOD Start() CLASS App
             ...
             IF ...      // it's French
                MyLanguage := "MYFRENCH.DLL"
             ELSEIF ... // it's German
                MyLanguage := "MYGERMAN.DLL"
             ENDIF

             RETURN NULL_STRING
```

The code for the MYFRENCH DLL string table would be as follows:

```
DEFINE MyFileString := 100

RESOURCE French STRINGTABLE
BEGIN
             MyFileString, "&Fichier"
             ...
END
```

The code for the MYGERMAN DLL string table would be as follows:

```
DEFINE MyFileString := 100

RESOURCE German STRINGTABLE
BEGIN
             MyFileString, "&Datei"
             ...
END
```

Prototype LoadResString(*cDef*, *ID*, *xModule*) AS STRING CLIPPER

Library System Library

See Also RESOURCE

SetAMExt() Function

Purpose Set the morning extension for time strings in 12-hour format.

Syntax SetAMExt(<*cAMExt*>) ---> VOID

Arguments

<cAMExt> The extension to be used for time strings that occur in the morning hours. This string can be up to eight characters in length—additional characters are ignored. The initial default depends on SetInternational(), which you can refer to for more information.

Description Time strings formatted using the 12-hour time format have an extension to specify if the time is in the morning or the evening. SetAMExt() allows you to programmatically set the extension identifying morning time strings (that is, those between 12:00:00 midnight and just before 12:00:00 noon). A typical extension for morning time strings is AM, which is how the function derives its name.

The initial default for the morning extension is affected by SetInternational(), which you can refer to for more information. Use GetAMExt() to retrieve the current value of this setting.

Examples This example turns on the 12-hour time format and displays a time string using two different extensions:

```
SetAMPM(TRUE)
SetAMExt(" am")
? TString(3800)  // 01:03:20 am
SetAMExt(" morning")
? TString(3800)  // 01:03:20 morning
```

Prototype SetAMExt(*cExt* AS STRING) AS VOID PASCAL

Library System Library

See Also GetAMExt(), SetAMPM(), SetInternational(), SetPMExt(), Time(), TString()

SetAMPM() Function

Purpose Return and optionally change the setting that determines whether time strings are in 12-hour or 24-hour format.

Syntax SetAMPM([<*lNewSetting*>]) ---> lCurrentSetting

Arguments

<lNewSetting> TRUE indicates 12-hour format. FALSE indicates 24-hour format. The initial default depends on SetInternational(), which you can refer to for more information.

Returns If <*lNewSetting*> is not specified, SetAMPM() returns the current setting. If <*lNewSetting*> is specified, the previous setting is returned.

Examples This example shows the difference between the 12-hour and 24-hour display formats:

```
SetPMExt(" pm")
SetAMPM(FALSE)
? Time()         // Result: 13:10:15 pm
```

```
SetAMPM(TRUE)
? Time()         // Result: 01:10:15 pm
```

Prototype SetAMPM(*l12h*) AS LOGIC CLIPPER

Library System Library

See Also SetAMExt(), SetInternational(), SetPMExt(), Time()

SetAnsi() Function*

Purpose Return and optionally change the setting that determines whether database files are created using ANSI or OEM format and whether certain text file operations convert between the two character sets.

Syntax SetAnsi([<*lNewSetting*>]) ---> lCurrentSetting

Arguments

<*lNewSetting*> TRUE specifies the ANSI format; FALSE specifies the OEM format. If the Ansi entry in the CA-Visual Objects section of WIN.INI is not present, an entry of Ansi=1 will be written, setting the initial default to TRUE; if this entry is set to 0, the initial default is FALSE.

Returns If <*lNewSetting*> is not specified, SetAnsi() returns the current setting. If <*lNewSetting*> is specified, the previous setting is returned.

Description SetAnsi() is a dual purpose function.

Database Files First and foremost, it determines the format used to create new database files—ANSI, which uses the Windows character set, or OEM, which uses the ASCII character set. Thus, functions such as DBCreate() and DBCopyStruct() are affected by this setting. The information that determines the ANSI/OEM format of a database file is stored in the header record of the file.

Important! *If your application must share data with a DOS-based product, either simultaneously or at different times, you must use the OEM format when creating .DBF files. (See Data Sharing in the "Using DBF Files" chapter of the Programmers Guide, Volume II.)*

If SetAnsi() is TRUE, your CA-Visual Objects application will perform an automatic conversion between OEM and ANSI for existing database files that are in OEM format, thereby preserving the format of the data and allowing it to be used by other applications that do not support the ANSI format.

Is SetAnsi() is FALSE, however, no automatic conversion is performed when reading to or writing from any database file. In this case, the data will appear at runtime just as it was read from disk. For OEM databases, therefore, the programmer is fully responsible for

ANSI/OEM conversion of displayed data or data modified by any edit control or terminal function. You can use the Ansi2Oem() and Oem2Ansi() functions to convert data between the two formats.

Text Files

The second purpose of SetAnsi() is to control how functions that read and write text files deal with the two different character sets. When SetAnsi() is FALSE, functions that read from text files (FReadText() and MemoRead()) and write to text files (FWriteText() and MemoWrit()) automatically convert between ANSI to OEM. When SetAnsi() is TRUE, these functions perform no conversion.

Independent of SetAnsi(), however, all data that is read or written using the FRead(), FWrite(), and related functions is not converted to and from the OEM character set.

Examples

This example stores the SetAnsi() setting in the beginning of an application and restores it to its original setting at the end:

```
METHOD Start() CLASS App
             LOCAL lAnsiSetting AS LOGIC
             lAnsiSetting := SetAnsi(FALSE)
             ...
             SetAnsi(lAnsiSetting)
```

Prototype

SetAnsi(*lSet*) AS LOGIC CLIPPER

Library

System Library

See Also

Ansi2Oem(), DBCopy(), DBCopyStruct(), DBCopyXStruct(), DBCreate(), FRead(), FReadText(), FWrite(), FWriteText(), MemoRead(), MemoWrit(), Oem2Ansi()

SetCollation() Function

Purpose

Return and optionally change the setting that determines the internal collation routine used for string comparisons.

Syntax

SetCollation([<*symNewSetting*>]) ---> symCurrentSetting

Arguments

<*symNewSetting*>

The collation mode to use. The available modes are #Windows (the default) and #Clipper.

Returns

If <*symNewSetting*> is not specified, SetCollation() returns the current setting. If <*symNewSetting*> is specified, the previous setting is returned.

Description

SetCollation() sets the internal collation routine that is used for all string comparisons, except the ones done using the == operator. Note that this includes sort and index operations, as well as programmatic string comparisons using the various operators.

Note: Changing SetInternational() automatically changes SetCollation() so that the two settings are the same.

This setting allows CA-Visual Objects to operate in different collation modes. The #Clipper mode is provided for compatibility with CA-Clipper applications and uses a collation routine defined in the nation module (CAVONT10.DLL). The #Windows mode uses string comparison services provided by Windows that automatically handle foreign character sets.

Therefore, if an application uses the #Clipper collation mode, it will behave the same on all machines. Thus, to achieve a different collation sequence based on a language other than English, you would need a version of CAVONT10.DLL specialized to the desired language. On the other hand, if the application uses the #Windows collation mode, it will behave differently from machine to machine, depending on the language defined in the International settings of the Control Panel. In this case, all languages supported by Windows are also supported by your application, including right-to-left languages, such as Hebrew and Arabic, and double-byte languages, such as Chinese, Japanese, and Korean.

Note: String functions, such as Substr() and SLen(), that operate at the byte level will not function correctly with double-byte characters.

The collation sequence for the regular Latin character set is different for #Clipper and #Windows. For #Clipper:

A < B < C < ... < Z < a < b < c < ... < z

For #Windows:

A < a < B < b < C < c < ... < Z < z

Warning! *SetCollation() determines how index files and the orders within them are created and maintained. Attempting to use different collation modes in the same order will corrupt the order.*

Examples

This example, based on the Start() method for the Standard Program, checks the current Windows language configuration, allowing the application to run only when set to use the French language collation:

```
METHOD Start() CLASS App
    LOCAL oWindow AS Window
    LOCAL nLen, nBufSize := 10 AS SHORTINT
    LOCAL pszLang := Psz(Space(nBufSize)) AS PSZ

    // Initialize StandardShellWindow
    Enable3DControls()
    oWindow := StandardShellWindow{SELF}
    oWindow:Show()

    // Retrieve language collation setting from WIN.INI
```

```
nLen := GetProfileString("intl", "sLanguage",     ;
   "", pszLang, nBufSize)

// Run application only if system configured to use
// French Windows collation
IF (Left(String(_CAST, pszLang), nLen) != "fra")
   // Incorrect language so give error and quit
   TextBox{oWindow, "System Configuration",      ;
      "You must have the Language set to " +     ;
      "'French' in the International " +        ;
      "section of your Control Panel to " +      ;
      "run this application."}:Show()
ELSE
   // Run application
   SELF:Exec()
ENDIF
```

Prototype SetCollation(*symID*) AS SYMBOL CLIPPER

Library System Library

See Also SET COLLATION, SetInternational()

SetInternational() Function

Purpose Return and optionally change the setting that determines the international mode for the application.

Syntax SetInternational([<*symNewSetting*>]) ---> symCurrentSetting

Arguments

<*symNewSetting*>
The international mode to use. The available modes are #Windows (the default) and #Clipper.

Returns If <*symNewSetting*> is not specified, SetInternational() returns the current setting. If <*symNewSetting*> is specified, the previous setting is returned.

Description SetInternational() allows CA-Visual Objects to operate in different international modes. The #Clipper mode is provided for compatibility with CA-Clipper applications and uses an internationalization routine defined in the nation module (CAVONT10.DLL). The #Windows mode uses international services provided by Windows.

Therefore, if an application uses the #Clipper international mode, it will behave the same on all machines. Thus, to achieve a different internationalization routine, you would need a specialized version of CAVONT10.DLL. On the other hand, if the application uses the

#Windows international mode, it will behave differently from machine to machine, depending on information defined in the International settings of the Control Panel. Changing SetInternational() automatically changes SetCollation() so that the two settings are the same. SetInternational() also determines the initial defaults for the functions listed in the table below and resets them each time you call the function. For SetInternational(#Windows), the settings are taken from the corresponding International settings in the Control Panel.

_

Function	**SetInternational(#Clipper) Default**
GetAMExt()/SetAMExt()	NULL_STRING
GetPMExt()/SetPMExt()	NULL_STRING
SetAMPM()	24-hour format
SetDateFormat()	mm/dd/yy
SetDecimal()	2
SetDecimalSep()	Period (.)
SetTimeSep()	Colon (:)
SetThousandSep()	Comma (,)

_

Notes

Leading zeros in time strings: The leading zeros International setting in the Control Panel is ignored. Leading zeros are always displayed for time strings.

Decimal and thousand separators in picture clauses: The comma as the thousands separator and the period used as the decimal separator in picture clauses is unaffected by this setting. Therefore, your source code will not change based on the status of SetInternational(). Note that picture clauses are defined with the @...SAY...GET command, the Transform() function, or the FieldSpec:Picture property.

@E picture function: The @E picture function is ignored if SetInternational(#Windows) is in effect.

Examples

This example sets the international mode to #Clipper in the applications startup routine in order to maintain backward compatibility with CA-Clipper applications. Note that SetCollation() is also changed to #Clipper as a result:

```
METHOD Start() CLASS App
           ...
           SetInternational(#Clipper)
           ...
```

Prototype

SetInternational(*symID*) AS SYMBOL CLIPPER

Library

System Library

See Also SET INTERNATIONAL, SetCollation()

SetPMExt() Function

Purpose Set the evening extension for time strings in 12-hour format.

Syntax SetPMExt(<*cPMExt*>) ---> VOID

Arguments

<*cPMExt*> The extension to be used for time strings that occur in the evening hours. This string can be up to eight characters in length—additional characters are ignored.

Description Time strings formatted using the 12-hour time format have an extension to specify if the time is in the morning or the evening. SetPMExt() allows you to programmatically set the extension identifying evening time strings (that is, those between 12:00:00 noon and just before 12:00:00 midnight). A typical extension for evening time strings is PM, which is how the function derives its name.

The initial default for the evening extension is affected by SetInternational(), which you can refer to for more information. Use GetPMExt() to retrieve the current value of this setting.

Note: If SetInternational(#Windows) is in effect, the evening extension is also displayed as part of the 24-hour time format. This is a Windows standard that is not specific to CA-Visual Objects.

Examples This example turns on the 12-hour time format and displays a time string using two different extensions:

```
SetAMPM(TRUE)
SetPMExt(" pm")
? TString(47000) // 01:03:20 pm
SetAMExt(" evening")
? TString(47000) // 01:03:20 evening
```

Prototype SetPMExt(*cExt* AS STRING) AS VOID PASCAL

Library System Library

See Also GetPMExt(), SetAMExt(), SetAMPM(), SetInternational(), Time(), TString()

SetTermAnsi() Function

Purpose Return and optionally change the display mode (ANSI or OEM) used by the terminal window.

Syntax SetTermAnsi([<*lNewSetting*>]) ---> lCurrentSetting

Arguments

<*lNewSetting*> TRUE specifies ANSI mode. FALSE specifies OEM mode, which is the default.

Returns If <*lNewSetting*> is not specified, SetTermAnsi() returns the current setting. If <*lNewSetting*> is specified, the previous setting is returned.

Description By default, the terminal window starts out in OEM mode. This is advantageous if you are migrating existing CA-Clipper applications to CA-Visual Objects because your screens will display in much the same was as they always have. For example, you will have no problems displaying various graphical characters, such as those used to draw boxes and borders, because these are part of the OEM character set.

However, if your application is intended for international use, this may not be the best setting, because the application must constantly convert data back and forth between ANSI and OEM. Most of this conversion is done automatically and will, therefore, not concern you. For example, data that is input via the terminal window is automatically converted, allowing users to enter international characters without any special processing by your application. Conversion of data from database files is also handled automatically, by default (controlled by SetAnsi() flag).

If you are dealing with data from other sources, however, such as text or binary files, you may find the need to convert the data before displaying it in the terminal window. Use Ansi2Oem() and Oem2Ansi() to accomplish this.

SetTermAnsi(TRUE) switches the terminal window display mode to ANSI, which means that all fonts used are ANSI. Using this setting makes your application take advantage of the Windows international language settings.

The only issue you need to be aware of when SetTermAnsi() is TRUE is that borders (or boxes) are drawn differently than when SetTermAnsi() is FALSE. In OEM mode, borders are drawn using the graphical drawing character set, which is not part of the ANSI character set. In ANSI mode, borders are drawn graphically and become elements in the terminal window queue. Visually, there is no difference; however, borders drawn in OEM can be overwritten (for example, you can erase all or a portion of the border by displaying spaces on top of it), whereas in ANSI mode, you must use CLS() or CRTEraseQElement() to get rid of a border.

SetTermAnsi() is an on-the-fly setting, so if you change modes, the content of the terminal window can be visibly affected. Changing SetTermAnsi() on-the-fly can even affect the size and look of the font in the terminal window if the needed font cannot be found. This is because the terminal window calls on the Windows font mapper, which always chooses the best possible match if it cannot find the font it is looking for.

Examples This example sets various international flags in the startup routine to fully support the Windows internationalization standards:

```
FUNCTION Start()
             ...
             SetInternational(#Windows)
             SetAnsi(TRUE)
             SetTermAnsi(TRUE)
             ...
```

Prototype SetTermAnsi() AS LOGIC

Library Terminal

See Also SetAnsi()

SetThousandSep() Function

Purpose Return and optionally change the setting that determines the thousands separation character to be used in numeric-to-string conversion functions.

Syntax SetThousandSep([<*nNewSetting*>]) ---> wCurrentSetting

Arguments

<*nNewSetting*> An ASCII code representing the new thousands separator. The initial default depends on SetInternational(), which you can refer to for more information.

Returns If <*nNewSetting*> is not specified, SetThousandSep() returns the current setting. If <*nNewSetting*> is specified, the previous setting is returned.

Description Normally, numeric values are displayed without thousands separators, unless you explicitly specify the number using pictured output. Pictured output is achieved using a picture clause, which can be defined with the @...SAY...GET command, the Transform() function, or the FieldSpec:Picture property.

Note: The comma character used in your source code to represent the thousands separator in picture clauses is unaffected by this setting, just as the period used to represent the decimal separator is unaffected by SetDecimalSep(). Therefore, your source code will not change based on these settings, as illustrated in the example below.

Examples This example changes the decimal separator to a comma and the thousands separator to a period:

```
FUNCTION Start()
             LOCAL nValue := 123456789.00
             SetDecimalSep(Asc(","))
             SetThousandSep(Asc("."))
             @ 10, 0 SAY nValue PICTURE "999,999,999.99
```

```
                    // Result: 123.456.789,00
```

Prototype SetThousandSep(*wChar*) AS WORD CLIPPER

Library System Library

See Also SetDecimalSep(), SetInternational()

Time() Function*

Purpose Return the system time in a format determined by various international settings.

Syntax Time() ---> cTimeString

Returns The format of the return value is dependent on several functions, including SetAMPM(), SetAMExt(), SetPMExt(), and SetInternational(), which you can refer to for more information.

Description Time() returns the system time. Time() is related to Seconds(), which returns the integer value representing the number of seconds since midnight.

Examples These examples show the results of Time() under various circumstances:

```
SetInternational(#Windows)
// Assuming 12-hour time with "AM" and "PM" extensions
// and colon separator set in Control Panel
? Time()                                  // 02:37:17 PM

SetInternational(#Clipper)
? Time()                                  // 14:37:19
SetAMPM(TRUE)                             // 12-hour format
SetPMExt(" pm")
? Time()                                  // 02:37:21 pm
```

Prototype Time() AS STRING PASCAL

Library System Library

See Also ElapTime(), Seconds(), SetAMExt(), SetAMPM(), SetInternational(), SetPMExt(), SetTimeSep(), Substr(), Today()

Time24() Function

Purpose Return the system time in 24-hour format.

Syntax Time24() ---> cTimeString

Returns The system time in the form *hh:mm:ss*, where *hh* is hours in 24-hour format, *mm* is minutes, and *ss* is seconds.

Description Time24() does not depend on any other setting to determine its output format. Instead, it returns the system time. Time() is related to Seconds(), which returns the integer value representing the number of seconds since midnight.

Examples These examples show the results of Time24() used with Substr() to extract the hour, minutes, and seconds:

```
? Time24()                                 // 10:37:17
? Substr3(Time24(), 1, 2)                        // 10
? Substr3(Time24(), 4, 2)                        // 37
? Substr3(Time24(), 7, 2)                        // 17
```

Prototype Time24() AS STRING PASCAL

Library System Library

See Also Time()

TString() Function*

Purpose Convert a specified number of seconds to a time string.

Syntax TString(<*nSeconds*>) ---> cString

Arguments

<*nSeconds*> The number of seconds to convert.

Returns The format of the return value is dependent on several functions, including SetAMPM(), SetAMExt(), SetPMExt(), and SetInternational(), which you can refer to for more information. If <*nSeconds*> is greater than 86,400 (the number of seconds in one day), the number of seconds past the whole number of days is returned (see the second example below).

Description TString() converts a specified number of seconds to a time string. Days() is a related function that is particularly useful when <*nSeconds*> is greater than 86,400, since it returns the number of full days in a given number of seconds.

Examples This example uses TString() on a number less than 86,400. The initial settings define a 12-hour time format with AM and PM as the morning and evening extensions:

```
SetAMPM(TRUE)
```

```
SetAMExt(" AM")
SetPMExt(" PM")
? TString(6400)                     // 01:46:40 AM
```

This example passes 303,600 to TString(). Since 303,600 seconds is equal to 3 days, 11 hours, and 20 minutes, TString() only returns the time past the whole 3 days. The initial settings define a 24-hour format with no morning or evening extension:

```
SetAMPM(FALSE)
SetAMExt(NULL_STRING)
SetPMExt(NULL_STRING)
? TString(303600)                         // 12:20:00
? Days(303600)                      // 3
```

Prototype TString(*uSeconds* AS USUAL) AS STRING PASCAL

Library System Library

See Also Days(), Secs(), SetAMExt(), SetAMPM(), SetInternational(), SetPMExt()

SET COLLATION Command

Purpose Change the setting that determines the internal collation routine used for string comparisons.

Syntax SET COLLATION TO WINDOWS | clipper

Description SET COLLATION is functionally equivalent to SetCollation(), which you can refer to for more information and examples.

Library System Library

See Also SET INTERNATIONAL, SetCollation()

SET INTERNATIONAL Command

Purpose Change the setting that determines the international mode for the application.

Syntax SET INTERNATIONAL TO WINDOWS | clipper

Description SET INTERNATIONAL is functionally equivalent to SetInternational(), which you can refer to for more information and examples.

Library System Library

See Also SET COLLATION, SetInternational()